Moab HPC Suite - Basic Edition

Adminstrator Guide 8.0.1

November 2014

Adaptive Computing Enterprises, Inc.
1712 S. East Bay Blvd., Suite 300
Provo, UT 84606
+1 (801) 717-3700
www.adaptivecomputing.com

Scan to open online help

Contents

Welcome

Welcome to Moab HPC Suite - Basic Edition 8.0.1!

The following sections will help you quickly get started with Moab HPC Suite:

Release Notes

Provides information on enhancements and fixes.

Installation and Configuration

Instructs on how to install, upgrade, and configure the Moab HPC Suite and components.

Component Functions and Procedures

Documentation, including references and tasks, for the components in the Moab HPC Suite. Documentation is grouped by the specific component.

- Moab Workload Manager
- Moab Web Services
- TORQUE Resource Manager

Moab HPC Suite Release Notes

Moab HPC Suite - Basic Edition 8.0.1 release notes

The release notes file contains the following sections:

New Features

The following is a summary of key new features in Moab HPC Suite - Basic Edition.

Moab Workload Manager

8.0.1

Added Support for "flags=allprocs"

You can now request one or more nodes and Moab will allocate all the processors on every node that is assigned to the job. For example, `qsub -l nodes=1,flags=allprocs`.

8.0.0

Data staging

The old method of data staging has been deprecated in Moab Workload Manager 8.0 and will be removed from the product in a future release.

Moab data staging has undergone a major redesign in 8.0. Consequently, Moab includes several new features that improve the data staging experience. These features include the following:

- Moab data staging system jobs, separately schedulable from the user job, that increase system performance and utilization by not reserving compute nodes during input or output data-staging unless a compute node's local file system is involved.
- New msub options that are required to submit data staging jobs. At least one of `--stagein`, `--stageinfile`, `--stageout`, and/or `--stageoutfile` is required to stage data for your job.
  - `--stagein` and `--stageout` specify individual files and/or directories to stage in or out, respectively.
  - `--stageinfile` and `--stageoutfile` specify the path to an individual file that contain the paths of files and/or directories to stage in or out, respectively.
- `--stageinsize` and `--stageoutsize` options that give Moab an estimate of the size of the files or directories to stage in or out, respectively, so it can more accurately schedule the associated data staging jobs. For more information, see **msub** on page 290 for more information.
- A customizable reference data staging job submit filter that sums the size of all input files and passes the sum to Moab. It uses the `--stageinsize` option so you do not have to estimate an input data size and Moab can more accurately estimate wallclock time when scheduling the input data staging system job. See **Configuring data staging** on page 884 for more information.
- Customizable reference scripts that use scp or rsync to stage data (`/opt/moab/tools/data-staging/ds_move_scp` and `/opt/moab/tools/data-staging/ds_move_rsync`, respectively). The scripts work in an out-of-the-box environment, but you should modify one or both to work with your unique environment. You can use the reference scripts as a guide to create your own script that supports data staging with a different Linux file transfer utility or commercial file transfer utility, such as one from Aspera.
- The ability to set generic metrics on partitions (for more information, see **Per-Partition Settings** on page 498).
  - The new `DATASTAGINGBANDWIDTH_MBITS_PER_SEC` metric, required for data staging, specifies the transfer rate of the partition in megabits per second. Moab uses it and the `--stageinsize` and/or `--stageoutsize` msub options submitted with the job to estimate the wallclock time for better scheduling of the input and output data staging system jobs.
  - The new metric can be dynamically updated by a resource manager or configured by an administrator.
- When you run checkjob `-v` on a data staging job, Moab returns the source and destination of data staging files and their total size. See **checkjob** on page 165 for more information.
- Advanced configuration options allow you to rename the default template, support multiple file transfer script utilities in a grid on a per-partition basis, configure a notification email to be sent after a data staging job completes, add a non-default template via msub, and use msub to return all job IDs in the workflow at submission time. This is useful when you dynamically generate user workflows using scripts that must create dependencies on data-staging system jobs (when input or output data-staging has completed, for example), not the user job. For more information, see **Configuring data staging with advanced options** on page 899.

The features associated with data staging in previous Moab releases have been deprecated in this release and will be removed in the next. For more information about data staging in Moab Workload Manager, see **About data staging** on page 880.

CPU frequency control

Moab can now ask the resource manager to change the CPU frequency on allocated nodes for submitted jobs. The request is made with the new CPUCLOCK resource manager extension. A user can specify a desired clock frequency in megahertz, a Linux power governor policy name, or an ACPI performance state (P-state) number. For more information, see **CPUCLOCK** on page 621.

The mjobctl-m command has been updated to allow modification of the requested CPU frequency on an already submitted job.

CPUCLOCK has also been added as a job template extension. The job template extension overrides the job script CPUCLOCK extension and the job submission CPUCLOCK option.

Improved Performance and Scalability

Moab 8.0 includes new multi-threaded scheduling routines and is now compiled with optimizations enabled (`-O`). These improvements should increase scalability, efficiency, and performance for the majority of sites. The size of the thread pool can be throttled using the THREADPOOLSIZE parameter.

Finer-grained Logging Timestamps

The timestamps in the Moab logs now include milliseconds, which can be helpful with higher LOGLEVEL settings.

Deleting checkpoint file doesn't reset IDs

Moab now persists counters for job and reservation IDs to an external file (`<moab home>/.counters`), in addition to persisting the counters to checkpoints. The new external counter file is created during installation. On startup, Moab reads the ID counters from either or both sources and uses the maximum of the two values that it sees as the resumption point for new IDs. This means that you can safely delete a checkpoint file without causing your IDs to be reset.

If both the external counter file and the checkpoint file are missing, Moab refuses to start. This is a precaution to protect the integrity of a relational database or external processes that rely on the IDs. If you need to override this behavior, you can do so by manually creating a new external counter file that starts IDs at an arbitrary number.

CANCELFAILEDDEPENDENCYJOBS scheduler flag

The new *CANCELFAILEDDEPENDENCYJOBS* scheduler flag automatically cancels dependency jobs that will never run because of an unmet requirement. For more information, see **Job Dependencies** on page 529.

Reduction in command processing time

If your system's scheduling cycle regularly takes longer than the **CLIENTTIMEOUT** value, you can configure Moab to fork a copy of itself that will respond to certain information-only client commands (checkjob, showbf, and showstart). This enables you to run intense diagnostic commands while Moab is in the middle of its scheduling process. For more information, see **Reduce command processing time** on page 1242.

New --workflowjobids option for msub

A new `--workflowjobids` option for the msub command returns all the job IDs, including data-staging system job IDs, in a data-staging workflow at submission time (Moab creates an internal workflow for jobs submitted with data-staging options). For more information, see **Configuring data staging with advanced options** on page 899.

The `support.diag.pl` script used for gathering information for Adaptive support representatives has been deprecated with the 8.0.1 release. Use the new `support-diag.py` script instead. For more information, see **Diagnostic Scripts** on page 689.

Moab Web Services

8.0.1

No new features.

8.0.0

PAM authentication support

Support for PAM (pluggable authentication module) has been added to MWS. PAM treats the user as if it is local to the machine doing the authenticating, and it uses whatever the user is authenticating with, whether it be LDAP or NIS. For information about configuring PAM with MWS, see **PAM (pluggable authentication module) configuration using mws-config.groovy** on page 1383.

> There is a security risk when authenticating local users through your PAM configuration. This behavior is highly discouraged and is not supported by Adaptive Computing.

OAuth authentication support

Support for OAuth has been added to MWS. OAuth allows trusted client applications to securely delegate authentication to MWS. Once MWS has authenticated a user by verifying the username and password in LDAP, PAM, or NIS, MWS returns an access token to the client. The client then presents this access token to MWS to access resources. For information about configuring PAM with MWS, see **OAuth configuration using mws-config.groovy** on page 1385.

Node power management plugin

The MWS power management plugin acts as a power resource manager; it monitors the power states of the nodes in the Node Configuration File using the TORQUE pbsnodes command and a query script. The plugin also allows you to customize the power state of each node when it should be "turned off." For more information, see the documentation included with the plugin in Moab Web Services.

Fairshare policy added to MWS Policies object

The Moab "fairshare" policy has been added to the Policies object in MWS. With this addition, you can get fairshare policy information and make fairshare policy modifications through the MWS API. For more information, see **Policies** on page 1589.

Modifiable attributes added to the MWS job resource

You can now use MWS to modify an idle job's system priority, the nodes that the job requests, and the resources per task required for the job. For more information, see **Modify job attributes** on page 1540.

View and modify single credentials

You can now query a single credential in MWS. Additionally, you can modify a credential's fields and use the change-mode parameter to modify the credential's list fields. For more information, see **Credentials** on page 1484.

View and modify scheduler priorities

You can now query and modify the scheduler priorities through the Priority resource in MWS. For more information, see **Priority** on page 1611.

TORQUE Resource Manager

5.0.1

Modification of the Output Location

TORQUE now allows for the modification of the output location based on the Mother superior hostname. An environment variable ($HOSTNAME) has been added to the job's environment.

5.0.0

CPU frequency control

TORQUE can now set the CPU frequency on requested nodes for submitted jobs. The request is made with the new cpuclock resource extension. A user can specify a desired clock frequency in megahertz, a Linux power governor policy name, or an ACPI performance state (P-state) number. For more information, see **cpuclock** on page 2238.

The qalter -l command has been updated to allow modification of the requested CPU frequency on an already submitted job.

The pass_cpuclock server parameter was added allowing administrators to track, but not grant, the CPU frequency request portion of a job submission. For more information, see **pass_cpuclock** on page 2432.

qrerun all command

When you execute the `qrerun all` command, you will be prompted for confirmation. TORQUE will then place all running jobs in a queued state without contacting the MOMs. You should only use this when the entire cluster is down and you cannot contact it.

Node power state control

TORQUE can now set the power state of a node. Depending on the hardware and software capabilities of the node, TORQUE can set the power state to Running, Standby, Suspend, Hibernate, or Shutdown. A new

-m option was added to the pbsnodes command to make this work. For more information, see **-m** on page 2358.

The syntax of the command is:

```
pbsnodes -m [running|standby|suspend|hibernate|shutdown] <space delimited list of
nodes to alter>
```

In order to wake a node from a low-power state, Wake-on-LAN must be supported and configured on the node. For more information, see **Changing Node Power States** on page 2270.

Differences

This section contains differences in previously existing features that require a change in configuration or routine.

Moab Workload Manager

8.0.1

MAXPROCPERNODE Can Also Be Configured Per Node

Added capability to configure MAXPROCPERNODE per class and per node. Before, MAXPROCPERNODE could be configured per class but it applied to all nodes. Now you can configure something like the following:

```
CLASSCFG[cpu] MAXPROCPERNODE[n1,n2,n3]=20 MAXPROCPERNODE[n4,n5,n6]=10
```

8.0.0

Green policy scripts might need updated values

If you do string comparisons in your green scripts, you might need to convert the incoming strings from Moab to all uppercase or lowercase.

mshow -a policy flag deprecated

The mshow -a policy flag (`mshow -a --flags=policy`) has been deprecated and may be removed in a future release.

Moab versions have been condensed to one build

There is now only one Moab build to download, install, and configure. In past releases, customers have had to choose between Moab builds that were compiled against ODBC and/or TORQUE libraries. Moab is still compatible with ODBC and TORQUE libraries, however there is now only one package to download.

Use of these libraries is enabled/disabled with the existing configuration parameters in `moab.cfg` (e.g., USEDATABASE ODBC).

CLASSCFG MAXPROCPERNODE no longer solely global

You can now specify individual nodes with the **CLASSCFG** **MAXPROCPERNODE** attribute, which limits the number of processors a class can have on a node.

```
CLASSCFG[batch] MAXPROCPERNODE[n1,n2,n3]=2
```

The batch class is limited to 2 processors on nodes n1, n2, and n3.

Complicated workflows contained in a single VC

Jobs that are created from a workflow template are now associated with a single VC (Virtual Container). Previously, for complex workflows, multiple VCs were created that each contained portions of the workflow jobs.

Moab Web Services

8.0.1

Node Power States

Moab only supports node power states of "On" and "Off". The API has been updated accordingly. See **Power Management Plugin** on page 1736 for details on supporting a larger set of node power states.

Reporting Plugins Are No Longer Turned On By Default

The reporting plugins (i.e., VMUtilizationReport and NodeUtilizationReport) are no longer turned on by default during installation. They are still available for use for those wishing to enable them.

Tenancy Plugin Removed

The MWS tenancy plugin was removed in 8.0.

8.0.0

API version 1 removed

API version 1 was deprecated in 7.5.0 and has been removed from Moab Web Services in this release. It is highly recommended that you use the most recent API version (version 3). See **Requesting specific API versions** on page 1406 for more information.

TORQUE Resource Manager

5.0.1

qmgr Server Parameter "copy_on_rerun"

A new qmgr option: set server copy_on_rerun=[True|False] is available. When set to True, Torque will copy the OU, ER files over to the user-specified directory when the qrerun command is executed (i.e a job preemption). This setting requires a pbs_server restart for the new value to take in effect. Note that the MOMs and the pbs_server must be updated to this version before setting copy_on_rerun=True will behave as expected.

qmgr Server Parameter "job_exclusive_on_use"

A new qmgr option: job_exclusive_on_use=[True|False] is available. When set to True, pbsnodes will report job-exclusive anytime 1 or more processors are in use. This resolves discrepancies between Moab and TORQUE node reports in cases where Moab is configured with a SINGLEJOB policy.

TORQUE Accounting Improvements

Two new fields were added to the accounting file for completed jobs: total_execution_slots and unique_node_count. total_execution_slots should be 20 for a job that requests nodes=2:ppn=10. unique_node_count should be the number of unique hosts the job occupied.

scan_for_terminated

Improved performance by moving scan_for_terminated to its own thread.

Port Using the Munge API Instead of Forking

TORQUE now uses the Munge API, rather than forking, when configured with the `--enable-munge-auth` option.

pbsdsh -o Option Captures Stdeer

The pbsdsh -o option is modified to add stdeer capturing.

5.0.0

The job_stat_rate parameter has a new default and function

Before this release, pbs_server asked the mother superior of every job for an update on the job every **job_stat_rate** seconds. The mother superior now sends updates on every job with its regular status, so there is no need for pbs_server to regularly poll. Instead, this parameter sets a timeout.

Two threadpools and new default for max_threads

The **max_threads** parameter has a new default: The value of min_threads ((2 * the number of procs listed in /proc/cpuinfo) + 1) * 20. In previous versions, 20 was 10.

Additionally, threadpools are now split. One-fourth of the threads are allocated for background tasks and three-fourths of the threads are allocated for incoming requests from MOMs and through the API (client commands, Moab, and so forth). Additionally, incoming requests no longer build up indefinitely. If a new request comes in, pbs_server evaluates whether it is too busy to service the request. For managers, the request is serviced as long as there are at least two threads available in the threadpool. For non-managers, the request is serviced as long as at least 5% of the threadpool is available. When pbs_server is too busy, it returns the error code PBSE_SERVER_BUSY with the message: "Pbs Server is currently too busy to service this request. Please retry this request."

Job status polling removed from TORQUE

Pbs_server now polls a mom for a job's information only if it hasn't received the information in 5 minutes. Otherwise, the information is communicated with the mom's status information.

TORQUE no longer searches linearly for the nodes in a node list

TORQUE now recognizes when a request to run a job specifies a node list. It directly accesses those nodes instead of searching for them linearly.

The exec_host list has one entry per node

The exec_host list has been condensed to contain one entry per node instead of one entry per execution slot. The node entry contains a string specifying each execution slot index.

TORQUE no longer displays the value of exec_port in qstat.

The qstat -f output for CPUs has been reduced

The output of qstat -f has been condensed to reduce clutter in the CPU section.

Installation and Upgrade Information

When installing or upgrading, it is *strongly* recommended that administrators configure Moab with mauth authentication with a complex key value. See **Appendix E: Security** on page 1195 for more information.

Moab 8.0.x supports TORQUE 4.2.8 and later. However, TORQUE 5.0.x requires Moab 8.0.x.

Moab has been tested with the latest versions of Postgres 9.3, MySQL 5.6, and Oracle 12c.

Installing Moab HPC Suite – Basic Edition 8.0.1

Please see **Requirements** on page 17 and also see **Preparing for installation** on page 24 and **Installing Moab HPC Suite - Basic Edition** on page 62 for manual or RPM-based installation instructions, respectively.

Installing TORQUE 5.0.x on RHEL 5

TORQUE requires the download of boost version 1.36 or later header files in order to build on most RHEL 5 installations. TORQUE 5.0.x needs the `unordered_map` from the boost library. This did not become part of the boost standard until boost version 1.36. Standard RHEL 5 installations only have boost version 1.33.1. TORQUE 5.0.x has added a configure option named `--with-boost-path`. This option allows the user to be able to download a newer version of boost without the need to install boost.

You can download a new boost library from `www.boost.org`. Extract the tarball to a directory of your choosing. When you are configuring TORQUE you then add the `--with-boost-path=<boost_path>`

to the configuration line. The `boost_path` is the location where you unzipped the boost library. Within the unzipped files is a directory named `boost`. This directory contains the `.hpp` files with the needed boost classes. Once the `--with-boost-path` option has been added to the configuration line you can continue to install TORQUE as per the installation guide.

> This procedure is not required on RHEL 6 and later versions.

Upgrading to Moab HPC Suite – Basic Edition 8.0.1

Please see **Preparing for upgrade** on page 52 and **Upgrading Moab HPC Suite - Basic Edition** on page 80 for manual or RPM-based installation instructions, respectively.

Moab Database Schema Updates

The Moab database schema has been updated for 8.0.

Table name	New additions	Changes
Nodes	• CPUClock (VARCHAR (64))	• Column name "Partition" changed to "PartitionName". The word "partition" became a reserved word as of MySQL 5.6
Requests	• CPUClock (VARCHAR (64))	n/a
NodeStats	• CPUClock integer • CPUMaxClock integer • CPUMinClock integer	n/a
Reservations	n/a	• Column name "Partition" changed to "PartitionName". The word "partition" became a reserved word as of MySQL 5.6

See for instructions on migrating your database schema.

Known Issues

The following are known issues in Moab HPC Suite - Basic Edition. Following each issue description is an associated issue number in parentheses.

Known issues are aggregated and grouped by the release version for which they first occurred or where reported.

8.0.1/5.0.1

- The node collection in the moab database in MongoDB has an index on the attributes field. This field can grow too large to index. (MOAB-7506)
  - For existing installations, the following commands on the MongoDB server will fix the problem:

```
$ mongo moab -u moab_user -p secret2
> db.node.dropIndex({"attributes":1})
```

    The username and password for your database are most likely different from the above example. Check with your database administrator.
  - For new installations using this and future releases, the index is no longer created and does not need to be dropped.
- Jobs submitted with invalid credentials are put in a held state, instead of rejected, until the administrator can respond. The checkjob command gives administrators further information regarding why the job is held. Blindly assuming that all held jobs should in fact be running RIGHT NOW is not only unsafe, but circumvents intentional Moab policies and workflow. An administrator should exercise care when resolving held jobs. (MOAB-7478)
- Some limitations exist in the way that pbsdsh can be used. Please note the following situations are not currently supported:
  - Running multiple instances of pbsdsh concurrently within a single job. (TRQ-2851)
  - Launching a large number of processes in succession (causes pbdsdsh to hang). (TRQ-2890)
- If the Moab **JOBNODEMATCHPOLICY** is set to *EXACTNODE* and `requirements.tasksPerNode` is used in a job submission to MWS, then Moab will double the resources requested. To avoid this problem, use `requirements.resourcesPerTask.processors.dedicated` instead. (MOAB-7424)
- BACKFILLPOLICY BESTFIT does not support multi-req jobs. Only FIRSTFIT supports multi-req jobs. (MOAB-6824)

8.0.0/5.0.0

- Connecting Moab to MySQL via ODBC can cause Moab to slow down. You can work around this issue by creating the tables with the old storage engine (ENGINE = MyISAM). Also note that other factors will also affect DB performance, such as the type of DB connection (remote vs. local), the DB server's hardware specifications, Moab's configuration for how much to write to the DB (see REALTIMEDBOBJECTS), etc (MOAB-6316).
- ~~Sites that used RPMs to install Moab HPC Suite will not be able to upgrade to 8.0.0 RPMs until 7.2.9 is released. The issue exists in all pre-8.0 RPMs and cannot be worked around without uninstalling the suite. Thank you for your patience as we work to resolve these upgrade issues. Until the issues are addressed, we suggest a fresh installation (either manual or RPM) on a~~

~~separate machine (MOAB-7192).~~ This issue has been resolved when upgrading from Moab HPC Suite 7.2.9.

- Verification and validation of the Moab 8.0 integration with Oracle 12c is not yet complete (AC-7407).
- When you submit jobs to Moab with a proxy user but no group specified via MWS, Moab uses the root group rather than the group of the proxy user (WS-2111).
- When you submit jobs with MWS, you must set the commandFile field to the absolute path of the job script on the MWS server. MWS must have read access to the file given in commandFile. Submitting remote job scripts (that is, including the script as part of the JSON payload) is not currently supported (WS-2112).
- Jobs submitted with invalid credentials are put in a held state, instead of rejected, until the administrator can respond. The checkjob command gives administrators further information regarding why the job is held. Blindly assuming that all held jobs should in fact be running RIGHT NOW is not only unsafe, but circumvents intentional Moab policies and workflow. An administrator should exercise care when resolving held jobs. (CVE-2014-5375, MOAB-7478, MOAB-7526)
- When installing or upgrading, it is *strongly* recommended that administrators configure Moab with mauth authentication with a complex key value. See **Appendix E: Security** on page 1195 for more information. (CVE-2014-5376, MOAB-7525, MOAB-7480)

Resolved issues

The following is a list of some key bugs fixed in Moab HPC Suite – Basic Edition. Following each issue description is an associated issue number in parentheses.

Resolved issues are aggregated and grouped by the release version in which they were resolved.

8.0.1/5.0.1

- **Submitting remote job scripts (that is, including the script as part of the JSON payload) is not currently supported.** Job scripts can now be included in the POST body when submitting jobs via MWS. The new field is called commandScript. (WS-2112)
- **Role permission updates were incorrectly applied to user permissions.** This issue is fixed. (WS-2340)
- **Moab was not decrementing GRES correctly within a Cray environment.** Fixed a GRES bug found in Cray environments. (MOAB-6577)
- **multireq jobs take hours to start.** Fix bug where multi-req jobs were slow to start in certain cases. (MOAB-6824)
- **When using .moab.key and mauth, having the string "actor=" in an environment variable in the job script causes mauth to fail.** Fixed bug in environment variable handling in mauth. (MOAB-6908)

- **Moab reserved extra nodes for generic resources requested in MAM.** Moab no longer reserves an extra node for MAM gres. (MOAB-6228)
- **Jobs were not rejected even though they violated fairshare policy.** This bug is fixed. (MOAB-7077)
- **support-diag.py does not processes backtrace information correctly.** This bug is fixed. (INT-808)
- **Script fails when more than one moab service is running on the same system.** Updated support-diag.py to better handle cases where more than one moab service is running simultaneously. (INT-811)
- **Moab changes task count on jobs on restart.** Fixed bug around incorrect task counts on multi-req jobs across a Moab restart in grid environment. (MOAB-7104)
- **Moab was still blocking on *checkjob* and *showres*.** Added *showres* to the list of commands available when UIMANAGEMENTPOLICY is set to FORK. (MOAB-7233)
- **ALERT log message did not end in a new line.** Added a missing new line character to a log message. (MOAB-7329)
- **Reservation was not being created when some resources are available.** Fixed an issue with standing reservations that occasionally weren't created when they should be. (MOAB-7384)
- **Moab XML submission not working.** Fixed some API bugs preventing job submission via XML (Moab Wire Protocol). (MOAB-7485)
- **Some bugs found that caused Moab 8.0 crashes.** Fixed bugs that were causing crashes.
- **Issue found with sqlite 8.0.0 integration.** Fixed an issue where some of the sqlite integration scripts were not being packaged properly.
- **Moab was unable to modify the hostlist of a idle job correctly.** Fixed bug with modifying hostlists on idle jobs. (MOAB-7490)
- **nallocpolicy=cpuload was being ignored.** Fixed bug where requested nallocpolicy was inadvertantly getting overridden. (MOAB-7443)
- **UIMANAGEMENTPOLICY caused Moab to hang.** Fixed a bug where UIMANAGEMENTPOLICY caused Moab to hang. (MOAB-7578)
- **Job dependencies were not consistently working as expected.** Fixed bug related to job dependencies getting out of sync with TORQUE in some cases. (MOAB-7472)
- **Performance bug found in ReportJob.groovy.** Fixed scalability issue with large sample sets (e.g., node utilization). (WS-2273)
- **Services stuck in Deploying state.** Fixed bug where some services were getting stuck in Deployed state. (WS-2293)
- **Slow performance reported on large LDAP systems.** Fixed bug with LDAP integration to improve performance on large LDAP systems. (WS-2252)
- **Torque job can kill processes not owned by the job owner.** Fixed issue around unauthorized termination of processes. (CVE-2014-3684, TRQ-2885)

- **qstat -Q <bad_queue_name> prints queue name twice.** Fixed bug where giving a bad queue name to qstat -Q results in duplicate output. (TRQ-2025)
- **Mail output for qsub -m options was failing to output new lines.** Fixed bug in qsub -m when TORQUE is configured --with-sendmail. Some missing newlines were added. (TRQ-2937)
- **mppnodes hostlist was being re-ordered.** This bug is fixed. (TRQ-2112)
- **Some tasks were incorrectly listed as 0 in qstat -a when requested specific nodes.** This has been resolved. (TRQ-2292)
- **TORQUE accounting problems - jobs without accounting records.** Fixed bug related to accounting records on large systems. (TRQ-2367)
- **qstat wouldn't parse anything after a bad job ID.** Improved qstat behavior in cases where bad job IDs were referenced in the command. (TRQ-2410)
- **Separate headers for multiple jobs IDs provided to qstat.** Fixed output format bug in cases where multiple job IDs are passed into qstat. (TRQ-2411)
- **qsub did not process arguments correctly when a submitfilter is used.** Fixed bug where qsub did not process args correctly when using a submit filter. (TRQ-2646)
- **Parsing bug found when using hostlist ranges in qsub.** This bug is fixed. (TRQ-2652)
- **Build bug reported with MIC libraries.** Fixed build bug related to newer Intel MIC libraries installing in different locations. (TRQ-2653)
- **XML job log error.** Corrected mismatched <Job_Id> XML tags in the job log. (TRQ-2692)
- **TORQUE was not dividing GPUS amongst NUMA nodes.** Fixed problem where GPUs were not split between NUMA nodes. You now need to specify which gpus belong to each node board in the mom.layout file. (TRQ-2730)

  A sample mom.layout file might look like

  ```
  nodes=0 gpu=0
  nodes=1 gpu=1
  ```

  This only works if you use nvml. The nvidia-smi command is not supported.
- **TORQUE was leaving behind error and out files when a job was preempted or requeued.** Fixed bug where OU files were being left in spool when job was preempted or requeued. (TRQ-2732)
- **Reported cput was incorrect.** Fixed bug where reported cput was incorrect. (TRQ-2759)
- **pbsnodes -l offline -n unexpected error.** Fixed unexpected error when running `pbsnodes -l offline -n`. (TRQ-2760)
- **max_user_queuable limit reached, however, there were no jobs in the queue**. Fixed bug where jobs rejected due to max_user_queuable limit reached, yet no jobs in the queue. (TRQ-2795)
- **momctl -q clearmsg didn't seem to clear error messages permanently.** Fixed bug where `momctl -q clearmsg` didn't properly clear error messages. (TRQ-2828)
- **Some bugs found that caused TORQUE core crashes.** These bugs are fixed.

- **TORQUE was crashing.** Fixed crashing.
- **pbs_server segfault after large array deletion.** Fixed segmentation fault. (TRQ-2835)
- **GPU nodes where not passed to sister nodes.** This bug is fixed. (TRQ-2837)
- **pbs_server did not write resource_default units to serverdb file.** This bug is fixed. (TRQ-2852)
- **pbs_mom filling up the logs in a HA environment.** Reduced verbosity in error logging in HA environments. (TRQ-2863)
- **Make trqauthd error messages more meaningful and non-repetitive.** Improved trqauthd error messages to be more meaningful and less redundant. (TRQ-2882)
- **Remote job submissions were being rejected, even when host is in submit_hosts list.** Fixed problem with remote client job submission during ruserok() calls. (TRQ-2918)
- **pbsdsh did not support running multiple instances concurrently.** Improved pbsdsh to better handle simultaneous use of -o and -s options. Also fixed some problems where -o output was sometimes getting truncated. (TRQ-2890)
- **TORQUE was not notifying Moab of completed jobs.** Fixed bug where TORQUE was not honoring KeepCompleted server parameter when job_nanny was set to true. (TRQ-2904)
- **HOST_NAME_SUFFIX was no longer adding suffix to job names.** This bug is fixed. (TRQ-2956)
- **Deadlock when running 'qdel -p' as non-root user.** Fixed deadlock issue. (TRQ-2919)

8.0.0/5.0.0

- **Moab attempted to power off drained nodes.** Moab now skips drained nodes when it powers off nodes. (MOAB-6036)
- **Moab Authentication Bypass issue.** This has been fixed. (CVE-2014-5300, MOAB-7100, MOAB-7524)
- **In environments where a license RM was configured and created a global node, Moab would try to power the global node on and off.** Now Moab does not allow the global node to be powered down as part of a pool of eligible nodes. (MOAB-6608)
- **The mdiag -n command did not return the correct amount of available memory on the node.** Running mdiag -n once again returns the correct amount of available memory. (DOC-6708)
- **Moab did not honor features configured in the DEFAULTNODEFEATURES parameter.** The DEFFAULTNODEFEATURES parameters once again works as expected.
- **When Moab would rerun a job after the job migrated to another partition, it attempted the rerun on the wrong partition.** Moab reruns migrated jobs in the correct partition. (MOAB-6734)
- **When using a JOBMIGRATEPOLICY of *IMMEDIATE*, Moab did not call the CREATEURL Native Accounting Manager Interface script.** Moab now calls the CREATEURL NAMI script. (MOAB-6867)
- **Running msub and qsub in an interactive environment produced different results.** Running msub -I no longer hangs in an interactive environment. (MOAB-7165)
- **Moab ignored requirements set using the "hostlist" parameter.** Moab now honors hostlist requirements. (MOAB-6475)

- **When the queue contained more than 50,000 jobs, TORQUE slowed down substantially.** TORQUE no longer slows down with a large number of jobs in the queue. (TRQ-2345)
- **When pbs_server had a high load, it would get stuck polling.** pbs_server no longer gets stuck polling under these conditions. (TRQ-2620)
- **When a job with a dependent job was deleted with qdel, TORQUE did not clean up the dependent job.** TORQUE now removes the dependent job when you delete its parent. (TRQ-2621)
- **When a lot of jobs were run at once, one job would get stuck in an exiting state.** This error no longer occurs. (TRQ-2622)
- **The afterok dependency did not work as expected.** afterok now works correctly. (TRQ-2626)
- **When running a large amount of jobs, the server would crash during job recycle.** The crash no longer occurs. (TRQ-2628)
- **pbsdsh requires FQDN even if other elements don't.** pbsdsh no longer requires FQDN. (TRQ-2632)
- **A deadlock would occur on job_save failure.** This error no longer occurs. (TRQ-2645)
- **Asynchronous job starts queued in TORQUE but not yet serviced caused Moab to reschedule jobs that would eventually run.** A new job sub-state in TORQUE prevents this from occurring. (TRQ-2715)

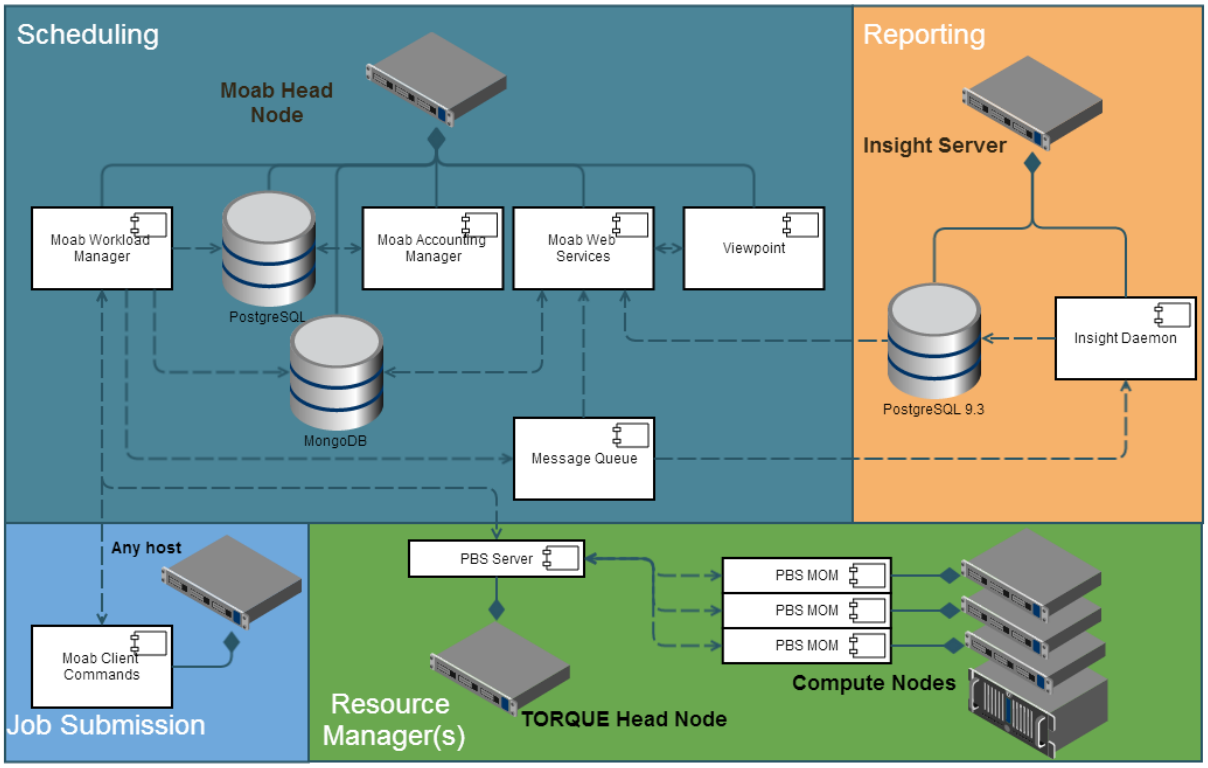
Installation and Configuration

Requirements

> It is highly recommended that you *first* perform installations and upgrades in a *test environment*. Standard installation and upgrade procedures and use cases are tested prior to release. However, due to the wide range of possible configurations and customizations, it is important to exercise caution when deploying new versions of software into your production environments. This is especially true when the workload has vital bearing on your organization's day-to-day operations. We recommend that you test in an environment that mirrors your production environment's configuration, workflow and load as closely as possible. Please contact your Adaptive Computing account manager for suggestions and options for installing / upgrading to newer versions.

There are many different ways to install and configure Moab HPC Suite. Each environment has its own set of requirements and preferences. The following installation instructions are intended to help an administrator understand how each of the Moab HPC Suite components interact and how to install and configure each one. Two approaches have been documented: the “Manual installation” and the “RPM installation”. Only one approach is required for installation; do not try to follow both sets of instructions on a single system.

The diagram below provides a general topology of the Moab HPC Suite.



Please note the following:

- Moab Accounting Manager is available only with the Moab HPC Enterprise Suite.
- Moab 8.0.0 and later supports TORQUE 4.2.8 and later. However, TORQUE 5.0.0 and later requires Moab8.0.0 and later.
- Smaller environments may elect to consolidate the TORQUE Head Node with the Moab Head Node, including PBS Server in the list of components installed on the Moab Head Node.
- The Requirements section gives further clarification regarding what each component requires.
- Although Moab Workload Manager and Moab Accounting Manager may share the same database instance, it is not a requirement. Two database instances may be used, one for each component. See the Requirements section for more information about what databases are supported.
- The Message Queue component is fulfilled by ZeroMQ™. The libraries are provided with the components that use the message queue and are enabled via configuration; no special installation is necessary.

Where to Start

1. Begin by reading the Requirements section below. Whether installing manually or with RPMs, it is important to be familiar with the hardware and software requirements.
2. Decide whether you will perform a manual installation or an RPM installation.

The manual installation provides advantages to administrators who want to pick and choose what components to install and administrators who want non-standard configure options. The RPM installation provides advantages to administrators who want a fairly standard installation.

> Code samples have been provided for convenience. Some code samples provide sample passwords (i.e. "changeme!"). We strongly recommend that you do not use these passwords during installation, as using the documented passwords could introduce unnecessary security vulnerabilities into your system.

Then follow the appropriate installation instructions.

3. The "Additional Configuration" section in both the manual and the RPM installation instructions provide additional information and instructions for optional, but recommended configurations (i.e. Configuring SSL in Tomcat, etc.).
4. Refer to **Troubleshooting** on page 87 for assistance in addressing common problems during installation and configuration.
5. Refer to **Component documentation** on page 95 for links to additional administrator and reference guides.

Requirements

Moab HPC Suite

Hardware Requirements

The following are the minimum hardware requirements for an average environment. Larger environments should consider allocating more resources and/or spreading components across multiple servers. Please consult the table below for recommendations:

Type	# of Compute Nodes	Jobs/ week	Minimum Requirements (combine TORQUE & Moab head nodes on one server)	Recommended Requirements (targeting minimum number of servers)
Proof of Concept / Small Demo	50	<1k	**Moab+TORQUE Head Node:** • 4 Intel/AMD x86-64 cores • At least 8 GB RAM • At least 100 GB dedicated disk space	Same as minimum
Medium	500	<100k	**Moab+TORQUE Head Node:** • 8 Intel/AMD x86-64 cores • At least 16 GB RAM • At least 512 GB dedicated disk space	**Moab+TORQUE Head Node:** • 16 Intel/AMD x86-64 cores • At least 32 GB RAM • At least 1 TB dedicated disk space

Type	# of Compute Nodes	Jobs/ week	Minimum Requirements (combine TORQUE & Moab head nodes on one server)	Recommended Requirements (targeting minimum number of servers)
Medium with High Throughput or Larger	>500	>100k	**Moab Head Node:** • 8 Intel/AMD x86-64 cores • At least 16 GB RAM • At least 512 GB dedicated disk space **TORQUE Head Node:** • 8 Intel/AMD x86-64 cores • At least 16 GB RAM • At least 512 GB dedicated disk space	We recommend separating components onto separate servers where possible (some components should not be separated; see Requirements below). Specific requirements around the intended configuration and use of Moab HPC Suite will help determine suite topology and resource allocation.

Please note the following:

- All requirements above (minimum and recommended) target a minimum number of management servers. Administrators are encouraged to separate the TORQUE and Moab head nodes where possible for better results, especially when High Throughput is enabled.
- Although many factors may have an impact on performance (network bandwidth, intended use and configuration, etc.), we consider High Throughput as something that makes a significant enough difference between minimum and recommended hardware requirements to merit mention in the table above.
- Moab and TORQUE are both multi-threaded and perform better with more processors.
- Regarding disk space, consideration should be given to requirements related to log files, log depth, number of jobs/nodes/reservations (more objects impact database journal size), average number of events generated (more events take more space), etc.

Software Requirements

The installation documentation provides more details regarding how to install and configure the following software requirements. The information provided below is for your information only. No action is necessary.

Software requirements are listed per-component rather than suite-wide to make it easier for administrators who wish to install components on separate servers.

TORQUE

Supported Operating Systems

- CentOS 6.5 or later
- Red Hat 6.5 or later
- Scientific Linux 6.5 or later
- SUSE Linux Enterprise Server 11 SP3 or later

> CentOS 5.9, Red Hat 5.9 and Scientific Linux 5.9 are supported, largely to continue support for clusters where the compute nodes operating systems cannot be upgraded. We recommend that the TORQUE head node run on the supported operating systems listed above.

Software Requirements

- libxml2-devel package (package name may vary)
- openssl-devel package (package name may vary)
- Tcl/Tk version 8 or later if you plan to build the GUI portion of TORQUE or use a Tcl based scheduler
- If you use cpusets, libhwloc 1.1 or later is required (for TORQUE 4.0.0 and later)

If you build TORQUE from source (i.e. clone from github), the following additional software is required:

- gcc
- gcc-c++
- A posix compatible version of make
- libtool 1.5.22
- boost-devel 1.36.0

Moab Workload Manager

Supported Operating Systems

- CentOS 6.5 or later
- Red Hat 6.5 or later

- Scientific Linux 6.5 or later
- SUSE Linux Enterprise Server 11 SP3 or later

Software Requirements

- libcurl
- Perl 5.8.8 or later
- perl-cpan (package name may vary)
- libxml2-devel (package name may vary)
- *(Optional)* MySQL, PostgreSQL, or Oracle with ODBC driver (see **Database Configuration** on page 772 for details)

Supported Resource Managers

- TORQUE 5.0.1
- SLURM

Moab Accounting Manager

MAM is commonly installed on the same host as Moab Workload Manager; however, in some cases you might obtain better performance by installing them on separate hosts.

Supported Operating Systems

- CentOS 6.5 or later
- Red Hat 6.5 or later
- Scientific Linux 6.5 or later
- SUSE Linux Enterprise Server 11 SP3 or later

Software Requirements

- gcc
- perl-suidperl
- httpd
- mod_ssl
- rrdtool
- Moab Workload Manager 7.2 or later
- Perl modules

Depends On (not necessarily on the same server)

MAM uses an RDBMS as a back end.

- PostgreSQL 7.2 or later

Moab Web Services

MWS should be installed on the same host as Moab® Workload Manager.

Supported Operating Systems

- CentOS 6.5 or later
- Red Hat 6.5 or later
- Scientific Linux 6.5 or later
- SUSE Linux Enterprise Server 11 SP3 or later

Software Requirements

- Moab® Workload Manager 8.0.1
- Apache Tomcat™ 6
- Oracle® Java® 7 Runtime Environment

  > Oracle Java 7 Runtime Environment is the recommended Java environment, but Oracle Java 6 is also supported. All other versions of Java, including OpenJDK/IcedTea, GNU Compiler for Java, and so on cannot run Moab Web Services.

- MongoDB® 2.4.x

Depends On (not necessarily on the same server)

- OpenLDAP or PAM (see **Configuring Moab Web Services** on page 1373)

Manual installation

Installation

Preparing for installation

The installation process of the Moab HPC Suite includes installing the separate components in the suite. This guide contains detailed instructions for installing each component.

> Many individual components have dependencies on other components (see **Requirements** on page 17). However, if you do not require a certain component (Moab Web Services, for example), you do not have to install it.

The install instructions for each component include information about system requirements and dependencies. Some include prerequisite instructions that you will need to complete before you begin the install. Please read this information carefully, and make sure you have installed all the dependencies and packages that are necessary in order to avoid errors during the Moab HPC Suite install process.

Because many system-level files and directories are accessed during the installation, the instructions in this guide should be executed with root privileges.

You will see that the instructions execute commands as the root user. Please note that the same commands will work for a non-root user with the `sudo` command.

To install the Moab HPC Suite, install the packages in the following order:

1. Install the TORQUE and Moab Workload Manager dependencies (see the dependency installation instructions below).
2. Install TORQUE (see **Installing TORQUE** on page 2193).
3. Install Moab Workload Manager (see **Installing Moab Workload Manager** on page 34).
4. Install Moab Web Services (see **Installing Moab Web Services** on page 38).

Install TORQUE and Moab Workload Manager dependencies

You must install the following dependencies in order to use TORQUE and Moab Workload Manager:

- libxml2-devel package (package name may vary)
- openssl-devel package (package name may vary)
- boost-devel package (package name may vary)
- ANSI C compiler. The native C compiler is recommended if it is ANSI; otherwise use gcc.

Use the following commands to install the required dependencies and packages.

- **RHEL, CentOS, and Scientific Linux**:

```
[root]# yum install make perl-CPAN libxml2-devel openssl-devel boost-devel gcc
gcc-c++
```

- **SLES**:

  Before installing the dependencies, do the following:

  1. Verify that you have a licensed installation of SLES 11 SP3.
  2. Download the SuSE Linux Enterprise 11 Software Development Kit e-Media Kit and add the ISO to the repository.

```
[root]# zypper install make libxml2-devel libopenssl-devel boost-devel gcc gcc-
c++ git-core automake
```

Install Java

You must install the 64-bit RPM version of Oracle® Java® 7 Runtime Environment if you are installing Moab Web Services.

> Oracle Java 7 Runtime Environment is the recommended Java environment, but Oracle Java 6 is also supported. All other versions of Java, including OpenJDK/IcedTea, GNU Compiler for Java, and so on cannot run Moab Web Services.

Do the following:

- Download the *Linux x64 RPM* version of Oracle Java SE 7 JRE. (Go to the Oracle Java 7 download page, copy the URL to the *Linux x64 RPM* version, then run the following command.)

```
[root]# wget <URL> -O jre-7-linux-x64.rpm
```

  To verify that the download was successful, run the following on the RPM before installation:

```
[root]# rpm -qip jre-7-linux-x64.rpm
```

- Run the following to install Java 7:

```
[root]# rpm -Uh jre-7-linux-x64.rpm
```

Install Tomcat

You must install Tomcat if you are installing Moab Web Services.

- **RHEL, CentOS, and Scientific Linux**:

```
[root]# yum install tomcat6
```

- **SLES**:

```
[root]# zypper ar --refresh -r
http://download.opensuse.org/evergreen/11.4/openSUSE:Evergreen:11.4.repo
[root]# zypper in tomcat6
[root]# zypper mr -d openSUSE_Evergreen_11.4
```

Opening ports

A few ports need to be available through your firewall so components of the suite can communicate with each other. Some features of some components might need additional ports configured. The individual component documentation indicates when additional ports are needed.

The ports required for basic suite functionality are:

- 7112: Default, configurable port needed for Moab Accounting Manager client-server communication
- 443: Needed for Moab Accounting Manager web GUI (https)
- 8080: Needed for Moab Web Services web portal (http)

To open ports in your firewall

- **Use `iptables` for Red Hat-based distributions**:

```
[root]# iptables-save > /tmp/iptables.mod
[root]# vi /tmp/iptables.mod

# Add the following lines immediately *before* the line matching
# "-A INPUT -j REJECT --reject-with icmp-host-prohibited"
-A INPUT -p tcp --dport 7112 -j ACCEPT
-A INPUT -p tcp --dport 443 -j ACCEPT
-A INPUT -p tcp --dport 8080 -j ACCEPT

[root]# iptables-restore < /tmp/iptables.mod
[root]# service iptables save
```

- **Use `SuSEfirewall2` for SuSE-based distributions**:

```
[root]# vi /etc/sysconfig/SuSEfirewall2

FW_SERVICES_EXT_TCP="443 7112 8080"

[root]# service SuSEfirewall2_setup restart
```

Install MongoDB

You must install MongoDB if you are installing Moab Web Services.

To install and enable MongoDB

1. Install MongoDB.

   - **RHEL and CentOS, and Scientific Linux**:

     Create a file called `/etc/yum.repos.d/10gen.repo` and add the following lines.

     ```
     [10gen]
     name=MongoDB Repository
     baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64
     gpgcheck=0
     enabled=1
     ```

     Install `mongo-10gen-server`.

     ```
     [root]# yum install mongo-10gen-server --exclude mongodb-org,mongodb-org-server
     ```

   - **SLES**:

     ```
     [root]# zypper ar
     http://download.opensuse.org/repositories/server:/database/SLE_11_SP3
     OpenSuseDatabase
     [root]# zypper install mongodb
     ```

2. Start MongoDB.

   - **RHEL and CentOS, and Scientific Linux**:

   ```
   [root]# chkconfig mongod on
   [root]# service mongod start
   ```

   - **SLES**:

   ```
   [root]# chkconfig mongodb on
   [root]# service mongodb start
   ```

   There may be a short delay (approximately 3 minutes) for Mongo to start the first time.

   If you see errors while running the `chkconfig` command, make sure that `/sbin` is in your `PATH` environment variable, then run `chkconfig` again.

   ```
   export PATH=/sbin:$PATH
   ```

3. Prepare the MongoDB database by doing the following:

   a. Add the required MongoDB users.

   The passwords used below (`secret1`, `secret2`, and `secret3`) are examples. Choose your own passwords for these users.

   ```
   [root]# mongo
   > use admin;
   > db.addUser("admin_user", "secret1");
   > db.auth ("admin_user", "secret1");

   > use moab;
   > db.addUser("moab_user", "secret2");
   > db.addUser("mws_user", "secret3", true);

   > use mws;
   > db.addUser("mws_user", "secret3");
   > exit
   ```

   Because the `admin_user` has read and write rights to the `admin` database, it also has read and write rights to all other databases. See Control Access to MongoDB Instances with Authentication for more information.

   b. Enable authentication in MongoDB.

   - **RHEL and CentOS, and Scientific Linux**:

   ```
   [root]# vi /etc/mongod.conf

   auth = true

   [root]# service mongod restart
   ```

- **SLES**:

```
[root]# vi /etc/mongodb.conf

auth = true

[root]# service mongodb restart
```

On SLES machines, auth = true is enabled by default.

Install PostgreSQL

To install PostreSQL

1. Install and initialize PostgreSQL.

   CentOS, RHEL, and Scientific Linux

```
[root]# yum install postgresql-server
[root]# service postgresql initdb
```

   SLES

```
[root]# zypper install postgresql-server
[root]# service postgresql start
```

2. Configure trusted connections.

   Edit or add a "host" line in the pg_hba.conf file for the interface from which the server(s) (for example, Moab Workload Manager and/or Moab Accounting Manager) will be connecting to the database and ensure that it specifies a secure password-based authentication method (for example, md5).

```
[root]# vi /var/lib/pgsql/data/pg_hba.conf

# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
```

3. Configure PostgreSQL to accept connections from your host.

```
[root]# vi /var/lib/pgsql/data/postgresql.conf

# Uncomment the listen addresses line in the configuration:

listen_addresses = 'localhost'          # what IP address(es) to listen on;
```

4. Start or restart the database.

```
[root]# chkconfig postgresql on
[root]# service postgresql restart
```

Related topics

- **Welcome** on page xxi

Installing TORQUE

These instructions describe how to install and start TORQUE.

Requirements

Supported Operating Systems

- CentOS 6.5 or later
- Red Hat 6.5 or later
- Scientific Linux 6.5 or later
- SUSE Linux Enterprise Server 11 SP3 or later

> CentOS 5.9, Red Hat 5.9 and Scientific Linux 5.9 are supported, largely to continue support for clusters where the compute nodes operating systems cannot be upgraded. We recommend that the TORQUE head node run on the supported operating systems listed above.

Software Requirements

- libxml2-devel package (package name may vary)
- openssl-devel package (package name may vary)
- Tcl/Tk version 8 or later if you plan to build the GUI portion of TORQUE or use a Tcl based scheduler
- If you use cpusets, libhwloc 1.1 or later is required (for TORQUE 4.0.0 and later)

If you build TORQUE from source (i.e. clone from github), the following additional software is required:

- gcc
- gcc-c++
- A posix compatible version of make
- libtool 1.5.22
- boost-devel 1.36.0

Prerequisites

- TORQUE requires certain ports to be open for essential communication:
    - For client communication to `pbs_server`, all privileged ports must be open (ports under 1024).
    - For `pbs_server` communication to `pbs_mom`, the default port is 15003.
    - For `pbs_mom` to `pbs_server`, the default port is 15001.

  For more information on how to configure the ports that TORQUE uses for communication, see **Configuring Ports** on page 2204.

  > **Important**: If you intend to use TORQUE 5.0.x with Moab, you must run Moab version 8.0.x or later. TORQUE 5.0.x will not work with versions earlier than Moab 8.0.x.

- Make sure your host (with the correct IP address) is in your `/etc/hosts` file.
- The `libxml2-devel`, `openssl-devel`, and `boost-devel` packages must be installed (These packages should already be installed from following the steps in the **Preparing for installation** on page 24).

  RHEL 6.5 and CentOS 6.5, and Scientific Linux 6.5:

```
[root]# yum install openssl-devel libtool-devel libxml2-devel boost-devel gcc
gcc-c++
```

  SLES

```
[root]# zypper install openssl-devel libtool-devel libxml2-devel boost-devel
gcc gcc-c++
```

  RHEL 5 and CentOS 5, and Scientific Linux 5:

```
[root]# yum install openssl-devel libtool-devel libxml2-devel gcc gcc-c++
```

  > **Important**: TORQUE requires Boost version 1.36.0 or greater. The boost-devel package provided with RHEL 5, CentOS 5, and Scientific Linux 5 is older than this requirement. A new option, --with-boost-path has been added to configure (see **Customizing the Install** on page 2207 in the *TORQUE Administrator Guide* for more information). This allows you to point TORQUE to a specific version of boost during make. One way to compileTORQUE without installing Boost is to simply download the Boost version you plan to use from: http://www.boost.org/users/history/. Next, untar Boost—you do not need to build it or install it. When you run TORQUE configure, use the --with-boost-path option pointed to the extracted Boost directory.

To install TORQUE

1. Switch the user to root.

```
[user]$ su -
```

2. Download the latest 5.0.1 build from the Adaptive Computing website. It can also be downloaded via command line (github method or the tarball distribution).

   - **Clone the source from github:**

     If you clone the source from github, the libtool package must be installed.

```
# RHEL 6 and Scientific Linux 6:
[root]# yum install git libtool

# SLES:
[root]# zypper install libtool
```

```
[root]# git clone https://github.com/adaptivecomputing/torque.git -b 5.0.1 5.0.1
[root]# cd 5.0.1
[root]# ./autogen.sh
```

     If you are using CentOS 5, use these instructions for installing `libtool`:

```
[root]# cd /tmp
[root]# wget http://ftpmirror.gnu.org/libtool/libtool-2.4.2.tar.gz
[root]# tar -xzvf libtool-2.4.2.tar.gz
[root]# cd libtool-2.4.2
[root]# ./configure --prefix=/usr
[root]# make
[root]# make install
[root]# cd /tmp
[root]# git clone https://github.com/adaptivecomputing/torque.git -b
5.0.1 5.0.1
[root]# cd 5.0.1
[root]# ./autogen.sh
```

   - **Get the tarball source distribution:**

```
[root]# wget http://www.adaptivecomputing.com/download/torque/torque-
5.0.1.tar.gz -O torque-5.0.1.tar.gz

[root]# tar -xzvf torque-5.0.1.tar.gz
[root]# cd torque-5.0.1/
```

3. Run each of the following commands in order.

```
[root]# ./configure
[root]# make
[root]# make install
```

   For information on what options are available to customize the ./configure command, see **Customizing the Install** on page 2207.

4. Configure the `trqauthd` daemon to start automatically at system boot.

```
* If RHEL distribution, do the following *
[root]# cp contrib/init.d/trqauthd /etc/init.d/
[root]# chkconfig --add trqauthd
[root]# echo /usr/local/lib > /etc/ld.so.conf.d/torque.conf
[root]# ldconfig
[root]# service trqauthd start

* If SLES distribution, do the following *
[root]# cp contrib/init.d/suse.trqauthd /etc/init.d/trqauthd
[root]# chkconfig --add trqauthd
[root]# echo /usr/local/lib > /etc/ld.so.conf.d/torque.conf
[root]# ldconfig
[root]# service trqauthd start
```

5. The make packages command can be used to create self-extracting packages that can be copied and executed on your nodes. For information on creating packages and deploying them, see **Compute Nodes** on page 2198.

   You will also want to scp the `init.d` scripts to the compute nodes and install them there.

6. Verify that the `/var/spool/torque/server_name` file exists and contains the correct name of the server.

```
[root]# echo <pbs_server's_hostname> > /var/spool/torque/server_name
```

7. By default, TORQUE installs all binary files to `/usr/local/bin` and `/usr/local/sbin`. Make sure the path environment variable includes these directories for both the installation user and the root user.

```
[root]# export PATH=/usr/local/bin/:/usr/local/sbin/:$PATH
```

8. Initialize `serverdb` by executing the `torque.setup` script.

```
[root]# ./torque.setup root
```

9. Add nodes to the `/var/spool/torque/server_priv/nodes` file. For information on syntax and options for specifying compute nodes, see **Specifying Compute Nodes** on page 2202.

10. Configure the MOMs if necessary (see **Configuring TORQUE on Compute Nodes** on page 2203 in the TORQUE Administrator Guide).

11. Configure pbs_server and pbs_mom to start automatically at system boot, and then start their daemons.

```
* If RHEL distribution, do the following *
[root]# cp contrib/init.d/pbs_server contrib/init.d/pbs_mom /etc/init.d
[root]# chkconfig --add pbs_server
[root]# chkconfig --add pbs_mom
[root]# service pbs_server restart
[root]# service pbs_mom start

* If SLES distribution, do the following *
[root]# cp contrib/init.d/suse.pbs_server /etc/init.d/pbs_server
[root]# cp contrib/init.d/suse.pbs_mom /etc/init.d/pbs_mom
[root]# chkconfig --add pbs_server
[root]# chkconfig --add pbs_mom
[root]# service pbs_server restart
[root]# service pbs_mom start
```

Related topics

- **Preparing for installation** on page 24
- **Installing Moab Workload Manager** on page 34
- **Component documentation** on page 95

Installing Moab Workload Manager

These instructions describe how to install and start Moab Workload Manager (Moab).

Dependencies and packages installation

Use the following commands to install the required Moab Workload Manager dependencies and packages (listed in the **Installing Moab Workload Manager** section above).

RHEL , CentOS, and Scientific Linux:

```
[root]# yum update
[root]# yum install make libcurl perl-CPAN libxml2-devel
```

SLES:

```
[root]# zypper update
[root]# zypper install make curl libxml2-devel
```

Optional: To build a custom RPM

1. Install `rpm-build`.

```
[root]# yum install rpm-build
```

2. Download the latest Moab build (`moab-<version>-<OS>.tar.gz`) from the Adaptive Computing website.

> The variable marked *<version>* is the desired version of the suite; for example, 8.0-2014061017-8f96ac8d would be Moab 8.0 revision 2014061017 at changeset 8f96ac8d. The variable marked *<OS>* indicates which OS the build was designed for.

3. Untar the downloaded package.

4. Change directories into the untarred directory.
5. Edit the ./moab.spec file for RPM customization.
6. Run ./rpm-build.
7. Locate the custom RPM in rpm/RPMS/x86_64.

To install Moab Workload Manager

1. Download the latest Moab build (`moab-<version>-<OS>.tar.gz`) from the Adaptive Computing website.

   > The variable marked *<version>* is the desired version of the suite; for example, 8.0-2014061017-8f96ac8d would be Moab 8.0 revision 2014061017 at changeset 8f96ac8d. The variable marked *<OS>* indicates which OS the build was designed for.

2. As the root user, run each of the following commands in order.

```
[root]# tar xzvf moab-<version>-<OS>.tar.gz
[root]# cd moab-<version>-<OS>
```

3. Configure Moab. For a complete list of `./configure` options, use `./configure --help` or refer to **Moab Workload Manager configuration options** on page 49 for a list of commonly used options.

   It is strongly recommended that you configure Moab with the `--with-init` and `--with-profile` options. The `--with-profile` option makes it easier to execute Moab commands (see step 8). The `--with-init` option allows Moab to automatically start at OS startup (see step 11).

```
[root]# ./configure <options>
```

4. (Only if you are using green computing, or if you are using a resource manager other than TORQUE) Run the make perldeps command to install the necessary perl modules using CPAN. When first running CPAN, you will be asked for configuration information. It is recommended that you choose an automatic configuration. You will be prompted to provide input during module installation; running the make perldeps command with a script is not recommended.

```
[root]# make perldeps
```

5. Install Moab.

```
[root]# make install
```

6. (ONLY if installing on non-RHEL distributions)

   Copy the appropriate `init.d` file, set the permissions on it, and configure Moab to start automatically at system boot.

```
* If SLES distribution, do the following *
[root]# cp OS/SLES/etc/init.d/moab /etc/init.d/moab

[root]# chmod 755 /etc/init.d/moab
[root]# chkconfig --add moab
```

7. Modify the Moab configuration file.

```
[root]# vi /opt/moab/etc/moab.cfg
```

Do the following:

a. Verify that **SUBMITCMD** is set up for your TORQUE resource manager and that it points to a valid `qsub` executable. For example:

```
RMCFG[torque] SUBMITCMD=/usr/local/bin/qsub
```

If you use a SLURM resource manager, see Moab-SLURM Integration Guide for configuration information. If you use a NATIVE resource manager, see Managing Resources Directly with the Native Interface for configuration information.

b. *ONLY* if you are using Moab Web Services, add *tomcat* to the list of administrator **USERS**. For example:

```
ADMINCFG[1] USERS=root,tomcat
```

8. If you configured with the ./configure –with-profile option, source the following file to add the Moab executable directories to your current shell *$PATH* environment.

```
[root]# . /etc/profile.d/moab.sh
```

9. Copy your license file into the same directory as `moab.cfg` (`/opt/moab/etc/` by default). For example:

```
[root]# cp moab.lic $MOABHOMEDIR/etc/moab.lic
```

To verify the current status of your license, use `moab --about`.

Moab checks the status of the license every day just after midnight. At 60 and 45 days before, and daily from 30 days before license expiration to and including the license expiration date, Moab sends an e-mail to all level 1 administrators informing them of the pending Moab license expiration. A log record is also made of the upcoming expiration event. For the notifications to occur correctly, you must enable administrator email notification (see "**Notifying Administrators of Failures** on page 684" in the Moab Workload Manager Administrator Guide) and `moab.cfg` must contain email addresses for level 1 administrators. For example:

```
ADMINCFG[1] USERS=u1,u2,u3[,...]

USERCFG[u1] EMAILADDRESS=u1@company.com
USERCFG[u2] EMAILADDRESS=u2@company.com
USERCFG[u3] EMAILADDRESS=u3@company.com

MAILPROGRAM DEFAULT
```

Moab will not run without a license. For information about obtaining a trial license, please contact Adaptive Computing.

10. Start Moab.

```
[root]# chkconfig moab on
[root]# service moab start
```

11. Submit a sleep job as a non-root user and verify the job is running.

```
[root]# su - user
[user]$ echo sleep 150 | msub
[user]$ showq
[user]$ exit
```

12. Connecting Moab to MongoDB

If you will be installing Moab Web Services, connect Moab to MongoDB using the following instructions:

> The USEDATABASE parameter is unrelated to the MongoDB configuration.

a. Set the **MONGOSERVER** parameter in /opt/moab/etc/moab.cfg to the MongoDB server hostname. Use localhost as the hostname if Moab and MongoDB are hosted on the same server.

```
MONGOSERVER <host>[:<port>]
```

If your **MONGOSERVER** host is set to anything other than localhost, edit the /etc/mongod.conf file on the MongoDB server host and either comment out any bind_ip parameter or set it to the correct IP address:

```
# Listen to local interface only. Comment out to listen on all interfaces.
#bind_ip=127.0.0.1
```

b. In the /opt/moab/etc/moab-private.cfg file, set the **MONGOUSER** and **MONGOPASSWORD** parameters to the MongoDB moab_user credentials you set (for details, see **Install MongoDB** on page 27).

```
MONGOUSER     moab_user
MONGOPASSWORD secret2
```

c. Verify that Moab is able to connect to MongoDB.

```
[root]# service moab restart
[root]# mdiag -S
...
Mongo connection (localhost) is up (credentials are set)
...
```

13. Securing communication using secret keys

a. (required) Moab and MWS use Message Authentication Codes (MAC) to ensure messages have not been altered or corrupted in transit. Generate a key and store the result in /opt/moab/etc/.moab.key:

```
[root]# service moab stop
[root]# dd if=/dev/urandom count=18 bs=1 2>/dev/null | base64 >
/opt/moab/etc/.moab.key
[root]# chown root:root /opt/moab/etc/.moab.key
[root]# chmod 400 /opt/moab/etc/.moab.key
[root]# service moab start
```

The key you specify in the `.moab.key` file is the same key you must also specify in the `moab.secretKey` property when installing and configuring MWS (see **Installing Moab Web Services** on page 38).

b. (optional) Moab supports message queue security using AES. This feature requires a Base64-encoded 16-byte (128-bit) shared secret. Generate a key and append the result to `/opt/moab/etc/moab-private.cfg`:

```
[root]# service moab stop
[root]# echo "MESSAGEQUEUESECRETKEY $(dd if=/dev/urandom count=16 bs=1
2>/dev/null | base64)" >> /opt/moab/etc/moab-private.cfg
[root]# service moab start
```

The key you specify in the moab-private.cfg file is the same key you must also specify in the moab.messageQueue.secretKey property when installing and configuring MWS (see Installing Moab Web Services).

> If MWS is configured to encrypt the message queue and Moab is not (or vice versa), then MWS will ignore the messsages from Moab. Furthermore, all attempts to access the MWS service resource will fail.

c. (optional) Verify that encryption is on for the ZeroMQ connection.

```
[root]# mdiag -S|grep 'ZeroMQ MWS'
  ZeroMQ MWS connection is bound on port 5570 (encryption is on)
```

Related topics

- **Preparing for installation** on page 24
- **Installing TORQUE** on page 2193
- **Component documentation** on page 95

Installing Moab Web Services

These instructions describe how to install Moab Web Services (MWS).

To install Moab Web Services

> You must deploy Moab Web Services on the same server as Moab Workload Manager.

1. Verify Moab is installed and configured as desired (for details, see Installing Moab Workload Manager).
2. Start Moab.

```
[root]# service moab start
```

3. Create the MWS home directory and subdirectories (for more information, see the "**Configuration** on page 1750" section of the Moab Web Services Reference Guide).

The default location for the MWS home directory is /opt/mws. These instructions assume the default location.

Here is a sample script for this setup:

```
[root]# mkdir -p \
    /opt/mws/etc/mws.d \
    /opt/mws/hooks \
    /opt/mws/log \
    /opt/mws/plugins \
    /opt/mws/spool/hooks \
    /opt/mws/utils
[root]# chown -R tomcat:tomcat /opt/mws # Depending on your OS, the Tomcat username
might be tomcat6.
[root]# chmod -R 555 /opt/mws
[root]# chmod u+w \
    /opt/mws/log \
    /opt/mws/plugins \
    /opt/mws/spool \
    /opt/mws/spool/hooks \
    /opt/mws/utils
```

4. Download the latest MWS build (mws-<version>.tar.gz) from the Adaptive Computing website.
5. Extract the contents of the MWS download tarball into a temporary directory. For example:

```
[root]# mkdir /tmp/mws-install
[root]# cd /tmp/mws-install
[root]# tar xvzf $HOME/Downloads/mws-8.0.1.tar.gz
```

6. Copy the extracted utility files to the utility directory created above and give the tomcat user ownership of the directory.

```
[root]# cd /tmp/mws-install/mws-8.0.1/utils
[root]# cp * /opt/mws/utils
[root]# chown tomcat:tomcat /opt/mws/utils/*
```

7. Set up the MWS configuration files. In the extracted directory are several configuration files.
    a. Copy mws-config.groovy to /opt/mws/etc.

    ```
    [root]# cd /tmp/mws-install/mws-8.0
    [root]# cp mws-config.groovy /opt/mws/etc
    ```

    b. Copy the appropriate suite-specific file to /opt/mws/etc/mws.d. Pick from the files matching filename mws-config-*.groovy in /tmp/mws-install/mws-8.0.

    ```
    [root]# cp mws-config-<your suite choice>.groovy /opt/mws/etc/mws.d
    ```

    c. Give the Tomcat user read access to /opt/mws/etc/mws-config.groovy and /opt/mws/etc/mws.d/mws-config-*.groovy.

```
[root]# chown tomcat:tomcat /opt/mws/etc/mws-config.groovy
/opt/mws/etc/mws.d/mws-config-hpc.groovy
[root]# chmod 400 /opt/mws/etc/mws-config.groovy /opt/mws/etc/mws.d/mws-config-
hpc.groovy
```

d. In the `/opt/mws/etc/mws-config.groovy` file, change these settings:

- **moab.secretKey**: Must match the Moab secret key you generated earlier (contained in `/opt/moab/etc/.moab.key`).
- **auth.defaultUser.username**: Any value you like, or leave as is.
- **auth.defaultUser.password**: Any value you like, but choose a strong password.
- **moab.messageQueue.secretKey**: Add this property to configure the message queue security key in MWS.

> The key you specify must be encoded in Base64, and must match *exactly* the key specified in the **MESSAGEQUEUESECRETKEY** when installing Moab Workload Manager (see **Installing Moab Workload Manager** on page 34).

> Important: If MWS is configured to encrypt the message queue and Moab is not (or vice versa) then the messages from Moab will be ignored. Furthermore, all attempts to access the MWS service resource will fail.

```
[root]# vi /opt/mws/etc/mws-config.groovy

// Replace <ENTER-KEY-HERE> with the contents of /opt/moab/etc/.moab.key.

moab.secretKey = "<ENTER-KEY-HERE>"
moab.server = "localhost"
moab.port = 42559

// Replace <ENTER-KEY-HERE> with the value of MESSAGEQUEUESECRETKEY in
/opt/moab/etc/moab-private.cfg.

moab.messageQueue.secretKey = "<ENTER-KEY-HERE>"

// Change these to be whatever you like.

auth.defaultUser.username = "moab-admin"
auth.defaultUser.password = "changeme!"
```

> If you do not change **auth.defaultUser.password**, your MWS will not be secure (because anyone reading these instructions would be able to log into your MWS). Here are some tips for choosing a good password.

e. Do *one* of the following:

You can configure only one authentication method in mws-config.groovy—LDAP or PAM, but not both. If you have configured both LDAP and PAM, MWS defaults to using LDAP.

If you need multiple authentication methods, you must add them to your local PAM configuration. See your distribution documentation for details.

- If you are configuring an MWS connection to your LDAP server, add the following parameters to /opt/mws/etc/mws-config.groovy:

```
ldap.server  = "192.168.0.5"
ldap.port = 389
ldap.baseDNs = ["dc=acme,dc=com"]
ldap.bindUser = "cn=Manager,dc=acme,dc=com"
ldap.password = "*****"
ldap.directory.type = "OpenLDAP Using InetOrgPerson Schema"
```

This is just an example LDAP connection. Be sure to use the appropriate domain controllers (dc) and common names (cn) for your environment.

If you followed the Adaptive Computing tutorial, **Setting up OpenLDAP on CentOS 6** on page 72, your **ldap.directory.type** should be set to "OpenLDAP Using InetOrgPerson Schema." However, the use of other schemas is supported. For more information see **LDAP Configuration using mws-config.groovy** on page 1380.

To see how to configure a secure connection to the LDAP server, see **Securing the LDAP connection** on page 1394.

- If you are configuring MWS to use PAM, add the the **pam.configuration.service** parameter to the mws-config.groovy file. For example:

```
pam.configuration.service = "login"
```

This is just an example PAM configuration file name. Make sure you specify the name of the configuration file you want MWS to use.

For more information about PAM configuration with MWS, see **PAM (pluggable authentication module) configuration using mws-config.groovy** on page 1383.

There is a security risk when authenticating local users through your PAM configuration. This behavior is highly discouraged and not supported by Adaptive Computing.

f. Add the **grails.mongo.username** and **grails.mongo.password** parameters to the mws-config.groovy file. Use the MWS credentials you added to MongoDB in the **Preparing for installation** on page 24 section.

```
...
grails.mongo.username = "mws_user"
grails.mongo.password = "secret3"
```

8. Add the following lines to the end of /etc/tomcat6/tomcat6.conf:

```
CATALINA_OPTS="-DMWS_HOME=/opt/mws -Xms256m -Xmx3g -XX:MaxPermSize=384m -
Dfile.encoding=UTF8"
JAVA_HOME="/usr/java/latest"
```

> Some Linux distributions use /etc/default/tomcat6 or /etc/sysconfig/tomcat6 instead of /etc/tomcat6/tomcat6.conf.

9. Deploy the mws.war file and start Tomcat.

```
[root]# chkconfig tomcat6 on
[root]# service tomcat6 stop
[root]# cp /tmp/mws-install/mws-8.0.1/mws.war /usr/share/tomcat6/webapps
[root]# service tomcat6 start
```

10. Navigate to http://localhost:8080/mws/ in a web browser to verify that MWS is running (you will see some sample queries and a few other actions).

11. Log in to MWS to verify that your credentials are working. (Your login credentials are the auth.defaultUser.username and auth.defaultUser.password values you set in the /opt/mws/etc/mws-config.groovy file.)

> If you encounter problems, or if the application does not seem to be running, see the steps in **Moab Web Services issues** on page 91.

Additional configuration

Configuring SSL in Tomcat

To configure SSL in Tomcat, please refer to the Apache Tomcat documentation.

Related topics

Setting up OpenLDAP on CentOS 6

These instructions are intended to help first-time LDAP administrators get up and running. The following procedures contain instructions for getting started using OpenLDAP on a CentOS 6 system. For more complete information on how to set up OpenLDAP see the OpenLDAP documentation.

- **Installing and configuring OpenLDAP on Centos 6** on page 43
- **Adding an organizational unit (OU)** on page 47
- **Adding a user** on page 47
- **Adding a group** on page 48
- **Adding a user to a group** on page 48

Adaptive Computing is not responsible for creating, maintaining, or supporting customer LDAP or Active Directory configurations.

Installing and configuring OpenLDAP on Centos 6

First, you will need to install OpenLDAP. These instructions explain how you can do this on a CentOS 6 system.

To install and configure OpenLDAP on Centos 6

1. Run the following command:

```
[root]# yum -y install openldap openldap-clients openldap-servers
```

2. Generate a password hash to be used as the admin password. This password hash will be used when you create the root user for your LDAP installation. For example:

```
[root]# slappasswd
New password : p@ssw0rd
Re-enter new password : p@ssw0rd
{SSHA}5lPFVw19zeh7LT53hQH69znzj8TuBrLv
```

3. Add the root user and the root user's password hash to the OpenLDAP configuration in the `olcDatabase={2}bdb.ldif` file. The root user will have permissions to add other users, groups, organizational units, etc. Do the following:

   a. Run this command:

```
[root]# cd /etc/openldap/slapd.d/cn\=config
[root]# vi olcDatabase\=\{2\}bdb.ldif
```

b. If the **olcRootPW** attribute does not already exist, create it. Then set the value to be the hash you created from slappasswd. For example:

```
olcRootPW: {SSHA}5lPFVw19zeh7LT53hQH69znzj8TuBrLv
...
```

4. While editing this file, change the distinguished name (DN) of the **olcSuffix** to something appropriate. The suffix typically corresponds to your DNS domain name, and it will be appended to the DN of every other LDAP entry in your LDAP tree.

   For example, let's say your company is called Acme Corporation, and that your domain name is "acme.com." You might make the following changes to the olcDatabase={2}bdb.ldif file:

```
olcSuffix: dc=acme,dc=com
...
olcRootDN: cn=Manager,dc=acme,dc=com
...
olcRootPW: {SSHA}5lPFVw19zeh7LT53hQH69znzj8TuBrLv
...
```

Throughout the following examples in this topic, you will see dc=acme,dc=com. "acme" is only used as an example to illustrate what you would use as your own domain controller if your domain name was "acme.com." You should replace any references to "acme" with your own organization's domain name.

Do not set the cn of your root user to "root" (cn=root,dc=acme,dc=com), or OpenLDAP will have problems.

5. Modify the DN of the root user in the olcDatabase={1}monitor.ldif file to match the **olcRootDN** line in the olcDatabase={2}bdb.ldif file. Do the following:

   a. Run this command to edit the olcDatabase={2}bdb.ldif file:

```
[root]# vi olcDatabase\=\{1\}monitor.ldif
```

   b. Modify the **olcAccess** line so that the **dn.base** matches the **olcRootDN** from the olcDatabase={2}bdb.ldif file. (In this example, **dn.base** should be "cn=Manager,dc=acme,dc=com".)

```
olcAccess: {0}to * by
dn.base="gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth" read by
dn.base="cn=Manager,dc=acme,dc=com" read by * none
```

   c. Now the root user for your LDAP is cn=Manager,dc=acme,dc=com. The root user's password is the password that you entered using slappasswd (see step 2), which, in this example, is **p@ssw0rd**

6. Hide the password hashes from users who should not have permission to view them.

A full discussion on configuring access control in OpenLDAP is beyond the scope of this tutorial. For help, see the OpenLDAP Access Control documentation.

a. Run this command to edit the `oclDatabase\=\{2\}bdb.ldif` file:

```
[root]# vi olcDatabase\=\{2\}bdb.ldif
```

b. Add the following two lines to the end of the file to restrict users from viewing other users' password hashes.

```
olcAccess: {0}to attrs=userPassword by self write by
dn.base="cn=Manager,dc=acme,dc=com" write by anonymous auth by * none
olcAccess: {1}to * by dn.base="cn=Manager,dc=acme,dc=com" write by self write by
* read
```

These lines allow a user to read and write his or her own password. It also allows a manager to read and write anyone's password. Anyone, including anonymous users, is allowed to view non-password attributes of other users.

7. Make sure that OpenLDAP is configured to start when the machine starts up, and start the OpenLDAP service.

```
[root]# chkconfig slapd on
[root]# service slapd start
```

8. Now, you must manually create the "dc=acme,dc=com" LDAP entry in your LDAP tree.

An LDAP directory is analogous to a tree. Nodes in this tree are called LDAP "entries" and may represent users, groups, organizational units, domain controllers, or other objects. The attributes in each entry are determined by the LDAP schema. In this tutorial we will build entries based on the InetOrgPerson schema (which ships with OpenLDAP by default).

In order to build our LDAP tree we must first create the root entry. Root entries are usually a special type of entry called a domain controller (DC). Because we are assuming that the organization is called Acme Corporation, and that the domain is "acme.com," we will create a domain controller LDAP entry called `dc=acme,dc=com`. Again, you will need to replace "acme" with your organization's domain name. Also note that `dc=acme,dc=com` is what is called an LDAP distinguished name (DN). An LDAP distinguished name uniquely identifies an LDAP entry.

Do the following:

a. Create a file called `acme.ldif`. (You can delete this file once its content has been added to LDAP, so in this example, we will create it in the `/tmp` folder.)

```
[root]# cd /tmp
[root]# vi acme.ldif
```

b. Add the following lines in `acme.ldif`:

```
dn: dc=acme,dc=com
objectClass: dcObject
objectClass: organization
dc: acme
o : acme
```

c. Now add the contents of this file to LDAP. Run this command:

```
[root]# ldapadd -f acme.ldif -D cn=Manager,dc=acme,dc=com -w p@ssw0rd
```

d. Verify that your entry was added correctly.

```
[root]# ldapsearch -x -LLL -b dc=acme,dc=com
dn: dc=acme,dc=com
objectClass: dcObject
objectClass: organization
dc: acme
o: acme
```

9. Run the following:

```
[root]# sudo iptables -L
[root]# sudo service iptables save
```

10. By default, the CentOS 6 firewall will block external requests to OpenLDAP. In order to allow MWS to access LDAP, you will have to configure your firewall to allow connections on port 389. (Port 389 is the default LDAP port.)

    Configuring your firewall is beyond the scope of this tutorial; however, it may be helpful to know that the default firewall on CentOS is a service called `iptables`. (For more information, see the documentation on iptables.) In the most basic case, you may be able to add a rule to your firewall that accepts connections to port 389 by doing the following:

    a. Edit your `iptables` file:

```
[root]# vi /etc/sysconfig/iptables
```

    b. Add the following line *after* all the **ACCEPT** lines but *before* any of the **REJECT** lines in your `iptables` file:

```
# ... lines with ACCEPT should be above
-A INPUT -p tcp --dport 389 -j ACCEPT
# .. lines with REJECT should be below
```

    For example, here is a sample `iptables` file with this line added:

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
-A INPUT -p tcp --dport 389 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited

COMMIT
```

    c. Now reload `iptables`.

```
[root]# service iptables reload
```

Although providing instructions is beyond the scope of this tutorial, it is also highly recommended that you set up OpenLDAP to use SSL or TLS security to prevent passwords and other sensitive data from being sent in plain text. For information on how to do this, see the OpenLDAP TLS documentation.

Now that you have installed and set up Open LDAP, you are ready to add organizational units (see **Adding an organizational unit (OU)** on page 47).

Adding an organizational unit (OU)

These instructions will describe how to populate the LDAP tree with organizational units (OUs), groups, and users, all of which are different types of LDAP entries. The examples that follow also presume an InetOrgPerson schema, because the InetOrgPerson schema is delivered with OpenLDAP by default.

To add an organizational unit (OU) entry to the LDAP tree

In this example, we are going to add an OU called "Users."

1. Create a temporary file called `users.ldif`. (You can delete this file once its content has been added to LDAP, so in this example, we will create it in the `/tmp` folder.)

```
[root]# cd /tmp
[root]# vi users.ldif
```

2. Add these lines to `users.ldif`:

```
dn: ou=Users,dc=acme,dc=com
objectClass: organizationalUnit
ou: Users
```

3. Add the contents of `users.ldif` file to LDAP.

```
[root]# ldapadd -f users.ldif -D cn=Manager,dc=acme,dc=com -w p@ssw0rd
```

Adding a user

To add a user to LDAP

In this example, we will add a user named "Bob Jones" to LDAP inside the "Users" OU.

1. Create a temporary file called `bob.ldif`. (You can delete this file once its content has been added to LDAP, so in this example, we will create it in the `/tmp` folder.)

```
[root]# cd /tmp
[root]# vi bob.ldif
```

2. Add these lines to `bob.ldif`:

```
dn: cn=Bob Jones,ou=Users,dc=acme,dc=com
cn: Bob Jones
sn: Jones
objectClass: inetOrgPerson
userPassword: p@ssw0rd
uid: bjones
```

3. Add the contents of `bob.ldif` file to LDAP.

```
[root]# ldapadd -f bob.ldif -D cn=Manager,dc=acme,dc=com -w p@ssw0rd
```

Adding a group

To add a group to LDAP

In this example, we will add a group called "Engineering" to LDAP inside the "Users" OU.

1. Create a temporary file called `engineering.ldif`. (You can delete this file once its content has been added to LDAP, so in this example, we will create it in the `/tmp` folder.)

```
[root]# cd /tmp
[root]# vi engineering.ldif
```

2. Add these lines to `engineering.ldif`:

```
dn: cn=Engineering,ou=Users,dc=acme,dc=com
cn: Engineering
objectClass: groupOfNames
member: cn=Bob Jones,ou=Users,dc=acme,dc=com
```

3. Add the contents of `engineering.ldif` file to LDAP.

```
[root]# ldapadd -f engineering.ldif -D cn=Manager,dc=acme,dc=com -w p@ssw0rd
```

Adding a user to a group

To add a user to an LDAP group

In this example, we will add an LDAP member named "Al Smith" to the "Engineering" LDAP group. This example assumes that user, Al Smith, has already been added to LDAP.

> Before you add a user to an LDAP group, the user must first be added to LDAP. For more information, see **Adding a user** on page 47.

1. Create a temporary file called `addUserToGroup.ldif`. (You can delete this file once its content has been added to LDAP, so in this example, we will create it in the `/tmp` folder.)

```
[root]# cd /tmp
[root]# vi addUserToGroup.ldif
```

2. Add these lines to `addUserToGroup.ldif`:

```
dn: cn=Engineering,ou=Users,dc=acme,dc=com
changetype: modify
add: member
member: cn=Al Smith,ou=Users,dc=acme,dc=com
```

3. Now add the contents of `addUserToGroup.ldif` file to LDAP.

```
[root]# ldapadd -f addUserToGroup.ldif -D cn=Manager,dc=acme,dc=com -w p@ssw0rd
```

Related topics

Moab Workload Manager configuration options

The following is a list of commonly used configure options. For a complete list, use ./configure --help when configuring Moab.

Option	Description	Example
--with-flexlm	Causes Moab to install the `license.mon.flexLM.pl` script in the `/opt/moab/tools` directory. For more information about this script, see the **Interfacing to FLEXlm** on page 653 section in the Moab Administrator Guide.	`[root]# ./configure --with-flexlm`
--with-homedir	Specifies the location of the Moab configuration directory and the MOABHOMEDIR environment variable. The default location is `/opt/moab`. MOABHOMEDIR is automatically set on some distributions during installation, when the --with-profile option is enabled.	`[root]# ./configure --with-homedir=/var/moab` *The Moab HPC Suite home directory will be `/var/moab` instead of the default `/opt/moab`.*

Option	Description	Example
--with-init	Enables the installation of a distribution-specific `/etc/init.d/moab` service startup script. This option is required if you want to install this script onto a new system. If you do not set this option, you must manually set up the Moab daemon service. The startup script is located at `OS/EL/etc/init.d/moab`. The TORQUE and Moab HPC Suite initialization scripts are provided in the `contrib/init.d` directory as a courtesy and may be modified at your discretion to work on your system.	`[root]# ./configure --with-init`
--prefix	Specifies the location of the binaries and libraries of the Moab install. The default location is `/opt/moab`.	`[root]# ./configure --prefix=/usr/local`
--with-profile	Enables the installation of distribution-specific `/etc/profile.d/moab.[c]sh` setup script for bash and cshell. The MOABHOMEDIR, PERL5LIB, PATH and MANPATH environment variables are setup to specify where the new moab configuration, scripts, binaries and man pages reside. If you do not set this option, these scripts are not installed, and you must manually perform this set up. The environment setup scripts are located at `OS/EL/etc/profile.d/moab.[c]sh`.	`[root]# ./configure --with-profile`

Option	Description	Example
--with-am	Specifies that you want to configure Moab with Moab Accounting Manager. There is a similar `--with-torque` option that configures Moab with TORQUE, but you do not need to specify this option if you install the "torque" tarball version.	`[root]# ./configure --with-am`

Trusting servers in Java

Prerequisites

Some of these instructions refer to `JAVA_HOME`, which must point to the same directory that Tomcat uses. To set `JAVA_HOME`, do this:

```
[root]# source /etc/tomcat6/tomcat6.conf
```

Your system administrator might have defined Tomcat's `JAVA_HOME` in a different file.

Retrieve the server's X.509 public certificate

To retrieve the server's certificate, use the following command:

```
[root]# $JAVA_HOME/bin/keytool -printcert -rfc -sslserver <servername>:<port> >
/tmp/public.cert.pem
```

Replace *<servername>* with the server's host name and *<port>* with the secure port number. The default port for https is 443. The default port for ldaps is 636. If successful, `/tmp/public.cert.pem` contains the server's public certificate. Otherwise, `/tmp/public.cert.pem` contains an error message. This message is typical: `keytool error: java.lang.Exception: No certificate from the SSL server`. This message suggests that the server name or port is incorrect. Consult your IT department to determine the correct server name and port.

Add the server's certificate to Java's keystore

Java stores trusted certificates in a database known as the keystore. Because each new version of Java has its own keystore, you need to add the server certificate to the Java keystore (using the steps below) every time you install a new version of Java.

Java's keystore is located at `$JAVA_HOME/lib/security/cacerts`. If Tomcat's `JAVA_HOME` points to a JDK, then the keystore is located at `$JAVA_HOME/jre/lib/security/cacerts`. To add the server certificate to the keystore, run the following command:

```
[root]# $JAVA_HOME/bin/keytool -import -trustcacerts -file /tmp/public.cert.pem -alias
<servername> -keystore $JAVA_HOME/lib/security/cacerts
```

You will be prompted for the keystore password, which is "changeit" by default.

> Your system administrator might have changed this password.

After you've entered the keystore password, you'll see the description of the server's certificate. At the end of the description it prompts you to trust the certificate.

```
Trust this certificate? [no]:
```

Type `yes` and press **Enter** to add the certificate to the keystore.

Upgrading

Preparing for upgrade

The upgrade process of the Moab HPC Suite includes upgrading the database and separate components in the suite. This guide contains detailed instructions for upgrading each component.

> It is highly recommended that you *first* perform upgrades in a *test environment*. Installation and upgrade procedures are tested prior to release; however, due to customizable variations that may be utilized by your configuration, it is not recommended to drop new versions of software directly into production environments. This is especially true when the workload has vital bearing. Contact Adaptive Computing Professional Services for more information.

> Because many system-level files and directories are accessed during the upgrade, the upgrade instructions in this guide should be executed with root privileges.
>
> You will see that the instructions execute commands as the root user. Please note that the same commands will work for a non-root user with the `sudo` command.

Upgrade the Moab HPC Suite in the following order:

Mongo database. See Upgrading MongoDB

TORQUE. See Upgrading TORQUE

Moab Workload Manager. See Upgrading Moab Workload Manager

Moab Web Services. See Upgrading Moab Web Services

Related topics

- Requirements

Upgrading MongoDB

Adaptive Computing strongly recommends upgrading MongoDB to version 2.4.x. Support for environments using MongoDB 2.0 is now deprecated and will be removed in future releases. Please refer to docs.mongodb.org for instructions on how to upgrade MongoDB. Note that you must pay close attention to the information regarding instances with `auth` enabled (as this is the recommended setup for Moab HPC Suite).

```
[root]# service mongod stop
[root]# yum remove mongo20-10gen-server mongo20-10gen
[root}# yum install mongo-10gen-server --exclude mongodb-org,mongodb-org-server
[root]# service mongod start
```

> Note that the settings in the `/etc/mongod.conf` file were saved in `/etc/mongod.conf.rpmsave` while removing MongoDB 2.0. You may need to be restore any custom settings after MongoDB 2.4.x is installed in the new `/etc/mongod.conf` file (for example, "auth = true").

After upgrading to MongoDB 2.4.x, you should verify that the MongoDB credentials were preserved:

```
[root]# mongo -u mws_user mws -p
MongoDB shell version: 2.4.8
connecting to: mws
> show collections
event
mongeez
permission
...
```

> Running yum upgrade will replace MongoDB 2.4.x with a more recent, and incompatible version. Consider using yum version lock to maintain MongoDB 2.4.x.

Upgrading TORQUE

TORQUE 5.0.1 binaries are not backward compatible with previous versions of TORQUE. When you upgrade to TORQUE 5.0.1, all MOM and server daemons must be upgraded at the same time.

The job format is compatible between 5.0.1 and previous versions of TORQUE. Any queued jobs will upgrade to the new version with the exception of job arrays in TORQUE 2.4 and earlier. It is not recommended to upgrade TORQUE while jobs are in a running state.

Job arrays

Job arrays from TORQUE version 2.5 and 3.0 are compatible with TORQUE 5.0.1. Job arrays were introduced in TORQUE version 2.4 but modified in 2.5. If upgrading from TORQUE 2.4, you need to make sure all job arrays are complete before upgrading.

serverdb

The pbs_server configuration is saved in the file *$TORQUE_HOME*/server_priv/serverdb. When running TORQUE 4.1.0 or later for the first time, this file converts from a binary file to an XML-like format. This format can be used by TORQUE versions 2.5 and 3.0, but not earlier versions. Back up the *$TORQUE_HOME*/server_priv/serverdb file before moving to TORQUE 4.1.0 or later.

Jobs

Before upgrading the system, all running jobs must complete. To prevent queued jobs from starting, nodes can be set to offline or all queues can be disabled. Once all running jobs are complete, the upgrade can be made. Remember to allow any job arrays in version 2.4 to complete before upgrading. Queued array jobs will be lost.

Cray

For upgrading TORQUE to 5.0.1 on a Cray system, refer to the **Installation Notes for Moab and TORQUE for Cray** on page 1214 in Appendix G of the Moab Workload Manager Administrator Guide.

To upgrade TORQUE

1. Shut down TORQUE.

```
[root]# qterm
[root]# momctl -s

* If running TORQUE 4.6.0 or later *
[root]# trqauthd -d

*If running a version of TORQUE earlier than 4.6.0 *
[root]# ps -efw | grep trqauthd
root       1487      1   0 Dec18 ?        00:00:00 /usr/sbin/trqauthd
adaptive   4830   4374   0 15:07 pts/0    00:00:00 grep trqauthd

[root]# kill -9 1487
```

2. Back up your server_priv directory.

```
[root]# tar -cvf backup.tar.gz $TORQUE_HOME/server_priv
```

3. If not already installed, install the Boost C++ headers.

```
[root]# yum install boost-devel
```

For SLES, use `zypper install <package names>` instead of `yum install <package names>`.

4. Install the latest TORQUE tarball.

```
[root]# cd /tmp
[root]# tar xzvf torque-5.0.1-<build number>.tar.gz
[root]# cd torque-5.0.1-<build number>
[root]# ./configure
[root]# make
[root]# make install
```

5. If not already done, configure pbs_server and pbs_mom to start automatically at system boot.

```
* If Debian distribution, do the following *
[root]# cp contrib/init.d/debian.pbs_server /etc/init.d/pbs_server
[root]# cp contrib/init.d/debian.pbs_mom /etc/init.d/pbs_mom
[root]# chkconfig --add pbs_server
[root]# chkconfig --add pbs_mom

* If SLES distribution, do the following *
[root]# cp contrib/init.d/suse.pbs_server /etc/init.d/pbs_server
[root]# cp contrib/init.d/suse.pbs_mom /etc/init.d/pbs_mom
[root]# chkconfig --add pbs_server
[root]# chkconfig --add pbs_mom

* If RHEL distribution, do the following *
[root]# cp contrib/init.d/pbs_server contrib/init.d/pbs_mom /etc/init.d
[root]# chkconfig --add pbs_server
[root]# chkconfig --add pbs_mom
```

6. Start the services.

```
[root]# service trqauthd start
[root]# service pbs_mom start
[root]# service pbs_server start
```

7. Check the status of jobs in the queue and perform other checks for errors.

```
[root]# qstat
[root]# grep -i error /var/spool/torque/server_logs/*
[root]# grep -i error /var/spool/torque/mom_logs/*
```

Upgrading Moab Workload Manager

The following instructions will guide you through a 6.1.x, 7.0.x, 7.1.x, 7.2.x, or 7.5.0 to 8.0.1 upgrade. Depending on which version of Moab you are presently running, upgrade instructions may vary, so unless otherwise noted, all instructions assume use of a RHEL operating system; notes for SLES users are added in appropriate places.

You might want to test the newest version of Moab on your system (before making the new version live) to verify your policies, scripts, and queues work the way you want them to.

If you are also upgrading TORQUE from an older version (pre-4.0), you may encounter a problem where Moab HPC Suite core files are regularly created in `/opt/moab`. This can be caused by old TORQUE library files used by Moab that try to authorize with the old TORQUE pbs_iff authorization daemon. You can resolve the problem by removing the old version library files from `/usr/local/lib`.

Because many system-level files and directories are accessed during the installation, the instructions in this guide should be executed with root privileges.

You will see that the instructions execute commands as the root user. Please note that the same commands will work for a non-root user with the `sudo` command.

To upgrade Moab

1. Untar the distribution file. For example:

```
[root]# tar -xzvf moab-<version>-<OS>.tar.gz
```

2. Change directory into the extracted directory.

3. Verify `/etc/yum.repos.d/epel.repo` exists and has the following lines. If not, create it and add these lines.

```
[epel]
name=Extra Packages for Enterprise Linux 6 - x86_64
mirrorlist=http://mirrors.fedoraproject.org/mirrorlist?repo=epel-6&arch=x86_64
failovermethod=priority
enabled=1
gpgcheck=1
gpgkey=http://download.fedoraproject.org/pub/epel/RPM-GPG-KEY-EPEL-6
```

SLES users must add a repository to YaST. The URL of the repository is http://download.opensuse.org/repositories/server:/database/SLE_11_SP2/.

4. Configure the installation package.

   Use the same configure options as when Moab was installed previously. If you cannot remember which options were used previously, check the `config.log` file in the directory where the previous version of Moab was installed from.

   For a complete list of configure options, use `./configure --help`.

5. Stop Moab.

```
[root]# mschedctl -k
moab will be shutdown immediately
```

While Moab is down, all currently running jobs continue to run on the nodes, the job queue remains intact, and new jobs cannot be submitted to Moab.

6. Before proceeding to the following steps, consider backing up your Moab Workload Manager home directory (`/opt/moab/` by default).

7. If you use ODBC, you must upgrade to the 8.0.1 schema. See **Migrating Your Database to Newer Versions of Moab** on page 787 for more information.

8. Run the make perldeps command to install the necessary perl modules using CPAN (and install CPAN if you have not already done so). When first running CPAN, you will be asked for configuration information. It is recommended that you choose an automatic configuration. You will be prompted to

provide input during module installation; running the make perldeps command with a script is not recommended.

```
[root]# yum install cpan
[root]# make perldeps
```

9. Install Moab.

```
[root]# make install
```

> Default configuration files are installed during make install. Existing configuration files are not overwritten and the new files are given a `.dist` extension.

10. Verify the version number is correct before starting the new server version.

```
[root]# moab --about

Defaults:   server=:42559  cfgdir=/opt/moab  vardir=/opt/moab
Build dir:  /tmp/develop
Build host: crom
Build date: Mon Jun 16 16:00:00 MST 2014
Build args: NA
Compiler Flags:  -D__M64 -D_BUILDDATETIME="2014061616" -DMUSEWEBSERVICES -
DMUSEZEROMQ -DMUSEMONGODB -DMMAX_GRES=512 -DMMAX_RANGE=2048 -DMMAX_TASK=32768 -fPIC
-gdwarf-3 -Wall -Wextra -DVALGRIND -x c++ -std=c++11 -DDMAX_PJOB=512 -D_GNU_SOURCE
Compiled as little endian.
Version: moab server master (revision 2014061616, changeset
90ce9f804ddd09b061238e438ecb4d117cc83e81)
Copyright (C) 2000-2014 by Adaptive Computing Enterprises, Inc. All Rights
Reserved.
```

11. If you are upgrading from Moab Workload Manager 7.5 or earlier and use Moab Accounting Manager with the native interface, (AMCFG[...] **TYPE**=*native*), locate the entries in `moab.cfg` with the following form:

```
AMCFG[mam] *URL=exec:///$HOME/tools/mam/*.*.mam.pl
```

Replace the matching entries with the following:

```
AMCFG[mam] CREATEURL=exec://$TOOLSDIR/mam/usage.quote.mam.pl
AMCFG[mam] STARTURL=exec://$TOOLSDIR/mam/usage.reserve.mam.pl
AMCFG[mam] PAUSEURL=exec://$TOOLSDIR/mam/usage.charge.mam.pl
AMCFG[mam] UPDATEURL=exec://$TOOLSDIR/mam/usage.charge.mam.pl
AMCFG[mam] RESUMEURL=exec://$TOOLSDIR/mam/usage.reserve.mam.pl
AMCFG[mam] ENDURL=exec://$TOOLSDIR/mam/usage.charge.mam.pl
AMCFG[mam] DELETEURL=exec://$TOOLSDIR/mam/lien.delete.mam.pl
```

12. Start Moab.

```
[root]# moabd
```

13. If you will be using Moab Web Services, you must configure a secret key. See **Securing communication using secret keys** on page 37.

Upgrading MWS

Context

Before upgrading MWS, we recommend you upgrade to Java 7 and MongoDB 2.4.x. To upgrade Java, repeat the Java installation instructions. To upgrade MongoDB, see **Upgrading MongoDB** on page 53.

It is highly recommended that you perform a full database backup before updating your database. This can be done using the `mongodump` utility documented in the MongoDB documentation.

To perform an MWS upgrade

1. Extract the contents of the MWS download tarball into a temporary directory. For example:

```
[root]# mkdir /tmp/mws-install
[root]# cd /tmp/mws-install
[root]# tar xvzf mws-8.0.1.tar.gz
[root]# cd /tmp/mws-install/mws-8.0.1
```

2. Stop Tomcat, re-deploy `mws.war`, and remove the exploded `mws` directory.

```
# CentOS 6 example

[root]# service tomcat6 stop
[root]# cp /tmp/mws-install/mws-8.0.1/mws.war /usr/share/tomcat6/webapps
[root]# rm -rf /usr/share/tomcat6/webapps/mws
```

3. Create the MWS home directory and subdirectories (for more information, see the "**Configuration** on page 1750" section of the *Moab Web Services Reference Guide*).

   The default location for the MWS home directory is `/opt/mws`. These instructions assume the default location.

   Here is a sample script for this setup:

```
[root]# mkdir -p \
    /opt/mws/etc/mws.d \
    /opt/mws/hooks \
    /opt/mws/log \
    /opt/mws/plugins \
    /opt/mws/spool/hooks \
    /opt/mws/utils
[root]# chown -R tomcat:tomcat /opt/mws # Depending on your OS, the Tomcat username
might be tomcat6.
[root]# chmod -R 555 /opt/mws
[root]# chmod u+w \
    /opt/mws/log \
    /opt/mws/plugins \
    /opt/mws/spool \
    /opt/mws/spool/hooks \
    /opt/mws/utils
```

4. Copy the extracted utility files to the utility directory created above and give the tomcat user ownership of the directory.

```
[root]# cd /tmp/mws-install/mws-8.0.1/utils
[root]# cp * /opt/mws/utils
[root]# chown tomcat:tomcat /opt/mws/utils/*
```

5. Set up the MWS configuration files. In the extracted directory are several configuration files.
   a. Merge the `/opt/mws/etc/mws-config.groovy.rpmnew` file with the old `/opt/mws/etc/mws-config.groovy` file by editing `/opt/mws/etc/mws-config.groovy`. (Note the addition of the "auditAppender" in the default logging configuration of `/opt/mws/etc/mws-config.groovy.rpmnew`.)

```
moab.messageQueue.port = 5563
moab.messageQueue.port = 5570

log4j = {
  // Configure an appender for the events log.
  def eventAppender = new org.apache.log4j.rolling.RollingFileAppender(
    name: 'events', layout: pattern(conversionPattern: "%m%n"))
  def rollingPolicy = new org.apache.log4j.rolling.TimeBasedRollingPolicy(
    fileNamePattern: '/opt/mws/log/events.%d{yyyy-MM-dd}',
    activeFileName: '/opt/mws/log/events.log')
  rollingPolicy.activateOptions()
  eventAppender.setRollingPolicy(rollingPolicy)

  // Configure an appender for the audit log.
  def auditAppender = new org.apache.log4j.rolling.RollingFileAppender(
      name: 'audit',
      layout: new com.ace.mws.logging.ACPatternLayout("%j\t\t\t%c{1}
\t\t\t%m%n"))
  def auditRollingPolicy = new org.apache.log4j.rolling.TimeBasedRollingPolicy(
      fileNamePattern: '/opt/mws/log/audit.%d{yyyy-MM-dd}',
      activeFileName: '/opt/mws/log/audit.log')
  auditRollingPolicy.activateOptions()
  auditAppender.setRollingPolicy(auditRollingPolicy)

  appenders {
    rollingFile name: 'stacktrace',
        file: '/opt/mws/log/stacktrace.log',
        maxFileSize: '100MB'
    rollingFile name: 'rootLog',
        file: '/opt/mws/log/mws.log',
        maxFileSize: '100MB', //The maximum file size for a single log file
        maxBackupIndex: 10, //Retain only the 10 most recent log files, delete
older logs to save space
        layout:pattern(conversionPattern: '%d %p %c %m%n'), //Configures the
output format of each log entry
        layout: new com.ace.mws.logging.ACPatternLayout(), //Configures the
output format of each log entry
        threshold: org.apache.log4j.Level.ERROR //Ignore any logging entries
less verbose than this threshold

    appender eventAppender
    appender auditAppender
  }

  // NOTE: This definition is a catch-all for any logger not defined below
  root {
    error 'rootLog'
  }

  // Individual logger configurations
  ...

  // Logs event information to the events log, not the rootLog
  trace additivity:false, events:'com.ace.mws.events.EventFlatFileWriter'
  // Logs audit information to the audit log, not the rootLog
  trace additivity:false, audit:'mws.audit'
}
```

Additions are noted in red. Removed content is stricken out.

If necessary, open port 5570 in the firewall.

Note that the **mws.suite** parameter and the **mam.*** parameters have been moved to a suite-specific file in `/opt/mws/etc/mws.d/` and do not need to exist in `/opt/mws/etc/mws-config.groovy`.

Also note the new ***messageQueue** parameters in `/opt/mws/etc/mws-config.groovy.rpmnew`. These are required and the value for **moab.messageQueue.secretKey** should match the value located in `/opt/moab/etc/moab-private.cfg`. If you have not yet configured a secret key, see **Securing communication using secret keys** on page 37.

b. Copy the `mws-config-hpc.groovy` file to `/opt/mws/etc/mws.d`.

c. Give the Tomcat user read access to `/opt/mws/etc/mws-config.groovy` and `/opt/mws/etc/mws.d/mws-config-*.groovy`.

6. Upgrade the schema of the `mws` database in MongoDB.

You *must* perform this step, regardless of whether you upgraded MongoDB to version 2.4.x or not.

Run the database migration script provided with MWS. (It is safe to run this script more than once. If for any reason, errors occur during the execution of the script, run it again.)

```
[root]# mongo -u mws_user mws /opt/mws/utils/db-migrate.js -p
```

The script might take several minutes to execute.

7. Start Tomcat.

```
[root]# service tomcat6 start
```

8. Visit http://localhost:8080/mws/ in a web browser to verify that MWS is running again.

You will see some sample queries and a few other actions.

9. Log into MWS to verify configuration. (The credentials are the values of **auth.defaultUser.username** and **auth.defaultUser.password** set in `/opt/mws/etc/mws-config.groovy`.)

If you encounter problems, or if MWS does not seem to be running, see the steps in **Troubleshooting** on page 87.

RPM installation

Installing Moab HPC Suite - Basic Edition

The RPM installation only supports installation on Red Hat 6.5, CentOS 6.5 or Scientific Linux 6.5. Use the Manual installation instructions if installing on other supported operating systems.

Because many system-level files and directories are accessed during the installation, the instructions in this guide should be executed with root privileges.

You will see that the instructions execute commands as the root user. Please note that the same commands will work for a non-root user with the `sudo` command.

Dependencies and packages installation

Install Java

Install the *Linux x64 RPM* version of Oracle® Java® 7 Runtime Environment.

Oracle Java 7 Runtime Environment is the recommended Java environment, but Oracle Java 6 is also supported. All other versions of Java, including OpenJDK/IcedTea, GNU Compiler for Java, and so on cannot run Moab Web Services.

Do the following:

1. Download the *Linux x64 RPM* version of Oracle Java SE 7 JRE. (Go to the Oracle Java 7 download page, copy the URL to the *Linux x64 RPM* version, then run the following command.)

   ```
   [root]# wget <URL> -O jre-7-linux-x64.rpm
   ```

   To verify that the download was successful, run the following on the RPM before installation:

   ```
   [root]# rpm -qip jre-7-linux-x64.rpm
   ```

2. Run the following to install Java 7:

   ```
   [root]# rpm -Uh jre-7-linux-x64.rpm
   ```

Opening ports

A few ports need to be available through your firewall so components of the suite can communicate with each other. Some features of some components might need additional ports configured. The individual component documentation indicates when additional ports are needed.

- 8080: Needed for Moab Web Services web portal (http)

To open ports in your firewall

- **Use `iptables` for Red Hat-based distributions**:

```
[[root]# iptables-save > /tmp/iptables.mod
[root]# vi /tmp/iptables.mod

# Add the following lines immediately *before* the line matching
# "-A INPUT -j REJECT --reject-with icmp-host-prohibited"
-A INPUT -p tcp --dport 8080 -j ACCEPT

[root]# iptables-restore < /tmp/iptables.mod
[root]# service iptables save
```

- **Use `SuSEfirewall2` for SuSE-based distributions**:

```
[root]# vi /etc/sysconfig/SuSEfirewall2

FW_SERVICES_EXT_TCP="8080"

[root]# service SuSEfirewall2_setup restart
```

Installing the RPM suite

> If you want to build a custom RPM for component documentation (Moab Workload Manager, Moab Web Services orMoab Accounting Manager, refer to the Manual Installation Guide and follow instructions in the respective component.

To install the RPM suite

1. If you are upgrading from a previous installation of Moab, back up your `/opt/moab/tools` directory to prevent losing modifications made to the perl scripts. If you are performing a clean installation of Moab HPC Suite, skip this step.

```
[root]# tar czf backup-tools.tar.gz /opt/moab/tools
```

2. Download the latest 8.0.1 RPM suite tarball (`moab-hpc-basic-suite-<version>-<timestamp>-<OS>.tar.gz`, for example) from the Adaptive Computing website.

3. Untar the downloaded package.

```
[root]# tar zxf moab-hpc-basic-suite-<version>-<timestamp>-<OS>.tar.gz
```

4. Change directories into the untarred directory.

> Consider reviewing the README file for additional details on using the RPM distribution tarball.

5. Install the suite repositories. The `-y` option installs with the default settings for the RPM suite.

For a description of the options of the repository installer script, run:

```
[root]# ./install-rpm-repos.sh -h
```

```
[root]# ./install-rpm-repos.sh [<repository-directory>] [-y]
```

If the installation returns the following warning line:

`Warning: RPMDB altered outside of yum.`

This is normal and can safely be ignored.

The [*<repository-directory>*] option is the directory where you want to copy the RPMs. If no argument is given, run "`install-rpm-repos.sh -h`" and note the default directory location. If the [*<repository-directory>*] already exists, RPMs will be added to the existing directory. No files are overwritten in [*<repository-directory>*]. A repository file is also created in `/etc/yum.repos.d/` and points to the [*<repository-directory>*] location.

For ease in repository maintenance, the install script fails if Adaptive Computing RPMs are copied to different directories. If a non-default [*<repository-directory>*] is specified, please use the same directory for future updates.

The script installs the `createrepo` package and its dependencies. You must answer "y" to all the questions in order for the RPM install of the suite to work. Additionally, the script installs the EPEL and 10gen repositories.

6. Test the repository:

```
[root]# yum search moab
```

If no error is given, the repository is correctly installed. The output will look similar to the following (varying slightly depending on the suite and build type):

```
...

moab-hpc-basic-suite.noarch : Moab HPC Basic Suite virtual package
moab-perl-RRDs.noarch : Moab RRDs
moab-tomcat-config.x86_64 : Tomcat Configuration for Moab Viewpoint and Web
Services
moab-verify-oracle-java.noarch : Java Validator for Moab Viewpoint and Web Services
moab-web-services.x86_64 : Moab Web Services
moab-workload-manager.x86_64 : Moab Workload Manager
moab-workload-manager-client.x86_64 : Moab Workload Manager Client
moab-workload-manager-common.x86_64 : Moab Workload Manager Common Files
moab-perl-data.noarch : Perl Configuration for perl packages by Adaptive Computing
moab-torque-client.x86_64 : TORQUE Client
moab-torque-common.x86_64 : TORQUE Common Files
moab-torque-devel.x86_64 : TORQUE Development Files
moab-torque-mom.x86_64 : TORQUE MOM agent
moab-torque-server.x86_64 : TORQUE Server
moab-web-services-hpc-configuration.x86_64 : MWS configuration for HPC
moab-workload-manager-hpc-configuration.x86_64 : MWM configuration for HPC
```

7. Install the suite package.

```
[root]# yum install moab-hpc-basic-suite
```

> If you encounter the following error:
>
> ```
> ...
>   --> Finished Dependency Resolution
> krb5-workstation-1.6.1-62.el5.x86_64 from installed has depsolving problems
>   --> Missing Dependency: krb5-libs = 1.6.1-62.el5 is needed by package
> krb5-workstation-1.6.1-62.el5.x86_64 (installed)
> krb5-workstation-1.6.1-62.el5.x86_64 from installed has depsolving problems
>   --> Missing Dependency: krb5-libs = 1.6.1-62.el5 is needed by package
> krb5-workstation-1.6.1-62.el5.x86_64 (installed)
> Error: Missing Dependency: krb5-libs = 1.6.1-62.el5 is needed by package
> krb5-workstation-1.6.1-62.el5.x86_64 (installed)
> You could try using --skip-broken to work around the problem
> You could try running: package-cleanup --problems
> package-cleanup --dupes
> rpm -Va --nofiles --nodigest
> ```
>
> Install the `krb5-workstation` package, then execute the install suite package again.
>
> ```
> [root]# yum install krb5-workstation
> [root]# yum install moab-hpc-basic-suite
> ```

> If you encounter CURL library errors, make sure you are installing the correct version for your OS.

8. Install and prepare the MongoDB database by doing the following:

   a. Install `mongo-10gen-server`.

```
[root]# yum install mongo-10gen-server --exclude mongodb-org,mongodb-org-server
```

> Running yum upgrade will replace MongoDB 2.4.x with a more recent, and incompatible version. Consider using yum version lock to maintain MongoDB 2.4.x.

   b. Start MongoDB.

RHEL and CentOS, and Scientific Linux:

```
[root]# chkconfig mongod on
[root]# service mongod start

[ OK ]
```

   c. Add the required MongoDB users.

> The passwords used below (`secret1`, `secret2`, and `secret3`) are examples. Choose your own passwords for these users.

```
[root]# mongo
> use admin;
> db.addUser("admin_user", "secret1");
> db.auth("admin_user", "secret1");

> use moab;
> db.addUser("moab_user", "secret2");
> db.addUser("mws_user", "secret3", true);

> use mws;
> db.addUser("mws_user", "secret3");
> exit
```

> Because the `admin_user` has read and write permissions to the `admin` database, it also has read and write permissions to all other databases. See Control Access to MongoDB Instances with Authentication for more information.

d. Enable authentication in MongoDB.

RHEL and CentOS, and Scientific Linux:

```
[root]# vi /etc/mongod.conf

auth = true

[root]# service mongod restart
```

Installing PostgreSQL

If you plan to use Moab Workload Manager with ODBC, you must install a PostgreSQL database.

To install PostgreSQL

1. Install and initialize PostgreSQL.

CentOS, RHEL, and Scientific Linux

```
[root]# yum install postgresql-server
[root]# service postgresql initdb
```

SLES

```
[root]# zypper install postgresql-server
[root]# service postgresql start
```

2. Configure trusted connections.

> Edit or add a "host" line in the pg_hba.conf file for the interface from which the server(s) (for example, Moab Workload Manager and/or Moab Accounting Manager) will be connecting to the database and ensure that it specifies a secure password-based authentication method (for example, md5).

```
[root]# vi /var/lib/pgsql/data/pg_hba.conf

# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
```

3. Configure PostgreSQL to accept connections from your host.

```
[root]# vi /var/lib/pgsql/data/postgresql.conf

# Uncomment the listen addresses line in the configuration:

listen_addresses = 'localhost'          # what IP address(es) to listen on;
```

4. Start or restart the database.

```
[root]# chkconfig postgresql on
[root]# service postgresql restart
```

Related topics

- **Welcome** on page xxi

Configuration

Configuring TORQUE

These instructions describe how to configure and start TORQUE.

Prerequisites

- TORQUE requires certain ports to be open for essential communication:
  - For client communication to `pbs_server`, all privileged ports must be open (ports under 1024).
  - For `pbs_server` communication to `pbs_mom`, the default port is 15003.
  - For `pbs_mom` to `pbs_server`, the default port is 15001.

  For more information on how to configure the ports that TORQUE uses for communication, see **Configuring Ports** on page 2204.

  > **Important**: If you intend to use TORQUE 5.0.x with Moab, you must run Moab version 8.0.x or later. TORQUE 5.0.x will not work with versions earlier than Moab 8.0.x.

- Make sure your host (with the correct IP address) is in your `/etc/hosts` file.

To configure TORQUE

1. Add nodes to the `/var/spool/torque/server_priv/nodes` file. For information on syntax and options for specifying compute nodes, see **Managing Nodes** on page 2268.

2. Start the servers.

```
[root]# service trqauthd start
[root]# service pbs_server start
[root]# service pbs_mom start
```

To configure MOMs

1. To set up the MOMs, in the directory of the unpackaged tarball, copy the `torque-common` and `torque-mom` RPM files to each MOM node.

```
[root]# scp RPMs/moab-torque-common-*.rpm <mom-node>
[root]# scp RPMs/moab-torque-mom-*.rpm <mom-node>
```

2. On each MOM node, install the RPMs, making sure that `torque-mom-common` is installed first.

```
[root]# ssh root@<mom-node>
[root]# yum install moab-torque-common-*.rpm moab-torque-mom-*.rpm
```

3. By default, on each MOM node, the `/var/spool/torque/server_name` file contains the hostname of the current host. If `pbs_server` is running on another host, change the name.

```
[root]# echo <pbs_server's_hostname> > /var/spool/torque/server_name
```

4. Edit the `/var/spool/torque/mom_priv/config` file on each node. This file is identical for all compute nodes and can be created on the head node and distributed in parallel to all systems.

```
[root]# vi /var/spool/torque/mom_priv/config

$pbsserver     headnode       # hostname running pbs server
$logevent      225            # bitmap of which events to log
```

Related topics

- **Installing Moab HPC Suite - Basic Edition** on page 62
- **Configuring Moab Workload Manager** on page 68
- **Component documentation** on page 95

Configuring Moab Workload Manager

These instructions describe how to configure and start Moab Workload Manager (Moab).

To configure Moab Workload Manager

1. Source the following file to add the Moab executable directories to your current shell *$PATH* environment.

```
[root]# . /etc/profile.d/moab.sh
```

2. Copy your license file into the same directory as `moab.cfg` (`/opt/moab/etc/` by default). For example:

```
[root]# cp moab.lic $MOABHOMEDIR/etc/moab.lic
```

To verify the current status of your license, use `moab --about`.

Moab checks the status of the license every day just after midnight. At 60 and 45 days before, and daily from 30 days before license expiration to and including the license expiration date, Moab sends an e-mail to all level 1 administrators informing them of the pending Moab license expiration. A log record is also made of the upcoming expiration event. For the notifications to occur correctly, you must enable administrator email notification (see "**Notifying Administrators of Failures** on page 684" in the Moab Workload Manager Administrator Guide) and `moab.cfg` must contain email addresses for level 1 administrators. For example:

```
ADMINCFG[1] USERS=u1,u2,u3[,...]

USERCFG[u1] EMAILADDRESS=u1@company.com
USERCFG[u2] EMAILADDRESS=u2@company.com
USERCFG[u3] EMAILADDRESS=u3@company.com

MAILPROGRAM DEFAULT
```

> Moab will not run without a license. For information about obtaining a trial license, please contact Adaptive Computing.

3. Start Moab.

```
[root]# chkconfig moab on
[root]# service moab start
```

4. If you have a resource manager configured, submit a sleep job as a non-root user and verify the job is running.

> If you do not have a resource manager configured, skip this step. For TORQUE, you can configure a basic queue.
>
> ```
> qmgr -c "set server scheduling=true"
> qmgr -c "create queue batch queue_type=execution"
> qmgr -c "set queue batch started=true"
> qmgr -c "set queue batch enabled=true"
> qmgr -c "set queue batch resources_default.nodes=1"
> qmgr -c "set queue batch resources_default.walltime=3600"
> qmgr -c "set server default_queue=batch"
> ```

```
[root]# su - moab
[moab]$ echo sleep 150 | msub
[moab]$ showq
[moab]$ exit
```

5. Connecting Moab to MongoDB

   If you will be installing Moab Web Services, connect Moab to MongoDB using the following instructions:

> The USEDATABASE parameter is unrelated to the MongoDB configuration.

a. In `/opt/moab/etc/moab.cfg`, set the **MONGOSERVER** parameter to the correct location of the MongoDB server. By default, Moab assumes it is on the same server.

```
MONGOSERVER <host>[:<port>]
```

b. In the `/opt/moab/etc/moab-private.cfg` file, set the **MONGOUSER** and **MONGOPASSWORD** parameters to the MongoDB `moab_user` credentials you set.

```
MONGOUSER     moab_user
MONGOPASSWORD secret2
```

c. Verify that Moab is able to connect to MongoDB.

```
[root]# service moab restart
[root]# mdiag -S
...
Mongo connection (localhost) is up (credentials are set)
...
```

Related topics

- **Installing Moab HPC Suite - Basic Edition** on page 62
- **Configuring TORQUE** on page 67
- **Component documentation** on page 95

Configuring Moab Web Services

These instructions describe how to configure Moab Web Services (MWS).

To configure Moab Web Services

1. Start Moab.

```
[root]# service moab start
```

2. Set up the MWS configuration file.

   a. In the `/opt/mws/etc/mws-config.groovy` file, change these settings:

   - **auth.defaultUser.username**: Any value you like, or leave as is.
   - **auth.defaultUser.password**: Any value you like, but choose a strong password.

```
[root]# vi /opt/mws/etc/mws-config.groovy


// Change these to be whatever you like.
auth.defaultUser.username = "moab-admin"
auth.defaultUser.password = "changeme!"
```

If you do not change **auth.defaultUser.password**, your MWS will not be secure (because anyone reading these instructions would be able to log into your MWS). Here are some tips for choosing a good password.

b. Do *one* of the following:

- If you are configuring an MWS connection to your LDAP server, add the following parameters:

```
ldap.server  = "192.168.0.5"
ldap.port = 389
ldap.baseDNs = ["dc=acme,dc=com"]
ldap.bindUser = "cn=Manager,dc=acme,dc=com"
ldap.password = "*****"
ldap.directory.type = "OpenLDAP Using InetOrgPerson Schema"
```

This is just an example LDAP connection. Be sure to use the appropriate domain controllers (dc) and common names (cn) for your environment.

If you followed the Adaptive Computing tutorial, **Setting up OpenLDAP on CentOS 6** on page 72, your **ldap.directory.type** should be set to "OpenLDAP Using InetOrgPerson Schema." However, the use of other schemas is supported. For more information see **LDAP Configuration using mws-config.groovy** on page 1380.

To see how to configure a secure connection to the LDAP server, see **Securing the LDAP connection** on page 1394.

- If you are configuring MWS to use PAM, add the the **pam.configuration.service** parameter to the `mws-config.groovy` file. For example:

```
pam.configuration.service = "login"
```

This is just an example PAM configuration file name. Make sure you specify the name of the configuration file you want MWS to use.

For more information about PAM configuration with MWS, see **PAM (pluggable authentication module) configuration using mws-config.groovy** on page 1383.

There is a security risk when authenticating local users through your PAM configuration. This behavior is highly discouraged and not supported by Adaptive Computing.

You can configure only one authentication method in `mws-config.groovy`—LDAP or PAM, but not both. If you have configured both LDAP and PAM, MWS defaults to using LDAP.

If you need multiple authentication methods, you must add them to your local PAM configuration. See your distribution documentation for details.

c. Add the **grails.mongo.username** and **grails.mongo.password** parameters to the mws-config.groovy file. Use the MWS credentials you added to MongoDB in the **Installing Moab HPC Suite - Basic Edition** on page 62section.

```
...
grails.mongo.username = "mws_user"
grails.mongo.password = "secret3"
```

3. Start or restart Tomcat.

```
[root]# chkconfig tomcat6 on
[root]# service tomcat6 restart
```

4. Navigate to `http://localhost:8080/mws/` in a web browser to verify that MWS is running (you will see some sample queries and a few other actions).

5. Log in to MWS to verify that your credentials are working. (Your login credentials are the `auth.defaultUser.username` and `auth.defaultUser.password` values you set in the `/opt/mws/etc/mws-config.groovy` file.)

> If you encounter problems, or if the application does not seem to be running, see the steps in**Moab Web Services issues** on page 91

Additional configuration

Configuring SSL in Tomcat

To configure SSL in Tomcat, please refer to the Apache Tomcat documentation.

Related topics

Setting up OpenLDAP on CentOS 6

These instructions are intended to help first-time LDAP administrators get up and running. The following procedures contain instructions for getting started using OpenLDAP on a CentOS 6 system. For more complete information on how to set up OpenLDAP see the OpenLDAP documentation.

> Adaptive Computing is not responsible for creating, maintaining, or supporting customer LDAP or Active Directory configurations.

Installing and configuring OpenLDAP on Centos 6

First, you will need to install OpenLDAP. These instructions explain how you can do this on a CentOS 6 system.

To install and configure OpenLDAP on Centos 6

1. Run the following command:

```
[root]# yum -y install openldap openldap-clients openldap-servers
```

2. Generate a password hash to be used as the admin password. This password hash will be used when you create the root user for your LDAP installation. For example:

```
[root]# slappasswd
New password : p@ssw0rd
Re-enter new password : p@ssw0rd
{SSHA}5lPFVw19zeh7LT53hQH69znzj8TuBrLv
```

3. Add the root user and the root user's password hash to the OpenLDAP configuration in the `olcDatabase={2}bdb.ldif` file. The root user will have permissions to add other users, groups, organizational units, etc. Do the following:

   a. Run this command:

   ```
   [root]# cd /etc/openldap/slapd.d/cn\=config
   [root]# vi olcDatabase\=\{2\}bdb.ldif
   ```

   b. If the **olcRootPW** attribute does not already exist, create it. Then set the value to be the hash you created from `slappasswd`. For example:

   ```
   olcRootPW: {SSHA}5lPFVw19zeh7LT53hQH69znzj8TuBrLv
   ...
   ```

4. While editing this file, change the distinguished name (DN) of the **olcSuffix** to something appropriate. The suffix typically corresponds to your DNS domain name, and it will be appended to the DN of every other LDAP entry in your LDAP tree.

For example, let's say your company is called Acme Corporation, and that your domain name is "acme.com." You might make the following changes to the `olcDatabase={2}bdb.ldif` file:

```
olcSuffix: dc=acme,dc=com
...
olcRootDN: cn=Manager,dc=acme,dc=com
...
olcRootPW: {SSHA}5lPFVw19zeh7LT53hQH69znzj8TuBrLv
...
```

Throughout the following examples in this topic, you will see `dc=acme,dc=com`. "acme" is only used as an example to illustrate what you would use as your own domain controller if your domain name was "acme.com." You should replace any references to "acme" with your own organization's domain name.

Do not set the cn of your root user to "root" (`cn=root,dc=acme,dc=com`), or OpenLDAP will have problems.

5. Modify the DN of the root user in the `olcDatabase={1}monitor.ldif` file to match the **olcRootDN** line in the `olcDatabase={2}bdb.ldif` file. Do the following:

   a. Run this command to edit the `olcDatabase={2}bdb.ldif` file:

   ```
   [root]# vi olcDatabase\=\{1\}monitor.ldif
   ```

   b. Modify the **olcAccess** line so that the **dn.base** matches the **olcRootDN** from the `olcDatabase={2}bdb.ldif` file. (In this example, **dn.base** should be "cn=Manager,dc=acme,dc=com".)

   ```
   olcAccess: {0}to * by
   dn.base="gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth" read by
   dn.base="cn=Manager,dc=acme,dc=com" read by * none
   ```

   c. Now the root user for your LDAP is `cn=Manager,dc=acme,dc=com`. The root user's password is the password that you entered using `slappasswd` (see step 2), which, in this example, is **p@ssw0rd**

6. Hide the password hashes from users who should not have permission to view them.

   A full discussion on configuring access control in OpenLDAP is beyond the scope of this tutorial. For help, see the OpenLDAP Access Control documentation.

   a. Run this command to edit the `oclDatabase\=\{2\}bdb.ldif` file:

   ```
   [root]# vi olcDatabase\=\{2\}bdb.ldif
   ```

   b. Add the following two lines to the end of the file to restrict users from viewing other users' password hashes.

```
olcAccess: {0}to attrs=userPassword by self write by
dn.base="cn=Manager,dc=acme,dc=com" write by anonymous auth by * none
olcAccess: {1}to * by dn.base="cn=Manager,dc=acme,dc=com" write by self write by
* read
```

These lines allow a user to read and write his or her own password. It also allows a manager to read and write anyone's password. Anyone, including anonymous users, is allowed to view non-password attributes of other users.

7. Make sure that OpenLDAP is configured to start when the machine starts up, and start the OpenLDAP service.

```
[root]# chkconfig slapd on
[root]# service slapd start
```

8. Now, you must manually create the "dc=acme,dc=com" LDAP entry in your LDAP tree.

   An LDAP directory is analogous to a tree. Nodes in this tree are called LDAP "entries" and may represent users, groups, organizational units, domain controllers, or other objects. The attributes in each entry are determined by the LDAP schema. In this tutorial we will build entries based on the InetOrgPerson schema (which ships with OpenLDAP by default).

   In order to build our LDAP tree we must first create the root entry. Root entries are usually a special type of entry called a domain controller (DC). Because we are assuming that the organization is called Acme Corporation, and that the domain is "acme.com," we will create a domain controller LDAP entry called `dc=acme,dc=com`. Again, you will need to replace "acme" with your organization's domain name. Also note that `dc=acme,dc=com` is what is called an LDAP distinguished name (DN). An LDAP distinguished name uniquely identifies an LDAP entry.

   Do the following:

   a. Create a file called `acme.ldif`. (You can delete this file once its content has been added to LDAP, so in this example, we will create it in the `/tmp` folder.)

```
[root]# cd /tmp
[root]# vi acme.ldif
```

   b. Add the following lines in `acme.ldif`:

```
dn: dc=acme,dc=com
objectClass: dcObject
objectClass: organization
dc: acme
o : acme
```

   c. Now add the contents of this file to LDAP. Run this command:

```
[root]# ldapadd -f acme.ldif -D cn=Manager,dc=acme,dc=com -w p@ssw0rd
```

   d. Verify that your entry was added correctly.

```
[root]# ldapsearch -x -LLL -b dc=acme,dc=com
dn: dc=acme,dc=com
objectClass: dcObject
objectClass: organization
dc: acme
o: acme
```

9. Run the following:

```
[root]# sudo iptables -L
[root]# sudo service iptables save
```

10. By default, the CentOS 6 firewall will block external requests to OpenLDAP. In order to allow MWS to access LDAP, you will have to configure your firewall to allow connections on port 389. (Port 389 is the default LDAP port.)

    Configuring your firewall is beyond the scope of this tutorial; however, it may be helpful to know that the default firewall on CentOS is a service called `iptables`. (For more information, see the documentation on iptables.) In the most basic case, you may be able to add a rule to your firewall that accepts connections to port 389 by doing the following:

    a. Edit your `iptables` file:

    ```
    [root]# vi /etc/sysconfig/iptables
    ```

    b. Add the following line *after* all the **ACCEPT** lines but *before* any of the **REJECT** lines in your `iptables` file:

    ```
    # ... lines with ACCEPT should be above
    -A INPUT -p tcp --dport 389 -j ACCEPT
    # .. lines with REJECT should be below
    ```

    For example, here is a sample `iptables` file with this line added:

    ```
    *filter
    :INPUT ACCEPT [0:0]
    :FORWARD ACCEPT [0:0]
    :OUTPUT ACCEPT [0:0]
    -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
    -A INPUT -p icmp -j ACCEPT
    -A INPUT -i lo -j ACCEPT
    -A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
    -A INPUT -p tcp --dport 389 -j ACCEPT
    -A INPUT -j REJECT --reject-with icmp-host-prohibited
    -A FORWARD -j REJECT --reject-with icmp-host-prohibited

    COMMIT
    ```

    c. Now reload `iptables`.

    ```
    [root]# service iptables reload
    ```

Although providing instructions is beyond the scope of this tutorial, it is also highly recommended that you set up OpenLDAP to use SSL or TLS security to prevent passwords and other sensitive data from being sent in plain text. For information on how to do this, see the OpenLDAP TLS documentation.

Now that you have installed and set up Open LDAP, you are ready to add organizational units (see **Adding an organizational unit (OU)** on page 77).

Adding an organizational unit (OU)

These instructions will describe how to populate the LDAP tree with organizational units (OUs), groups, and users, all of which are different types of LDAP entries. The examples that follow also presume an InetOrgPerson schema, because the InetOrgPerson schema is delivered with OpenLDAP by default.

To add an organizational unit (OU) entry to the LDAP tree

In this example, we are going to add an OU called "Users."

1. Create a temporary file called `users.ldif`. (You can delete this file once its content has been added to LDAP, so in this example, we will create it in the `/tmp` folder.)

```
[root]# cd /tmp
[root]# vi users.ldif
```

2. Add these lines to `users.ldif`:

```
dn: ou=Users,dc=acme,dc=com
objectClass: organizationalUnit
ou: Users
```

3. Add the contents of `users.ldif` file to LDAP.

```
[root]# ldapadd -f users.ldif -D cn=Manager,dc=acme,dc=com -w p@ssw0rd
```

Adding a user

To add a user to LDAP

In this example, we will add a user named "Bob Jones" to LDAP inside the "Users" OU.

1. Create a temporary file called `bob.ldif`. (You can delete this file once its content has been added to LDAP, so in this example, we will create it in the `/tmp` folder.)

```
[root]# cd /tmp
[root]# vi bob.ldif
```

2. Add these lines to `bob.ldif`:

```
dn: cn=Bob Jones,ou=Users,dc=acme,dc=com
cn: Bob Jones
sn: Jones
objectClass: inetOrgPerson
userPassword: p@ssw0rd
uid: bjones
```

3. Add the contents of `bob.ldif` file to LDAP.

```
[root]# ldapadd -f bob.ldif -D cn=Manager,dc=acme,dc=com -w p@ssw0rd
```

Adding a group

To add a group to LDAP

In this example, we will add a group called "Engineering" to LDAP inside the "Users" OU.

1. Create a temporary file called `engineering.ldif`. (You can delete this file once its content has been added to LDAP, so in this example, we will create it in the `/tmp` folder.)

```
[root]# cd /tmp
[root]# vi engineering.ldif
```

2. Add these lines to `engineering.ldif`:

```
dn: cn=Engineering,ou=Users,dc=acme,dc=com
cn: Engineering
objectClass: groupOfNames
member: cn=Bob Jones,ou=Users,dc=acme,dc=com
```

3. Add the contents of `engineering.ldif` file to LDAP.

```
[root]# ldapadd -f engineering.ldif -D cn=Manager,dc=acme,dc=com -w p@ssw0rd
```

Adding a user to a group

To add a user to an LDAP group

In this example, we will add an LDAP member named "Al Smith" to the "Engineering" LDAP group. This example assumes that user, Al Smith, has already been added to LDAP.

> Before you add a user to an LDAP group, the user must first be added to LDAP. For more information, see **Adding a user** on page 77.

1. Create a temporary file called `addUserToGroup.ldif`. (You can delete this file once its content has been added to LDAP, so in this example, we will create it in the `/tmp` folder.)

```
[root]# cd /tmp
[root]# vi addUserToGroup.ldif
```

2. Add these lines to `addUserToGroup.ldif`:

```
dn: cn=Engineering,ou=Users,dc=acme,dc=com
changetype: modify
add: member
member: cn=Al Smith,ou=Users,dc=acme,dc=com
```

3. Now add the contents of `addUserToGroup.ldif` file to LDAP.

```
[root]# ldapadd -f addUserToGroup.ldif -D cn=Manager,dc=acme,dc=com -w p@ssw0rd
```

Related topics

Trusting servers in Java

Prerequisites

Some of these instructions refer to `JAVA_HOME`, which must point to the same directory that Tomcat uses. To set `JAVA_HOME`, do this:

```
[root]# source /etc/tomcat6/tomcat6.conf
```

Your system administrator might have defined Tomcat's `JAVA_HOME` in a different file.

Retrieve the server's X.509 public certificate

To retrieve the server's certificate, use the following command:

```
[root]# $JAVA_HOME/bin/keytool -printcert -rfc -sslserver <servername>:<port> >
/tmp/public.cert.pem
```

Replace *<servername>* with the server's host name and *<port>* with the secure port number. The default port for https is 443. The default port for ldaps is 636. If successful, `/tmp/public.cert.pem` contains the server's public certificate. Otherwise, `/tmp/public.cert.pem` contains an error message. This message is typical: `keytool error: java.lang.Exception: No certificate from the SSL server`. This message suggests that the server name or port is incorrect. Consult your IT department to determine the correct server name and port.

Add the server's certificate to Java's keystore

Java stores trusted certificates in a database known as the keystore. Because each new version of Java has its own keystore, you need to add the server certificate to the Java keystore (using the steps below) every time you install a new version of Java.

Java's keystore is located at `$JAVA_HOME/lib/security/cacerts`. If Tomcat's `JAVA_HOME` points to a JDK, then the keystore is located at `$JAVA_HOME/jre/lib/security/cacerts`. To add the server certificate to the keystore, run the following command:

```
[root]# $JAVA_HOME/bin/keytool -import -trustcacerts -file /tmp/public.cert.pem -alias
<servername> -keystore $JAVA_HOME/lib/security/cacerts
```

You will be prompted for the keystore password, which is "changeit" by default.

Your system administrator might have changed this password.

After you've entered the keystore password, you'll see the description of the server's certificate. At the end of the description it prompts you to trust the certificate.

```
Trust this certificate? [no]:
```

Type `yes` and press **Enter** to add the certificate to the keystore.

Upgrading

Upgrading Moab HPC Suite - Basic Edition

It is highly recommended that you perform a full database backup before updating your database. This can be done using the `mongodump` utility documented in the MongoDB documentation.

This upgrade removes all roles and permissions and recreates the default roles. If you have modified any permissions or roles, you will need to recreate them after the upgrade is complete.

Because many system-level files and directories are accessed during the installation, the instructions in this guide should be executed with root privileges.

You will see that the instructions execute commands as the root user. Please note that the same commands will work for a non-root user with the `sudo` command.

To upgrade the RPM suite

1. Shut down all Adaptive applications.

```
[root]# service moab stop      # you can also run mschedctl -k
[root]# service tomcat6 stop
[root]# service pbs_server stop
```

2. Back up your `/opt/moab/tools` directory to prevent losing modifications made to the perl scripts.

```
[root]# tar czf backup-tools.tar.gz /opt/moab/tools
```

3. Download the latest 8.0.1 build executable (`moab-hpc-basic-suite-`*`<version>`*`-`*`<timestamp>`*`-`*`<OS>`*`.tar.gz`, for example) from the Adaptive Computing website.

   The variable marked *<version>* is the desired version of the suite; for example, 8.0-2014061017-8f96ac8d would be Moab 8.0 revision 2014061017 at changeset 8f96ac8d. The variable marked *<OS>* indicates which OS the build was designed for.

4. Untar the package.

```
[root]# tar xzf moab-hpc-basic-suite-<version>-<timestamp>-<OS>.tar.gz
```

5. Change directories into the root directory of the untarred directory.

   > Consider reviewing the README file for additional details on using the RPM distribution tarball.

6. Install the suite repositories.

   ```
   [root]# ./install-rpm-repos.sh [repository-directory] -y
   ```

   > The `-y` option will install with the default settings for the RPM suite.

   > The installation returns the following warning line:
   >
   > `Warning: RPMDB altered outside of yum.`
   >
   > This is normal and can safely be ignored.

   The *[<repository-directory>]* option is the directory where you want to copy the RPMs. If no argument is given, *[<repository-directory>]* defaults to `/opt/adaptive-rpm-repository/rpm`. If the *[<repository-directory>]* already exists, RPMs will be added to the existing directory. No files are overwritten in *[<repository-directory>]*. A repository file is also created in `/etc/yum.repos.d/` and points to the *[<repository-directory>]* location.

   For ease in repository maintenance, the install script fails if Adaptive Computing RPMs are copied to different directories. If a non-default *[<repository-directory>]* is specified, please use the same directory for future updates.

   The script installs the `createrepo` package and its dependencies. You must answer "y" to all the questions in order for the RPM install to work. Additionally, the script installs the EPEL and 10gen repositories.

7. Merge the new `.repo` files in `/etc/yum.repos.d/` with the existing ones.

   The `install-rpm-repos.sh` script will not overwrite existing RPM, GPG key or `.repo` files. Because some `.repo` files may have changed from previous releases, some merging of the `.repo` files is necessary. The newest files will have the `.new` extension.

   Please compare older `.repo` files with the newer ones to ensure that the latest changes are reflected. In some cases, there is no change, and you can remove the new file. In most cases, however, it is safe to overwrite the old `.repo` file with the new one. For example:

   ```
   [root]# mv /etc/yum.repos.d/AC.repo.new /etc/yum.repos.d/AC.repo
   ```

   After making changes in the `/etc/yum.repos.d` directory, make sure to run the following command to update the `yum` cache:

   ```
   [root]# yum clean all
   ```

8. Install the 8.0.1 suite packages.

```
[root]# yum install moab-hpc-basic-suite
```

> The Moab and MWS RPMs automatically create a backup of all relevant files. These backups are stored in /var/tmp/backup-*<rpmName>*-*<timestamp>*.tar.gz.
>
> If changes are detected between any existing configuration files and new configuration files, a version of the new configuration file will be saved under *<configurationFileLocation>*/*<fileName>*.rpmnew.

9. Upgrade the mongo database. Adaptive Computing strongly recommends MongoDB version 2.4.x. New versions of MongoDB are not supported. When upgrading, you must add 'exclude=mongodb-org mongodb-org-server' to the mongo.repo file to maintain 2.4.x:

```
name=MongoDB Repository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64/
gpgcheck=0
enabled=1
exclude=mongodb-org mongodb-org-server
```

> Running yum upgrade without 'exclude=mongodb-org mongodb-org-server', will replace 2.4.x with a more recent, and incompatible version.

Support for environments using 2.0 is now deprecated and will be removed in future releases. See **Upgrading from MongoDB 2.0 to 2.4.x** on page 86 for instructions on updating from 2.0 to 2.4.x

10. Upgrade the schema of the mws database in MongoDB.

> You *must* perform this step, regardless of whether you upgraded MongoDB to version 2.4.x or not. (See previous step.)

> Before updating this database, you should perform a full backup. This can be done by using the mongodump utility documented in the MongoDB documentation.

Run the database migration script provided with MWS. (It is safe to run this script more than once. If for any reason, errors occur during the execution of the script, run it again.)

```
[root]# mongo -u mws_user mws /opt/mws/utils/db-migrate.js -p
```

> Depending on the number of events and services in the system, the script may take several minutes to execute.

11. (Optional, but recommended) You must update the moab-verify-oracle-java RPM package before you can proceed with the Java 7 upgrade. You may need to remove Java 6 before upgrading to Java 7.

Oracle Java 7 Runtime Environment is the recommended Java environment, but Java 6 is also supported. All other versions of Java, including OpenJDK/IcedTea, GNU Compiler for Java, etc. cannot run Moab Web Services.

Do the following:

a. Update the `moab-verify-oracle-java` RPM package.

```
[root]# yum update moab-verify-oracle-java
```

b. Download the *Linux x64 RPM* version of Oracle Java SE 7 JRE. (Go to the Oracle Java 7 download page, copy the URL to the *Linux x64 RPM* version, then run the following command.)

```
[root]# wget <URL> -O jre-7-linux-x64.rpm
```

To verify that the download was successful, run the following on the RPM before installation:

```
[root]# rpm -qip jre-7-linux-x64.rpm
```

c. Run the following to install Java 7:

```
[root]# rpm -Uh jre-7-linux-x64.rpm
```

12. Merge the configuration files.

Whether or not to start with the old configuration file and add newer configuration options (or vice versa) depends on the amount of customization you previously made in earlier versions. In instances where you have modified very little, you should consider using the newer configuration and merging site-specific settings from the old file into the new one. The following steps highlight important changes between the 7.2.x default configuration and the 8.0.1 default configuration. Also note that new configuration files may have auto-generated content for secret keys and default passwords—be careful to ensure that secret keys shared between components are configured correctly.

The recommended layout for the `/opt/moab/etc/` directory appears as follows:

```
[root]# ls -l /opt/moab/etc
total 29
-rw-r--r--. 1 root moab  2323 Nov 13 13:41 config.moab.pl
-rw-r--r--. 1 root moab   989 Nov 13 13:41 config.sql.pl
lrwxrwxrwx. 1 root root    14 Nov 13 15:46 moab.cfg -> moab.hpc.cfg
-rw-r--r--. 1 root moab 23500 Nov 13 15:43 moab.hpc.cfg
drwxr-xr-x. 2 root moab  4096 Nov 13 15:41 moab.d
-rw-r--r--. 1 root moab   391 Nov 13 13:41 moab.dat
-r--r--r--. 1 root root   493 Nov  6 16:14 moab.lic
-rw-------. 1 root moab   288 Nov 13 15:39 moab-private.cfg
lrwxrwxrwx. 1 root root    14 Nov 13 15:46 nami.cfg -> nami.hpc.cfg
-rw-r--r--. 1 root moab   563 Nov 13 15:43 nami.hpc.cfg
```

Do the following:

a. Merge the /opt/moab/etc/moab-private.cfg file. Make sure that unique items in /opt/moab/etc/moab-private.cfg.rpmnew are added to the existing /opt/moab/etc/moab-private.cfg file. Include the new MWS RM credentials if you configure MWS as a resource manager in /opt/moab/etc/moab.cfg:

```
CLIENTCFG[RM:mws] USERNAME=moab-admin PASSWORD=changeme!
```

> The default MWS credentials in 7.2.x were *admin:adminpw*. These have been changed in 8.0.1 to *moab-admin:changeme!* in order to mirror the Viewpoint default credentials. Use whatever credentials you have configured in /opt/mws/etc/mws-config.groovy.

b. Merge customizations from /opt/moab/etc/moab.cfg and /opt/moab/etc/moab.d/* into /opt/moab/etc/moab.hpc.cfg.

   Although there are several ways to configure and merge changes into the /opt/moab/etc/moab.cfg file, the following instructions outline the recommended best practices. *Deviations from these best practices may result in unexpected behavior or added difficulty in future upgrades.*

   It is best to use the new default configuration file (/opt/moab/etc/moab.hpc.cfg) and merge changes from previous files into that one. You will notice that content from the /opt/moab/etc/moab.d/ directory has been merged into /opt/moab/etc/moab.hpc.cfg. Ensure that custom configuration options in all files located in /opt/moab/etc/moab.d/ directory get merged in to /opt/moab/etc/moab.hpc.cfg.

   You should avoid #include configurations.

   Although the upgrade should have created a backup of the moab.cfg file (in /var/tmp/backup-<*rpmName*>-<*timestamp*>.tar.gz), it is best to create your own backup until you can confirm the updated configuration behaves as expected.

```
[root]# mv /opt/moab/etc/moab.cfg /opt/moab/etc/moab.hpc.cfg.bak
```

   Once the changes have been merged to /opt/moab/etc/moab.hpc.cfg, configure Moab to use the new file. The recommended configuration is to use a symlink called /opt/moab/etc/moab.cfg that points to /opt/moab/etc/moab.hpc.cfg.

```
[root]# ln -s /opt/moab/etc/moab.hpc.cfg /opt/moab/etc/moab.cfg
```

c. Merge the /opt/mws/etc/mws-config.groovy file.

   Merge the /opt/mws/etc/mws-config.groovy.rpmnew file with the old /opt/mws/etc/mws-config.groovy file by editing /opt/mws/etc/mws-config.groovy. (Note the addition of the "auditAppender" in the default logging configuration of /opt/mws/etc/mws-config.groovy.rpmnew.)

```
moab.messageQueue.port = 5563
moab.messageQueue.port = 5570

log4j = {
  // Configure an appender for the events log.
  def eventAppender = new org.apache.log4j.rolling.RollingFileAppender(
    name: 'events', layout: pattern(conversionPattern: "%m%n"))
  def rollingPolicy = new org.apache.log4j.rolling.TimeBasedRollingPolicy(
    fileNamePattern: '/opt/mws/log/events.%d{yyyy-MM-dd}',
    activeFileName: '/opt/mws/log/events.log')
  rollingPolicy.activateOptions()
  eventAppender.setRollingPolicy(rollingPolicy)

  // Configure an appender for the audit log.
  def auditAppender = new org.apache.log4j.rolling.RollingFileAppender(
      name: 'audit',
      layout: new com.ace.mws.logging.ACPatternLayout("%j\t\t\t%c{1}
\t\t\t%m%n"))
  def auditRollingPolicy = new org.apache.log4j.rolling.TimeBasedRollingPolicy(
      fileNamePattern: '/opt/mws/log/audit.%d{yyyy-MM-dd}',
      activeFileName: '/opt/mws/log/audit.log')
  auditRollingPolicy.activateOptions()
  auditAppender.setRollingPolicy(auditRollingPolicy)

  appenders {
    rollingFile name: 'stacktrace',
        file: '/opt/mws/log/stacktrace.log',
        maxFileSize: '100MB'
    rollingFile name: 'rootLog',
        file: '/opt/mws/log/mws.log',
        maxFileSize: '100MB', //The maximum file size for a single log file
        maxBackupIndex: 10, //Retain only the 10 most recent log files, delete
older logs to save space
        layout:pattern(conversionPattern: '%d %p %c %m%n'), //Configures the
output format of each log entry
        layout: new com.ace.mws.logging.ACPatternLayout(), //Configures the
output format of each log entry
        threshold: org.apache.log4j.Level.ERROR //Ignore any logging entries
less verbose than this threshold

    appender eventAppender
    appender auditAppender
  }

  // NOTE: This definition is a catch-all for any logger not defined below
  root {
    error 'rootLog'
  }

  // Individual logger configurations
  ...

  // Logs event information to the events log, not the rootLog
  trace additivity:false, events:'com.ace.mws.events.EventFlatFileWriter'
  // Logs audit information to the audit log, not the rootLog
  trace additivity:false, audit:'mws.audit'
                                       }
```

Additions are noted in red. Removed content is stricken out.

> If necessary, open port 5570 in the firewall.

Note that the **mws.suite** parameter and the **mam.*** parameters have been moved to a suite-specific file in `/opt/mws/etc/mws.d/` and do not need to exist in `/opt/mws/etc/mws-config.groovy`.

Also note note the new ***messageQueue** parameters in `/opt/mws/etc/mws-config.groovy.rpmnew`. These are required and the **moab.messageQueue.secretKey** value should match the value located in `/opt/moab/etc/moab-private.cfg`.

13. Start all Adaptive applications.

```
[root]# service pbs_server start
[root]# service moab start
[root]# service tomcat6 start
```

Related topics

- [**Welcome** on page xxi]

Upgrading from MongoDB 2.0 to 2.4.x

Adaptive Computing strongly recommends MongoDB version 2.4.x. Support for environments using 2.0 is now deprecated and will be removed in future releases. Please refer to docs.mongodb.org for instructions on how to upgrade MongoDB. Note that you must pay close attention to the information regarding instances with `auth` enabled (as this is the recommended setup for Moab HPC Suite).

```
[root]# service mongod stop
[root]# yum remove mongo20-10gen-server mongo20-10gen
[root}# yum install mongo-10gen-server --exclude mongodb-org,mongodb-org-server
[root]# service mongod start
```

> Note that the settings in the `/etc/mongod.conf` file were saved in `/etc/mongod.conf.rpmsave` while removing MongoDB 2.0. You may need to be restore any custom settings after MongoDB 2.4.x is installed in the new `/etc/mongod.conf` file (for example, "auth = true").

After upgrading from 2.0 to 2.4.x, you should verify that the MongoDB credentials were preserved:

```
[root]# mongo -u mws_user mws -p
MongoDB shell version: 2.4.8
connecting to: mws
> show collections
event
mongeez
permission
...
```

Troubleshooting

This page details some common problems and general solutions. It contains these sections:

- **General issues** on page 87
- **Moab Web Services issues** on page 91

General issues

- **Moab error: "cannot determine local hostname"** on page 87
- **Moab error: "Moab will now exit due to license file not found"** on page 88
- **Other Moab issues** on page 88
- **Where do I change my passwords?** on page 89

Moab error: "cannot determine local hostname"

```
# service moab start
Starting moab: ERROR:    cannot determine local hostname - node is misconfigured
                                            [FAILED]
```

If you encounter this error when starting Moab, check the `/opt/moab/etc/moab.cfg` file to make sure a valid host is configured. For example:

```
...
SCHEDCFG[Moab]                           SERVER=<moab-hostname>:42559
...
```

Also check `/etc/hosts` to be sure the host name resolves, at least with localhost:

```
...
127.0.0.1    <moab-hostname> localhost localhost.localdomain localhost4
localhost4.localdomain4
...
```

Moab error: "Moab will now exit due to license file not found"

```
# service moab start
Starting moab: Moab will now exit due to license file not found
Please contact Adaptive Computing (sales@adaptivecomputing.com) to get a license
for your system
                                        [FAILED]
```

If you encounter this error when starting Moab, make sure your Moab license file is named **moab.lic** and is located in the `/opt/moab/etc/` directory.

Also make sure the license is not expired. The expiration date is listed in the license file. For example:

```
# cat /opt/moab/etc/moab.lic
...
# Expires after Tue Dec 31 10:43:46 2013
...
```

Other Moab issues

Please see "Troubleshooting and System Maintenance" in the Moab Workload Manager*Administrator Guide*.

Where do I change my passwords?

MWS super user username and password

The default username and password for MWS are **moab-admin** and **changeme!** (respectively).

To change the username and/or the password for the MWS super user:

1. Stop the `tomcat6` and `moab` services.

```
[root]# service moab stop
[root]# service tomcat6 stop
```

2. Change the respective values in the following files:

   - `/opt/mws/etc/mws-config.groovy`:

   ```
   auth.defaultUser.username = "moab-admin"
   auth.defaultUser.password = "changeme!"
   ```

   - `/opt/moab/etc/moab-private.cfg`:

   ```
   CLIENTCFG[RM:mws] USERNAME=moab-admin PASSWORD=changeme!
   ```

   - `/opt/moab/etc/cloud.cfg`:

   ```
   CONFIG[default]      MWS_USERNAME=moab-admin
   CONFIG[default]      MWS_PASSWORD=changeme!
   ```

3. Start the `tomcat6` service.

```
[root]# service tomcat6 start
```

4. Start the `moab` service.

```
[root]# service moab start
```

MongoDB passwords

To change the passwords for MongoDB:

1. Stop the `tomcat6` and `moab` services.

```
[root]# service moab stop
[root]# service tomcat6 stop
```

2. Change the passwords for the MongoDB accounts (i.e., **moab_user** and/or **mws_user**). For instructions, see the MongoDB documentation.

3. Edit the password values in the following files:

   - `/opt/moab/etc/moab-private.cfg`:

   ```
   MONGOUSER            moab_user
   MONGOPASSWORD        secret2
   ```

   - `/opt/mws/etc/mws-config.groovy`:

```
// MongoDB configuration.
grails.mongo.username = "mws_user"
grails.mongo.password = "secret3"
```

4. Start the `tomcat6` service.

```
[root]# service tomcat6 start
```

5. Start the `moab` service.

```
[root]# service moab start
```

Moab Web Services issues

If something goes wrong with MWS, look in the following files:

- The MWS log file. By default this is `/opt/mws/log/mws.log`.
- The Tomcat `catalina.out` file, usually in `/var/log/tomcat6 or $CATALINA_HOME/logs`.

> If you remove the `log4j` configuration from `mws-config.groovy`, MWS writes its log files to `java.io.tmpdir`. For Tomcat, `java.io.tmpdir` is generally set to `$CATALINA_BASE/temp` or `CATALINA_TMPDIR`.

Here is a list of some errors and their fixes:

- **MongoDB: Errors during MWS startup** on page 92
- **MongoDB: Out of semaphores to get db connection** on page 94
- **MongoDB: Connection wait timeout after 120000 ms** on page 94
- **java.lang.OutOfMemoryError: Java heap space** on page 94
- **java.lang.OutOfMemoryError: PermGen space** on page 95
- **SEVERE: Context [/mws] startup failed due to previous errors** on page 95
- **Moab Reached Maximum Number of Concurrent Client Connections** on page 95

MongoDB: Errors during MWS startup

If the application fails to start and gives error messages such as these:

```
Error creating bean with name 'mongoDatastore'
can't say something; nested exception is com.mongodb.MongoException
```

```
ERROR   grails.app.services.com.ace.mws.ErrorService    0
   Error encountered while attempting to authenticate account or query database;
the MongoDB server is not available. Please verify connection to server
'/127.0.0.1:27017' and that MongoDB is running.
```

MongoDB is most likely not running, or the MongoDB host and port are misconfigured.

In this case, there are a few things to verify:

- (*Not* relevant if MongoDB is installed on a separate server) **Is MongoDB installed?**

  Run the following commands to assess whether MongoDB is installed on the current server.

  ```
  $ mongo
  -bash: mongo: command not found
  ```

  To remedy, install MongoDB, start the `mongod` service and then restart the `tomcat6` service. See **Preparing for installation** on page 24 or **Installing Moab HPC Suite - Basic Edition** on page 62 for more information on how to install and configure MongoDB in the manual installation guide or the RPM-based installation guide, respectively.

- (*Only* relevant if MongoDB is installed on a separate server) **Is MWS configured to connect to the remote MongoDB server?**

  Run the following commands to assess whether MongoDB is installed on the current server.

  ```
  [root]# cat /opt/mws/etc/mws-config.groovy | grep 'grails.mongo'
  // grails.mongo.username = "mws_user"
  // grails.mongo.password = "<ENTER-KEY-HERE>"
  // grails.mongo.host = "127.0.0.1"
  // grails.mongo.port = 27017
  ```

  Make sure that the `grails.mongo.*` options are configured in `/opt/mws/etc/mws-config.groovy` for the remote MongoDB server and then restart the `tomcat6` service.

  ```
  [root]# service tomcat6 restart
  ```

- **Is MWS configured to authenticate with MongoDB, and is MongoDB configured to enforce authentication?**

  Run the following commands to assess the relevant MWS and MongoDB configurations.

  ```
  [root]# cat /opt/mws/etc/mws-config.groovy | grep 'grails.mongo'
  // grails.mongo.username = "mws_user"
  // grails.mongo.password = "<ENTER-KEY-HERE>"

  [root]# cat /etc/mongod.conf | grep 'auth'
  #noauth = true
  auth = true
  ```

  The configuration above is problematic because the `grails.mongo` credentials are commented out in the `/opt/mws/etc/mws-config.groovy` file while MongoDB is configured to enforce authentication ("auth = true"). Similar connection issues will exist if the `grails.mongo` parameters do not match the credentials configured for the "mws_user" on both the `mws` and `moab` databases in MongoDB.

(For upgrade scenarios only) If the application fails to start and gives the following message in `/opt/mws/etc/log/mws.log`:

```
java.lang.Exception: The db-migrate.js script has not yet been run. Please see the
upgrade section of the installation guide for instructions.
```

Then the `db-migrate.js` script must be run to update the schema of the `mws` database in MongoDB.

MongoDB: Out of semaphores to get db connection

To resolve this error, adjust the values of `connectionsPerHost` or `threadsAllowedToBlockForConnectionMultiplier` by adding them to `mws-config.groovy`. Example:

```
grails.mongo.options.connectionsPerHost = 60
grails.mongo.options.threadsAllowedToBlockForConnectionMultiplier = 10
```

For more information on these options, refer to these documents:

- "Configuration" (in the Moab Web Services *Reference Guide*), which briefly discusses a few MongoDB driver options.
- The MongoOptions documentation, which contains full details on all MongoDB driver options.

> You must restart Tomcat after adding, removing, or changing **grails.mongo.options** parameters.
>
> As shipped, `mws-config.groovy` does not contain any **grails.mongo.options** parameters. To adjust their values, you need to add them to `mws-config.groovy`.
>
> The default value of **connectionsPerHost** is normally 10, but MWS sets it internally to 50.
>
> The default value of **threadsAllowedToBlockForConnectionMultiplier** is 5.
>
> Any of the options listed in MongoOptions can be specified in `mws-config.groovy`. Just use the prefix **grails.mongo.options** as shown above.

MongoDB: Connection wait timeout after 120000 ms

See **MongoDB: Out of semaphores to get db connection** above.

java.lang.OutOfMemoryError: Java heap space

Increase the size of the heap using JVM options **-Xms** and **-Xmx**. Here are the suggested values:

```
CATALINA_OPTS="-DMWS_HOME=/opt/mws -Xms256m-Xmx3g -XX:MaxPermSize=384m"
```

- -Xms: Set initial Java heap size.
- -Xmx: Set maximum Java heap size.

java.lang.OutOfMemoryError: PermGen space

Increase the size of the permanent generation using JVM option **-XX:MaxPermSize**. Here are the suggested values:

```
CATALINA_OPTS="-DMWS_HOME=/opt/mws -Xms256m -Xmx3g -XX:MaxPermSize=384m"
```

SEVERE: Context [/mws] startup failed due to previous errors

If `catalina.out` contains this error, look in `/opt/mws/log/mws.log` and `/opt/mws/log/stacktrace.log` for more details on the error.

Also ensure that the `/opt/mws/etc/mws-config.groovy` file can be read by the Tomcat user. The permissions should appear as follows:

```
$ ls -al /opt/mws/etc/mws-config.groovy
-r-------- 1 tomcat tomcat 4056 Dec  4 12:07 mws-config.groovy
```

Moab Reached Maximum Number of Concurrent Client Connections

When this error message is encountered, simply add a new line to the `moab.cfg` file:

```
CLIENTMAXCONNECTIONS 256
```

This will change the Moab configuration when Moab is restarted. Run the following command to immediately use the new setting:

```
[root]# changeparam CLIENTMAXCONNECTIONS 256
```

> The number **256** above may be substituted for the desired maximum number of Moab client connections.

Component documentation

The individual components of the suite have more options and allow for more configuration than can be contained in this guide. Refer to the individual component guides for more information.

TORQUE

- TORQUE 5.0.1 Administrator Guide: HTML – PDF

Moab Workload Manager

- Moab Workload Manager 8.0.1 Administrator Guide: HTML

Moab Web Services

- Moab Web Services 8.0.1 Reference Guide: HTML

Related topics

- **Preparing for installation** on page 24
- **Installing Moab HPC Suite - Basic Edition** on page 62

- **Welcome** on page xxi

Moab Workload Manager

Moab Workload Manager overview

Moab Workload Manager is a highly advanced scheduling and management system designed for clusters, grids, and on-demand/utility computing systems. At a high level, Moab applies site policies and extensive optimizations to orchestrate jobs, services, and other workload across the ideal combination of network, compute, and storage resources. Moab enables true adaptive computing allowing compute resources to be customized to changing needs and failed systems to be automatically fixed or replaced. Moab increases system resource availability, offers extensive cluster diagnostics, delivers powerful QoS/SLA features, and provides rich visualization of cluster performance through advanced statistics, reports, and charts.

Moab works with virtually all major resource management and resource monitoring tools. From hardware monitoring systems like IPMI to provisioning systems and storage managers, Moab takes advantage of domain expertise to allow these systems to do what they do best, importing their state information and providing them with the information necessary to better do their job. Moab uses its global information to coordinate the activities of both resources and services, which optimizes overall performance in-line with high-level mission objectives.

Related topics

-

Philosophy

The scheduler's purpose is to optimally use resources in a convenient and manageable way. System users want to specify resources, obtain quick turnaround on their jobs, and have reliable resource allocation. On the other hand, administrators want to understand both the workload and the resources available. This includes current state, problems, and statistics—information about what is happening that is transparent to the end-user. Administrators need an extensive set of options to enable management enforced policies and tune the system to obtain desired statistics.

There are other systems that provide batch management; however, Moab is unique in many respects. Moab matches jobs to nodes, dynamically reprovisions nodes to satisfy workload, and dynamically modifies workload to better take advantage of available nodes. Moab allows sites to fully visualize cluster and user behavior. It can integrate and orchestrate resource monitors, databases, identity managers, license managers, networks, and storage systems, thus providing a cohesive view of the cluster—a cluster that fully acts and responds according to site mission objectives.

Moab can dynamically adjust security to meet specific job needs. Moab can create real and virtual clusters on demand and from scratch that are custom-tailored to a specific request. Moab can integrate visualization services, web farms and application servers; it can create powerful grids of disparate clusters. Moab maintains complete accounting and auditing records, exporting this data to information services on command, and even providing professional billing statements to cover all used resources and services.

Moab provides user- and application-centric web portals and powerful graphical tools for monitoring and controlling every conceivable aspect of a cluster's objectives, performance, workload, and usage. Moab is unique in its ability to deliver a powerful user-centric cluster with little effort. Its design is focused on ROI, better use of resources, increased user effectiveness, and reduced staffing requirements.

This chapter contains these sections:

Value of a Batch System

Batch systems provide centralized access to distributed resources through mechanisms for submitting, launching, and tracking jobs on a shared resource. This greatly simplifies use of the cluster's distributed resources, allowing users a *single system image* in terms of managing jobs and aggregate compute resources available. Batch systems should do much more than just provide a global view of the cluster, though. Using compute resources in a fair and effective manner is complex, so a scheduler is necessary to determine when, where, and how to run jobs to optimize the cluster. Scheduling decisions can be categorized as follows:

- Traffic Control
- Mission Policies
- Optimizations

Traffic Control

A scheduler must prevent jobs from interfering. If jobs contend for resources, cluster performance decreases, job execution is delayed, and jobs may fail. Thus, the scheduler tracks resources and dedicates requested resources to a particular job, which prevents use of such resources by other jobs.

Mission Policies

Clusters and other HPC platforms typically have specific purposes; to fulfill these purposes, or mission goals, there are usually rules about system use pertaining to who or what is allowed to use the system. To be effective, a scheduler must provide a suite of policies allowing a site to *map* site mission policies into scheduling behavior.

Optimizations

The compute power of a cluster is a limited resource; over time, demand inevitably exceeds supply. Intelligent scheduling decisions facilitate higher job volume and faster job completion. Though subject to the constraints of the traffic control and mission policies, the scheduler must use whatever freedom is available to maximize cluster performance.

Philosophy and Goals

Managers want high system utilization and the ability to deliver various qualities of service to various users and groups. They need to understand how available resources are delivered to users over time. They also need administrators to tune *cycle delivery* to satisfy the current site mission objectives.

Determining a scheduler's success is contingent upon establishing metrics and a means to measure them. The value of statistics is best understood if optimal statistical values are known for a given environment, including workload, resources, and policies. That is, if an administrator could determine that a site's typical workload obtained an average queue time of 3.0 hours on a particular system, that would be a useful *statistic*; however, if an administrator knew that through proper tuning the system could deliver an average queue time of 1.2 hours with minimal negative side effects, that would be valuable *knowledge*.

Moab development relies on extensive feedback from users, administrators, and managers. At its core, it is a tool designed to *manage* resources and provide meaningful information about what is actually happening on the system.

Management Goals

A manager must ensure that a cluster fulfills the purpose for which it was purchased, so a manager must deliver cycles to those projects that are most critical to the success of the funding organizations. Management tasks to fulfill this role may include the following:

- Define cluster mission objectives and performance criteria
- Evaluate current and historical cluster performance
- Instantly graph delivered service

Administration Goals

An administrator must ensure that a cluster is effectively functioning within the bounds of the established mission goals. Administrators translate goals into cluster policies, identify and correct cluster failures, and train users in best practices. Given these objectives, an administrator may be tasked with each of the following:

- Maximize utilization and cluster responsiveness
- Tune fairness policies and workload distribution
- Automate time-consuming tasks
- Troubleshoot job and resource failures

- Instruct users of available policies and in their use regarding the cluster
- Integrate new hardware and cluster services into the batch system

End-user Goals

End-users are responsible for learning about the resources available, the requirements of their workload, and the policies to which they are subject. Using this understanding and the available tools, they find ways to obtain the best possible responsiveness for their own jobs. A typical end-user may have the following tasks:

- Manage current workload
- Identify available resources
- Minimize workload response time
- Track historical usage
- Identify effectiveness of prior submissions

Workload

Moab can manage a broad spectrum of compute workload types, and it can optimize all four workload types within the same cluster simultaneously, delivering on the objectives most important to each workload type. The workload types include the following:

- Batch Workload
- Interactive Workload
- Calendar Workload
- Service Workload

Batch Workload

Batch workload is characterized by a *job* command file that typically describes all critical aspects of the needed compute resources and execution environment. With a batch job, the job is submitted to a job queue, and is run somewhere on the cluster as resources become available. In most cases, the submitter will submit multiple batch jobs with no execution time constraints and will process the job results as they become available.

Moab can enforce rich policies defining how, when, and where batch jobs run to deliver compute resources to the most important workload and provide general SLA guarantees while maximizing system utilization and minimizing average response time.

Interactive Workload

Interactive workload differs from batch in that requestors are interested in immediate response and are generally waiting for the interactive request to be executed before going on to other activities. In many cases, interactive submitters will continue to be *attached* to the interactive job, routing keystrokes and other input into the job and seeing both output and error information in real-time. While interactive workload may be submitted within a job file, commonly, it is routed into the cluster via a web or other graphical terminal and the end-user may never even be aware of the underlying use of the batch system.

For managing interactive jobs, the focus is usually on setting aside resources to guarantee immediate execution or at least a minimal wait time for interactive jobs. Targeted service levels require management when mixing batch and interactive jobs. Interactive and other jobs types can be dynamically steered in terms of what they are executing as well as in terms of the quantity of resources required by the application. Moab can apply dynamic or malleable job facilities to dynamically grow and shrink jobs as needed to meet these changing constraints.

Calendar Workload

Calendar workload must be executed at a particular time and possibly in a regular periodic manner. For such jobs, time constraints range from flexible to rigid. For example, some calendar jobs may need to complete by a certain time, while others must run exactly at a given time each day or each week.

Moab can schedule the future and can thus guarantee resource availability at needed times to allow calendar jobs to run as required. Furthermore, Moab provisioning features can locate or temporarily create the needed compute environment to properly execute the target applications.

Service Workload

Moab can schedule and manage both individual applications and long-running or persistent services. Service workload processes externally-generated transaction requests while Moab provides the distributed service with needed resources to meet target backlog or response targets to the service. Examples of service workload include parallel databases, web farms, and visualization services. Moab can apply cluster, grid, or dynamically-generated on-demand resources to the service.

When handling service workload, Moab observes the application in a highly abstract manner. Using the JOBCFG parameter, aspects of the service jobs can be discovered or configured with attributes describing them as resource consumers possessing response time, backlog, state metrics, and associated QoS targets. In addition, each application can specify the type of compute resource required (OS, arch, memory, disk, network adapter, data store, and so forth) as well as the support environment (network, storage, external services, and so forth).

If the QoS response time/backlog targets of the application are not being satisfied by the current resource allocation, Moab evaluates the needs of this application against all other site mission objectives and workload needs and determines what it must do to locate or create (that is, provision, customize, secure) the needed resources. With the application resource requirement specification, a site may also indicate proximity/locality constraints, partition policies, ramp-up/ramp-down rules, and so forth.

Once Moab identifies and creates appropriate resources, it hands these resources to the application via a site customized URL. This URL can be responsible for whatever application-specific hand-shaking must be done to launch and initialize the needed components of the distributed application upon the new resources. Moab engages in the hand-off by providing needed context and resource information and by launching the URL at the appropriate time.

Related topics

- Malleable Jobs
- QOS/SLA Enforcement

Scheduler Basics

Initial Moab Configuration

Configuring an RPM-based install of Moab

When Moab is installed via an RPM source, the `moab.cfg` file contains only one directive - an `#IMPORT` line that imports all the configuration files in `/opt/moab/etc`. The usual configuration settings that are normally contained in `moab.cfg` have been moved to `moab-server.cfg`. Moab still reads the moab.cfg file and, due to the `#INCLUDE` directive, reads in all the other configuration files as well.

To configure Moab in the case of an RPM install, you can modify the `moab.cfg` file, the `moab-server.cfg` file, or any of the configuration files that are read in by `moab.cfg` such as the accounting manager configuration file (`am.cfg`) or the resource manager configuration file (`rm.cfg`).

The RPMs allow for a client install of Moab, instead of a server install. In this instance, the `moab-server.cfg` file is replaced with a `moab-client.cfg` file. The server and client RPMs cannot be installed on the same machine.

Basic configuration of Moab

After Moab is installed, there may be minor configuration remaining within the primary configuration file, `moab.cfg`. While the `configure` script automatically sets these parameters, sites may choose to specify additional parameters. If the values selected in `configure` are satisfactory, then this section may be safely ignored.

The parameters needed for proper initial startup include the following:

Parameter	Instructions
SCHEDCFG	The **SCHEDCFG** parameter specifies how the Moab server will execute and communicate with client requests. The **SERVER** attribute allows Moab client commands to locate the Moab server and is specified as a URL or in *<HOST>[:<PORT>]* format. For example: `SCHEDCFG[orion] SERVER=cw.psu.edu` Specifying the server in the Moab configuration file is optional. If nothing is specified, `gethostname()` is called. You can restart Moab and run mdiag -S to confirm that the correct host name is specified. The **SERVER** attribute can also be set using the environment variable **$MOABSERVER**. Using this variable allows you to quickly change to the Moab server that client commands will connect to. `> export MOABSERVER=cluster2:12221`
ADMINCFG	Moab provides role-based security enabled via multiple levels of admin access. Users who are to be granted full control of all Moab functions should be indicated by setting the **ADMINCFG[1]** parameter. The first user in this **USERS** attribute list is considered the *primary* administrator. It is the ID under which Moab will execute. For example, the following may be used to enable users *greg* and *thomas* as level 1 admins: `ADMINCFG[1] USERS=greg,thomas` Moab may only be launched by the primary administrator user ID. The primary administrator should be configured as a manager/operator/administrator in every resource manager with which Moab will interface. If the msub command will be used, then "root" *must* be the primary administrator. Moab's home directory and contents should be owned by the primary administrator.
RMCFG	For Moab to properly interact with a resource manager, the interface to this resource manager must be defined as described in the Resource Manager Configuration Overview. Further, it is important that the primary Moab administrator also be a resource manager administrator within each of those systems. For example, to interface to a TORQUE resource manager, the following may be used: `RMCFG[torque1] TYPE=pbs`

Related topics

- Parameter Overview
- mdiag -C command (for diagnosing current Moab configuration)

Layout of Scheduler Components

Moab is initially unpacked into a simple one-deep directory structure. What follows demonstrates the default layout of scheduler components; some of the files (such as log and statistics files) are created while Moab runs.

- * $(MOABHOMEDIR) (default is /opt/moab and can be modified via the --with-homedir parameter during ./configure) contains the following files:

Filename	Description
.moab.ck	Checkpoint file
.moab.pid	Lock file
moab.lic	License file
contrib/	Directory containing contributed code and plug-ins
docs/	Directory for documentation
etc/	Directory for configuration files
moab.cfg	General configuration file
moab.dat	Configuration file generated by Moab Cluster Manager
moab-private.cfg	Secure configuration file containing private information
lib/	Directory for library files (primarily for tools/)
log/	Directory for log files
moab.log	Log file
moab.log.1	Previous log file

Filename	Description
stats/	Directory for statistics files: ◦ events.<date> – event files ◦ {DAY|WEEK|MONTH|YEAR}.<date> – usage profiling data ◦ FS.<PARTITION>.<epochtime> – fairshare usage data
samples/	Directory for sample configuration files, simulation trace files, etc.

- `$(MOABINSTDIR)` (default is `/opt/moab` and can be modified via the `--prefix` parameter during `./configure`) contains the following files:

Filename	Description
bin/	Directory for client commands (for example, showq, setres, etc.)
sbin/	Directory for server daemons
moab	Moab binary
tools/	Directory for resource manager interfaces and local scripts

- `/etc/moab.cfg` – If the Moab home directory cannot be found at startup, this file is checked to see if it declares the Moab home directory. If a declaration exists, the system checks the declared directory to find Moab. The syntax is: MOABHOMEDIR=*<DIRECTORY>*.

If you want to run Moab from a different directory other than `/opt/moab` but did not use the `--with-homedir` parameter during `./configure`, you can set the *$MOABHOMEDIR* environment variable, declare the home directory in the `/etc/moab.cfg` file, or use the `-C` command line option when using the Moab server or client commands to specify the configuration file location.

When Moab runs, it creates a log file, `moab.log`, in the `log/` directory and creates a statistics file in the `stats/` directory with the naming convention `events.WWW_MMM_DD_YYYY` (for example, `events.Sat_Oct_10_2009`). Additionally, a checkpoint file, `.moab.ck`, and lock file, `.moab.pid`, are maintained in the Moab home directory.

Layout of Scheduler Components with Integrated Database Enabled

If USEDATABASE INTERNAL is configured, the layout of scheduler components varies slightly. The `.moab.ck` file and usage profiling data (`stat/{DAY|WEEK|MONTH|YEAR}.<date>`) are stored in the `moab.db` database. In addition, the event information is stored in both event files: (`stat/events.<date>`) and `moab.db`.

Related topics

- Commands Overview
- Installation

Scheduling Environment

Moab functions by manipulating a number of elementary objects, including jobs, nodes, reservations, QoS structures, resource managers, and policies. Multiple minor elementary objects and composite objects are also used; these objects are defined in the scheduling dictionary.

- Jobs
  - Job States
  - Requirement (or Req)
- Nodes
- Advance Reservations
- Policies
- Resources
- Task
- PE
- Class (or Queue)
- Resource Manager (RM)

Moab functions by manipulating a number of elementary objects, including jobs, nodes, reservations, QoS structures, resource managers, and policies. Multiple minor elementary objects and composite objects are also used; these objects are defined in the scheduling dictionary.

Jobs

Job information is provided to the Moab scheduler from a resource manager such as Loadleveler, PBS, Wiki, or LSF. Job attributes include ownership of the job, job state, amount and type of resources required by the job, and a wallclock limit indicating how long the resources are required. A job consists of one or more task groups, each of which requests a number of resources of a given type; for example, a job may consist of two task groups, the first asking for a single master task consisting of *1 IBM SP node with at least 512 MB of RAM* and the second asking for a set of slave tasks such as *24 IBM SP nodes with at least 128 MB of RAM*. Each task group consists of one or more tasks where a task is defined as the minimal independent unit of resources. By default, each task is equivalent to one processor. In SMP environments, however, users may wish to tie one or more processors together with a certain amount of memory and other resources.

Job States

The job's *state* indicates its current status and eligibility for execution and can be any of the values listed in the following tables:

Table 3-1: Pre-execution states

State	Definition
Deferred	Job that has been held by Moab due to an inability to schedule the job under current conditions. Deferred jobs are held for DEFERTIME before being placed in the idle queue. This process is repeated DEFERCOUNT times before the job is placed in batch hold.
Hold	Job is idle and is not eligible to run due to a user, (system) administrator, or batch system *hold* (also, batchhold, systemhold, userhold).
Idle	Job is currently queued and eligible to run but is not executing (also, **notqueued**).
NotQueued	The job has not been queued.
Unknown	Moab cannot determine the state of the job.

Table 3-2: Execution states

State	Definition
Starting	Batch system has attempted to start the job and the job is currently performing *pre-start* tasks that may include provisioning resources, staging data, or executing system pre-launch scripts.
Running	Job is currently executing the user application.
Suspended	Job was running but has been suspended by the scheduler or an administrator; user application is still in place on the allocated compute resources, but it is not executing.

Table 3-3: Post-execution states

State	Definition
Completed	Job has completed running without failure.
Removed	Job has run to its requested walltime successfully but has been canceled by the scheduler or resource manager due to exceeding its walltime or violating another policy; includes jobs canceled by users or administrators either before or after a job has started.

State	Definition
Vacated	Job canceled after partial execution due to a system failure.

Task Group (or Req)

A job *task group* (or req) consists of a request for a single type of resources. Each task group consists of the following components:

Component	Description
Task Definition	A specification of the elementary resources that compose an individual task.
Resource Constraints	A specification of conditions that must be met for resource matching to occur. Only resources from nodes that meet *all* resource constraints may be allocated to the job task group.
Task Count	The number of task instances required by the task group.
Task List	The list of nodes on which the task instances are located.
Task Group Statistics	Statistics tracking resource utilization.

Nodes

Moab recognizes a node as a collection of resources with a particular set of associated attributes. This definition is similar to the traditional notion of a node found in a Linux cluster or supercomputer wherein a node is defined as one or more CPUs, associated memory, and possibly other compute resources such as local disk, swap, network adapters, and software licenses. Additionally, this node is described by various attributes such as an architecture type or operating system. Nodes range in size from small uniprocessor PCs to large symmetric multiprocessing (SMP) systems where a single node may consist of hundreds of CPUs and massive amounts of memory.

In many cluster environments, the primary source of information about the configuration and status of a compute node is the resource manager. This information can be augmented by additional information sources including node monitors and information services. Further, extensive node policy and node configuration information can be specified within Moab via the graphical tools or the configuration file. Moab aggregates this information and presents a comprehensive view of the node configuration, usages, and state.

While a node in Moab in most cases represents a standard compute host, nodes may also be used to represent more generalized resources. The GLOBAL node possesses floating resources that are available cluster wide, and created virtual nodes (such as network, software, and data nodes) track and allocate resource usage for other resource types.

For additional node information, see General Node Administration.

Advance Reservations

An advance reservation dedicates a block of specific resources for a particular use. Each reservation consists of a list of resources, an access control list, and a time range for enforcing the access control list. The reservation ensures the matching nodes are used according to the access controls and policy constraints within the time frame specified. For example, a reservation could reserve 20 processors and 10 GB of memory for users Bob and John from Friday 6:00 a.m. to Saturday 10:00 p.m. Moab uses advance reservations extensively to manage backfill, guarantee resource availability for active jobs, allow service guarantees, support deadlines, and enable metascheduling. Moab also supports both regularly recurring reservations and the creation of dynamic one-time reservations for special needs. Advance reservations are described in detail in the Advance Reservations overview.

Policies

A configuration file specifies policies controls how and when jobs start. Policies include job prioritization, fairness policies, fairshare configuration policies, and scheduling policies.

Resources

Jobs, nodes, and reservations all deal with the abstract concept of a resource. A resource in the Moab world is one of the following:

Resource	Description
processors	Specify with a simple count value
memory	Specify real memory or RAM in megabytes (MB)
swap	Specify virtual memory or *swap* in megabytes (MB)
disk	Specify local disk in megabytes (MB)

In addition to these elementary resource types, there are two higher level resource concepts used within Moab: **Task** and the processor equivalent, or (**PE**).

Task

A task is a collection of elementary resources that must be allocated together within a single node. For example, a task may consist of one processor, 512 MB of RAM, and 2 GB of local disk. A key aspect of a task is that the resources associated with the task must be allocated as an atomic unit, without spanning node boundaries. A task requesting 2 processors cannot be satisfied by allocating 2 uniprocessor nodes, nor can a task requesting 1 processor and 1 GB of memory be satisfied by allocating 1 processor on 1 node and memory on another.

In Moab, when jobs or reservations request resources, they do so in terms of tasks typically using a task count and a task definition. By default, a task maps directly to a single processor within a job and maps to a full node within reservations. In all cases, this default definition can be overridden by specifying a new task definition.

Within both jobs and reservations, depending on task definition, it is possible to have multiple tasks from the same job mapped to the same node. For example, a job requesting 4 tasks using the default task definition of 1 processor per task, can be satisfied by 2 dual processor nodes.

PE

The concept of the processor equivalent, or PE, arose out of the need to translate multi-resource consumption requests into a scalar value. It is not an elementary resource but rather a derived resource metric. It is a measure of the actual impact of a set of requested resources by a job on the total resources available system wide. It is calculated as follows:

```
 PE = MAX(ProcsRequestedByJob / TotalConfiguredProcs,
MemoryRequestedByJob / TotalConfiguredMemory,
DiskRequestedByJob / TotalConfiguredDisk,
SwapRequestedByJob / TotalConfiguredSwap) * TotalConfiguredProcs
```

For example, if a job requested 20% of the total processors and 50% of the total memory of a 128-processor MPP system, only two such jobs could be supported by this system. The job is essentially using 50% of all available resources since the system can only be scheduled to its most constrained resource - memory in this case. The processor equivalents for this job should be 50% of the processors, or PE = 64.

Another example: Assume a homogeneous 100-node system with 4 processors and 1 GB of memory per node. A job is submitted requesting 2 processors and 768 MB of memory. The PE for this job would be calculated as follows:

```
 PE = MAX(2/(100*4), 768/(100*1024)) * (100*4) = 3.
```

This result makes sense since the job would be consuming 3/4 of the memory on a 4-processor node.

The calculation works equally well on homogeneous or heterogeneous systems, uniprocessor or large SMP systems.

Class (or Queue)

A class (or queue) is a logical container object that implicitly or explicitly applies policies to jobs. In most cases, a class is defined and configured within the resource manager and associated with one or more of the following attributes or constraints:

Attribute	Description
Default Job Attributes	A queue may be associated with a default job duration, default size, or default resource requirements.

Attribute	Description
Host Constraints	A queue may constrain job execution to a particular set of hosts.
Job Constraints	A queue may constrain the attributes of jobs that may be submitted, including setting limits such as max wallclock time and minimum number of processors.
Access List	A queue may constrain who may submit jobs into it based on such things as user lists and group lists.
Special Access	A queue may associate special privileges with jobs including adjusted job priority.

As stated previously, most resource managers allow full class configuration within the resource manager. Where additional class configuration is required, the CLASSCFG parameter may be used.

Moab tracks class usage as a consumable resource allowing sites to limit the number of jobs using a particular class. This is done by monitoring class initiators that may be considered to be a ticket to run in a particular class. Any compute node may simultaneously support several types of classes and any number of initiators of each type. By default, nodes will have a one-to-one mapping between class initiators and configured processors. For every job task run on the node, one class initiator of the appropriate type is consumed. For example, a 3-processor job submitted to the class "batch" consumes three batch class initiators on the nodes where it runs.

Using queues as consumable resources allows sites to specify various policies by adjusting the class initiator to node mapping. For example, a site running serial jobs may want to allow a particular 8-processor node to run any combination of batch and special jobs subject to the following constraints:

- Only 8 jobs of any type allowed simultaneously.
- No more than 4 special jobs allowed simultaneously.

To enable this policy, the site may set the node's MAXJOB policy to 8 and configure the node with 4 special class initiators and 8 batch class initiators.

In virtually all cases, jobs have a one-to-one correspondence between processors requested and class initiators required. However, this is not a requirement, and with special configuration, sites may choose to associate job tasks with arbitrary combinations of class initiator requirements.

In displaying class initiator status, Moab signifies the type and number of class initiators available using the format [<CLASSNAME>:<CLASSCOUNT>]. This is most commonly seen in the output of node status commands indicating the number of configured and available class initiators, or in job status commands when displaying class initiator requirements.

Resource Manager (RM)

While other systems may have more strict interpretations of a resource manager and its responsibilities, Moab's multi-resource manager support allows a much more liberal interpretation. In essence, any object that can provide environmental information and environmental control can be used as a resource manager, including sources of resource, workload, credential, or policy information such as scripts, peer services, databases, web services, hardware monitors, or even flats files. Likewise, Moab considers to be a resource manager any tool that provides control over the cluster environment whether that be a license manager, queue manager, checkpoint facility, provisioning manager, network manager, or storage manager.

Moab aggregates information from multiple unrelated sources into a larger more complete world view of the cluster that includes all the information and control found within a standard resource manager such as TORQUE, including node, job, and queue management services. For more information, see the Resource Managers and Interfaces overview.

Arbitrary Resource

Nodes can also be configured to support various arbitrary resources. Use the NODECFG parameter to specify information about such resources. For example, you could configure a node to have *256 MB RAM, 4 processors, 1 GB Swap, and 2 tape drives*.

Scheduling Dictionary

Account	
Definition	A credential also known as "project ID." Multiple users may be associated a single account ID and each user may have access to multiple accounts. (See credential definition and ACCOUNTCFG parameter.)
Example	`ACCOUNT=hgc13`

ACL (Access Control List)	
Definition	In the context of scheduling, an access control list is used and applied much as it is elsewhere. An ACL defines what credentials are required to access or use particular objects. The principal objects to which ACLs are applied are reservations and QoSs. ACLs may contain both allow and deny statements, include wildcards, and contain rules based on multiple object types.
Example	Reservation META1 contains 4 access statements. • Allow jobs owned by user "john" or "bob " • Allow jobs with QoS "premium" • Deny jobs in class "debug" • Allow jobs with a duration of less than 1 hour

Allocation	
Definition	A logical, scalar unit assigned to users on a credential basis, providing access to a particular quantity of compute resources. Allocations are consumed by jobs associated with those credentials.
Example	`ALLOCATION=30000`

Class	
Definition	(See Queue) A class is a logical container object that holds jobs allowing a site to associate various constraints and defaults to these jobs. Class access can also be tied to individual nodes defining whether a particular node will accept a job associated with a given class. Class based access to a node is denied unless explicitly allowed via resource manager configuration. Within Moab, classes are tied to jobs as a credential.
Example	job "cw.073" is submitted to class batch node "cl02" accepts jobs in class batch reservation weekend allows access to jobs in class batch

CPU	
Definition	A single processing unit. A CPU is a consumable resource. Nodes typically consist of one or more CPUs. (same as processor)

Credential	
Definition	An attribute associated with jobs and other objects that determines object identity. In the case of schedulers and resource managers, credential based policies and limits are often established. At submit time, jobs are associated with a number of credentials such as user, group , account , QoS, and class. These job credentials subject the job to various polices and grant it various types of access. In most cases, credentials set both the privileges of the job and the ID of the actual job executable.
Example	Job "cw.24001" possesses the following credentials: `USER=john;GROUP=staff;ACCOUNT=[NONE];` `QOS=[DEFAULT];CLASS=batch`

Disk	
Definition	A quantity of local disk available for use by batch jobs. Disk is a consumable resource.

Execution Environment

Definition	A description of the environment in which the executable is launched. This environment may include attributes such as the following: • an executable • command line arguments • input file • output file • local user ID • local group ID • process resource limits
Example	Job "cw.24001" possesses the following execution environment: `EXEC=/bin/sleep;ARGS="60";` `INPUT=[NONE];OUTPUT=[NONE];` `USER=loadl;GROUP=staff;`

Fairshare

Definition	A mechanism that allows historical resource utilization information to be incorporated into job priority decisions.

Fairness

Definition	The access to shared compute resources that each user is granted. Access can be equal or based on factors such as historical resource usage, political issues, and job value.

Group

Definition	A credential typically directly mapping to a user's UNIX group ID.

Job

Definition	The fundamental object of resource consumption. A job contains the following components: • A list of required consumable resources • A list of resource constraints controlling which resources may be allocated to the job • A list of job constraints controlling where, when, and how the job should run • A list of credentials • An execution environment

Job Constraints

Definition	A set of conditions that must be fulfilled for the job to start. These conditions are far reaching and may include one or more of the following: • When the job may run. (After time X, within Y minutes.) • Which resources may be allocated. (For example, node must possess at least 512 MB of RAM, run only in partition or Partition C, or run on HostA and HostB.) • Starting job relative to a particular event. (Start after job X successfully completes.)
Example	`RELEASETIME>='Tue Feb 12, 11:00AM'` `DEPEND=AFTERANY:cw.2004` `NODEMEMORY==256MB`

Memory

Definition	A quantity of physical memory (RAM). Memory is provided by compute nodes. It is required as a constraint or consumed as a consumable resource by jobs. Within Moab, memory is tracked and reported in megabytes (MB).
Example	Node "node001" provides the following resources: `PROCS=1,MEMORY=512,SWAP=1024` "Job cw.24004" consumes the following resources per task: `PROCS=1,MEMORY=256`

Node

Definition	A node is the fundamental object associated with compute resources. Each node contains the following components: • A list of consumable resources • A list of node attributes

Node Attribute

Definition	A node attribute is a non-quantitative aspect of a node. Attributes typically describe the node itself or possibly aspects of various node resources such as processors or memory. While it is probably not optimal to aggregate node and resource attributes together in this manner, it is common practice. Common node attributes include processor architecture, operating system, and processor speed. Jobs often specify that resources be allocated from nodes possessing certain node attributes.
Example	`ARCH=AMD,OS=LINUX24,PROCSPEED=950`

Node Feature

Definition	A node feature is a node attribute that is typically specified locally via a configuration file. Node features are opaque strings associated with the node by the resource manager that generally only have meaning to the end-user, or possibly to the scheduler. A node feature is commonly associated with a subset of nodes allowing end-users to request use of this subset by requiring that resources be allocated from nodes with this feature present. In many cases, node features are used to extend the information provided by the resource manager.
Example	`FEATURE=s950,pIII,geology` *This may be used to indicate that the node possesses a 950 MHz Pentium III processor and that the node is owned by the Geology department.*

Processor

Definition	A processing unit. A processor is a consumable resource. Nodes typically consist of one or more processors. (same as CPU)

Quality of Service (QoS)

Definition	An object that provides special services, resources, and so forth.

Queue

Definition	(see Class)

Reservation

Definition	An object that reserves a specific collection or resources for a specific timeframe for use by jobs that meet specific conditions.
Example	Reserve 24 processors and 8 GB of memory from time T1 to time T2 for use by user X or jobs in the class batch.

Resource

Definition	Hardware, generic resources such as software, and features available on a node, including memory, disk, swap, and processors.

Resource, Available

Definition	A compute node's configured resources minus the *maximum* of the sum of the resources utilized by all job tasks running on the node and the resources dedicated; that is, R.Available = R.Configure - MAX(R.Dedicated,R.Utilized). In most cases, resources utilized will be associated with compute jobs that the batch system has started on the compute nodes, although resource consumption may also come from the operating system or *rogue* processes outside of the batch system's knowledge or control. Further, in a well-managed system, utilized resources are less than or equal to dedicated resources and when exceptions are detected, one or more usage-based limits are activated to preempt the jobs violating their requested resource usage.
Example	Node "cl003" has 4 processors and 512 MB of memory. It is executing 2 tasks of job "clserver.0041" that are using 1 processor and 60 MB of memory each. One processor and 250 MB of memory are reserved for user "jsmith" but are not currently in use. Resources available to user jsmith on node "cl003": • 2 processors • 392 MB memory Resources available to a user other than jsmith on node "cl003": • 1 processor • 142 MB memory

Resource, Configured

Definition	The total amount of consumable resources that are available on a compute node for use by job tasks.
Example	Node "cl003" has 4 processors and 512 MB of memory. It is executing 2 tasks of job "clserver.0041" that are using 1 processor and 60 MB of memory each. One processor and 250 MB of memory are reserved for user "jsmith" but are not currently in use. Configured resources for node "cl003": • 4 processors • 512 MB memory

Resource, Consumable

Definition	Any object that can be used (that is, consumed and thus made unavailable to another job) by, or dedicated to a job is considered to be a resource. Common examples of resources are a node's physical memory or local disk. As these resources may be given to one job and thus become unavailable to another, they are considered to be consumable. Other aspects of a node, such as its operating system, are not considered to be consumable since its use by one job does not preclude its use by another. Note that some node objects, such as a network adapter, may be dedicated under some operating systems and resource managers and not under others. On systems where the network adapter cannot be dedicated and the network usage per job cannot be specified or tracked, network adapters are not considered to be resources, but rather attributes. Nodes possess a specific quantity of consumable resources such as real memory, local disk, or processors. In a resource management system, the node manager may choose to report only those configured resources available to batch jobs. For example, a node may possess an 80-GB hard drive but may have only 20 GB dedicated to batch jobs. Consequently, the resource manager may report that the node has 20 GB of local disk available when idle. Jobs may explicitly request a certain quantity of consumable resources.

Resource, Constraint

Definition	A resource constraint imposes a rule on which resources can be used to match a resource request. Resource constraints either specify a required quantity and type of resource or a required node attribute. All resource constraints must be met by any given node to establish a match.

Resource, Dedicated

Definition	A job may request that a block of resources be dedicated while the job is executing. At other times, a certain number of resources may be reserved for use by a particular user or group. In these cases, the scheduler is responsible for guaranteeing that these resources, utilized or not, are set aside and made unavailable to other jobs.
Example	Node " cl003" has 4 processors and 512 MB of memory. It is executing 2 tasks of job "clserver.0041" that are using 1 processor and 60 MB of memory each. One processor and 250 MB of memory are reserved for user "jsmith" but are not currently in use. Dedicated resources for node "cl003": • 1 processor • 250 MB memory

Resource, Utilized

Definition	All consumable resources actually used by all job tasks running on the compute node.

Resource, Utilized	
Example	Node "cl003" has 4 processors and 512 MB of memory. It is executing 2 tasks of job "clserver.0041" that are using 1 processor and 60 MB of memory each. One processor and 250 MB of memory are reserved for user "jsmith" but are not currently in use. Utilized resources for node "cl003": • 2 processors • 120 MB memory

Swap	
Definition	A quantity of virtual memory available for use by batch jobs. Swap is a consumable resource provided by nodes and consumed by jobs.

Task	
Definition	An atomic collection of consumable resources.

User, Global	
Definition	The user credential used to provide access to functions and resources. In local scheduling, global user IDs map directly to local user IDs.

User, Local	
Definition	The user credential under which the job executable will be launched.

Workload	
Definition	Generalized term.

Scheduling Iterations and Job Flow

- Scheduling Iterations
    - Update State Information
    - Handle User Requests

Scheduling Iterations

In any given scheduling iteration, many activities take place, examples of which are listed below:

- Refresh reservations
- Schedule reserved jobs
- Schedule priority jobs
- Backfill jobs
- Update statistics
- Update State Information
- Handle User Requests
- Perform Next Scheduling Cycle

Update State Information

Each iteration, the scheduler contacts the resource manager(s) and requests up-to-date information on compute resources, workload, and policy configuration. On most systems, these calls are to a centralized resource manager daemon that possesses all information. Jobs may be reported as being in any of the following states listed in the job state table.

Handle User Requests

User requests include any call requesting state information, configuration changes, or job or resource manipulation commands. These requests may come in the form of user client calls, peer daemon calls, or process signals.

Perform Next Scheduling Cycle

Moab operates on a polling/event driven basis. When all scheduling activities complete, Moab processes user requests until a new resource manager event is received or an internal event is generated. Resource manager events include activities such as a new job submission or completion of an active job, addition of new node resources, or changes in resource manager policies. Internal events include administrator schedule requests, reservation activation/deactivation, or the expiration of the RMPOLLINTERVAL timer.

Detailed Job Flow

Determine Basic Job Feasibility

The first step in scheduling is determining which jobs are feasible. This step eliminates jobs that have job holds in place, invalid job states (such as Completed, Not Queued, Deferred), or unsatisfied preconditions. Preconditions may include stage-in files or completion of preliminary job steps.

Prioritize Jobs

With a list of feasible jobs created, the next step involves determining the relative priority of all jobs within that list. A priority for each job is calculated based on job attributes such as job owner, job size, and length of time the job has been queued.

Enforce Configured Throttling Policies

Any configured throttling policies are then applied constraining how many jobs, nodes, processors, and so forth are allowed on a per credential basis. Jobs that violate these policies are not considered for scheduling.

Determine Resource Availability

For each job, Moab attempts to locate the required compute resources needed by the job. For a match to be made, the node must possess all node attributes specified by the job and possess adequate available resources to meet the "TasksPerNode" job constraint. (Default "TasksPerNode" is 1.) Normally, Moab determines that a node has adequate resources if the resources are *neither utilized by nor dedicated to* another job using the calculation.

R.Available = R.Configured - MAX(R.Dedicated,R.Utilized).

The **NODEAVAILABILITYPOLICY** on page 979 parameter can be modified to adjust this behavior.

Allocate Resources to Job

If adequate resources can be found for a job, the node allocation policy is then applied to select the best set of resources. These allocation policies allow selection criteria such as speed of node, type of reservations, or excess node resources to be figured into the allocation decision to improve the performance of the job and maximize the freedom of the scheduler in making future scheduling decisions.

Launch Job

With the resources selected and task distribution mapped, the scheduler then contacts the resource manager and informs it where and how to launch the job. The resource manager then initiates the actual job executable.

Configuring the Scheduler

- Adjusting Server Behavior
  - Logging
  - Checkpointing
  - Client Interface
  - Scheduler Mode
  - **Configuring a job ID offset** on page 124

Scheduler configuration is maintained using the flat text configuration file `moab.cfg`. All configuration file entries consist of simple **<PARAMETER>** *<VALUE>* pairs that are whitespace delimited. Parameter names are not case sensitive but *<VALUE>* settings are. Some parameters are array values and should be specified as **<PARAMETER>***[<INDEX>]* (Example: **QOSCFG***[hiprio]* **PRIORITY**=*1000*); the *<VALUE>* settings may be integers, floats, strings, or arrays of these. Some parameters can be specified as arrays wherein index values can be numeric or alphanumeric strings. If no array index is specified for an array parameter, an index of zero (0) is assumed. The example below includes both array based and non-array based parameters:

```
SCHEDCFG[cluster2] SERVER=head.c2.org MODE=NORMAL
LOGLEVEL 6
LOGDIR   /var/tmp/moablog
```

See the parameters documentation for information on specific parameters.

The `moab.cfg` file is read when Moab is started up or recycled. Also, the mschedctl -m command can be used to reconfigure the scheduler at any time, updating some or all of the configurable parameters dynamically. This command can be used to modify parameters either permanently or temporarily. For example, the command mschedctl -m LOGLEVEL 3will temporarily adjust the scheduler log level. When the scheduler restarts, the log level restores to the value stored in the Moab configuration files. To adjust a parameter permanently, the option `--flags=persistent` should be set.

At any time, the current server parameter settings may be viewed using the mschedctl -l command.

Adjusting Server Behavior

Most aspects of Moab behavior are configurable. This includes both scheduling policy behavior and daemon behavior. In terms of configuring server behavior, the following realms are most commonly modified.

Logging

Moab provides extensive and highly configurable logging facilities controlled by parameters.

Parameter	Description
LOGDIR	Indicates directory for log files.

Parameter	Description
LOGFACILITY	Indicates scheduling facilities to track.
LOGFILE	Indicates path name of log file.
LOGFILEMAXSIZE	Indicates maximum size of log file before rolling.
LOGFILEROLLDEPTH	Indicates maximum number of log files to maintain.
LOGLEVEL	Indicates verbosity of logging.

Checkpointing

Moab checkpoints its internal state. The checkpoint file records statistics and attributes for jobs, nodes, reservations, users, groups, classes, and almost every other scheduling object.

Parameter	Description
CHECKPOINTEXPIRATIONTIME	Indicates how long unmodified data should be kept after the associated object has disappeared; that is, job priority for a job no longer detected.
CHECKPOINTFILE	Indicates path name of checkpoint file.
CHECKPOINTINTERVAL	Indicates interval between subsequent checkpoints.

Client Interface

The Client interface is configured using the SCHEDCFG parameter. Most commonly, the attributes **SERVER** and **PORT** must be set to point client commands to the appropriate Moab server. Other parameters such as CLIENTTIMEOUT may also be set.

Scheduler Mode

The scheduler mode of operation is controlled by setting the **MODE** attribute of the SCHEDCFG parameter. The following modes are allowed:

Mode	Description
INTERACTIVE	Moab interactively confirms each scheduling action before taking any steps. (See interactive mode overview for more information.)

Mode	Description
MONITOR	Moab observes cluster and workload performance, collects statistics, interacts with allocation management services, and evaluates failures, but it does not actively alter the cluster, including job migration, workload scheduling, and resource provisioning. (See monitor mode overview for more information.)
NORMAL	Moab actively schedules workload according to mission objectives and policies; it creates reservations; starts, cancels, preempts, and modifies jobs; and takes other scheduling actions.
SIMULATION	Moab obtains workload and resource information from specified simulation trace files and schedules the defined virtual environment.
SINGLESTEP	Moab behaves as in **NORMAL** mode but will only schedule a single iteration and then exit.
SLAVE	Moab behaves as in **NORMAL** mode but will only start a job when explicitly requested by a trusted grid peer service or administrator.
TEST	Moab behaves as in **NORMAL** mode, will make reservations, and scheduling decisions, but will then only log scheduling actions it would have taken if running in **NORMAL** mode. In most cases, "TEST" mode is identical to **MONITOR** mode. (See test mode overview for more information.)

Configuring a job ID offset

Moab assigns job IDs as integers in numeric order as jobs are submitted, starting with 1. In some situations, you might want to offset the integer at which Moab starts to assign job IDs in your system.

This example describes how you would offset the job IDs in a compound system consisting of Site A, Site B, and Site C, each of which runs its own instance of Moab. Users belonging to any of the sites can submit jobs to their own site and to the other two. To simplify aggregation of usage records from the three sites, offset the job IDs for Site B to a starting value higher than the expected total lifetime value for the system; in this example, to *20000000*. Likewise, set Site C to 20,000,000 more, or *40000000*. To do so, set the **MINJOBID** attribute of **SCHEDCFG** in each system's `moab.cfg` to the offset value. To ensure that Moab will never use the same job ID for two different sites, also set **MAXJOBID**. If the Moab job naming process ever reaches the **MAXJOBID**, it will start over again with the **MINJOBID**.

```
SCHEDCFG[moab] SERVER=moab_siteA:4244 MAXJOBID=19999999
```

```
SCHEDCFG[moab] SERVER=moab_siteB:4344 MINJOBID=20000000 MAXJOBID=39999999
```

```
SCHEDCFG[moab] SERVER=moab_siteC:4444 MINJOBID=40000000 MAXJOBID=59999999
```

When users submit jobs to Moab using **msub** on page 290, Moab selects the job ID in numeric order, starting with 1 in Site A, 20000000 in Site B, and 40000000 in Site C.

If the compound system in this example uses TORQUE as its resource manager and users submit jobs directly to TORQUE using qsub, TORQUE assigns the job ID instead of Moab. In this case, you should also offset the TORQUE job IDs by setting the **next_job_number** on page 2430 server parameter of Site B and Site C to *20000000* and *40000000*, respectively.

```
$user qmgr "set server next_job_number=20000000"
```

```
$user qmgr "set server next_job_number=40000000"
```

TORQUE job ID limits will allow you to use the 20,000,000 offset scheme for up to 4 sites.

Related topics

- Initial Configuration
- Adding #INCLUDE files to moab.cfg

Credential Overview

Moab supports the concept of credentials, which provide a means of attributing policy and resource access to entities such as users and groups. These credentials allow specification of job ownership, tracking of resource usage, enforcement of policies, and many other features. There are five types of credentials -user, group, account, class, and QoS. While the credentials have many similarities, each plays a slightly different role.

- General Credential Attributes
- User Credential
- Group Credential
- Account (or Project) Credential
- Class (or Queue) Credential
- QoS Credential

General Credential Attributes

Internally, credentials are maintained as objects. Credentials can be created, destroyed, queried, and modified. They are associated with jobs and requests providing access and privileges. Each credential type has the following attributes:

- Priority Settings
- Usage Limits
- Service Targets
- Credential and Partition Access
- Statistics
- Credential Defaults, State and Configuration Information

All credentials represent a form of identity, and when applied to a job, express ownership. Consequently, jobs are subject to policies and limits associated with their owners.

Credential Priority Settings

Each credential may be assigned a priority using the **PRIORITY** attribute. This priority affects a job's total credential priority factor as described in the Priority Factors section. In addition, each credential may also specify priority weight offsets, which adjust priority weights that apply to associated jobs. These priority weight offsets include FSWEIGHT (See Priority-Based Fairshare for more information.), QTWEIGHT, and XFWEIGHT.

For example:

```
# set priority weights
CREDWEIGHT       1
USERWEIGHT       1
CLASSWEIGHT      1
SERVICEWEIGHT    1
XFACTORWEIGHT    10
QUEUETIMEWEIGHT 1000
# set credential priorities
USERCFG[john] PRIORITY=200
CLASSCFG[batch] PRIORITY=15
CLASSCFG[debug] PRIORITY=100
QOSCFG[bottomfeeder] QTWEIGHT=-50 XFWEIGHT=100
ACCOUNTCFG[topfeeder] PRIORITY=100
```

Credential Usage Limits

Usage limits constrain which jobs may run, which jobs may be considered for scheduling, and what quantity of resources each individual job may consume. With usage limits, policies such as MAXJOB, MAXNODE, and MAXMEM may be enforced against both idle and active jobs. Limits may be applied in any combination as shown in the example below where usage limits include 32 active processors per group and 12 active jobs for user *john*. For a job to run, it must satisfy the most limiting policies of all associated credentials. The Throttling Policy section documents credential usage limits in detail.

```
GROUPCFG[DEFAULT] MAXPROC=32 MAXNODE=100
GROUPCFG[staff]   MAXNODE=200
USERCFG[john]     MAXJOB=12
```

Service Targets

Credential service targets allow jobs to obtain special treatment to meet usage or response time based metrics. Additional information about service targets can be found in the Fairshare section.

Credential and Partition Access

Access to partitions and to other credentials may be specified on a per credential basis with credential access lists, default credentials, and credential membership lists.

Credential Access Lists

You can use the **ALIST**, **PLIST**, and **QLIST** attributes (shown in the following table) to specify the list of credentials or partitions that a given credential may access.

Credential	Attribute
Account	**ALIST** (allows credential to access specified list of accounts
Partition	**PLIST** (allows credential to access specified list of partitions)
QoS	QLIST (allows credential to access specified list of QoSes)

Example 3-1:

```
USERCFG[bob]   ALIST=jupiter,quantum
USERCFG[steve] ALIST=quantum
```

Account-based access lists are only enforced if using an allocation manager or if the ENFORCEACCOUNTACCESS parameter is set to "TRUE."

Assigning Default Credentials

Use the ***DEF** attribute (shown in the following table) to specify the default credential or partition for a particular credential.

Credential	Attribute
Account	**ADEF** (specifies default account)
Class	**CDEF** (specifies default class)
QoS	**QDEF** (specifies default QoS)

Example 3-2:

```
# user bob can access accounts a2, a3, and a6. If no account is explicitly requested,
# his job will be assigned to account a3
USERCFG[bob]   ALIST=a2,a3,a6 ADEF=a3
# user steve can access accounts a14, a7, a2, a6, and a1. If no account is explicitly
# requested, his job will be assigned to account a2
USERCFG[steve] ALIST=a14,a7,a2,a6,a1 ADEF=a2
```

Specifying Credential Membership Lists

As an alternate to specifying access lists, administrators may also specify membership lists. This allows a credential to specify who can access it rather than allowing each credential to specify which credentials it can access. Membership lists are controlled using the **MEMBERULIST**, **EXCLUDEUSERLIST** and **REQUIREDUSERLIST** attributes, shown in the following table:

Credential	Attribute
User	---
Account, Group, QoS	MEMBERULIST
Class	EXCLUDEUSERLIST and REQUIREDUSERLIST

Example 3-3:

```
# account omega3 can only be accessed by users johnh, stevek, jenp
ACCOUNTCFG[omega3] MEMBERULIST=johnh,stevek,jenp
```

Example 3-4: Controlling Partition Access on a Per User Basis

A site may specify the user `john` may access partitions `atlas`, `pluto`, and `zeus` and will default to partition `pluto`. To do this, include the following line in the configuration file:

```
USERCFG[john] PLIST=atlas,pluto,zeus
```

Example 3-5: Controlling QoS Access on a Per Group Basis

A site may also choose to allow everyone in the group `staff` to access QoS `standard` and `special` with a default QoS of `standard`. To do this, include the following line in the configuration file:

```
GROUPCFG[staff] QLIST=standard,special QDEF=standard
```

Example 3-6: Controlling Resource Access on a Per Account Basis

An organization wants to allow everyone in the account `omega3` to access nodes 20 through 24. To do this, include the following in the configuration file:

```
ACCOUNTCFG[omega3] MEMBERULIST=johnh,stevek,jenp
SRCFG[omega3]   HOSTLIST=r:20-24 ACCOUNTLIST=omega3
```

Credential Statistics

Full statistics are maintained for each credential instance. These statistics record current and historical resource usage, level of service delivered, accuracy of requests, and many other aspects of workload. Note, though, that you must explicitly enable credential statistics as they are not tracked by default. You can enable credential statistics by including the following in the configuration file:

```
USERCFG[DEFAULT]         ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]        ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]      ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]        ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]          ENABLEPROFILING=TRUE
```

Job Defaults, Credential State, and General Configuration

Credentials may apply defaults and force job configuration settings via the following parameters:

COMMENT	
Description	Associates a comment string with the target credential.
Example	`USERCFG[steve] COMMENT='works for boss, provides good service'` `CLASSCFG[i3]   COMMENT='queue for I/O intensive workload'`

HOLD	
Description	Specifies a hold should be placed on all jobs associated with the target credential. The order in which this **HOLD** attribute is evaluated depends on the following credential precedence: **USERCFG**, **GROUPCFG**, **ACCOUNTCFG**, **CLASSCFG**, **QOSCFG**, **USERCFG***[DEFAULT]*, **GROUPCFG***[DEFAULT]*, **ACCOUNTCFG***[DEFAULT]*, **CLASSCFG***[DEFAULT]*, **QOSCFG***[DEFAULT]*.
Example	`GROUPCFG[bert] HOLD=yes`

JOBFLAGS	
Description	Assigns the specified job flag to all jobs with the associated credential.
Example	`CLASSCFG[batch] JOBFLAGS=suspendable` `QOSCFG[special] JOBFLAGS=restartable`

NOSUBMIT	
Description	Specifies whether jobs belonging to this credential can submit jobs using msub.
Example	`ACCOUNTCFG[general]  NOSUBMIT=TRUE` `CLASSCFG[special]    NOSUBMIT=TRUE`

OVERRUN	
Description	Specifies the amount of time a job may exceed its wallclock limit before being terminated. (Only applies to user and class credentials.)
Example	`CLASSCFG[bigmem] OVERRUN=00:15:00`

VARIABLE	
Description	Specifies attribute-value pairs associated with the specified credential. These variables may be used in triggers and other interfaces to modify system behavior.
Example	`GROUPCFG[staff] VARIABLE='nocharge=true'`

Credentials may carry additional configuration information. They may specify that detailed statistical profiling should occur, that submitted jobs should be held, or that corresponding jobs should be marked as preemptible.

User Credential

The user credential is the fundamental credential within a workload manager; each job requires an association with exactly one user. In fact, the user credential is the only required credential in Moab; all others are optional. In most cases, the job's user credential is configured within or managed by the operating system itself, although Moab may be configured to obtain this information from an independent security and identity management service.

As the fundamental credential, the user credential has a number of unique attributes.

- Role
- Email Address
- Disable Moab User Email

Role

Moab supports role-based authorization, mapping particular roles to collections of specific users. See the Security section for more information.

Email Address

Facilities exist to allow user notification in the event of job or system failures or under other general conditions. This attribute allows these notifications to be mailed directly to the target user.

```
USERCFG[sally]    EMAILADDRESS=sally@acme.com
```

Disable Moab User Email

You can disable Moab email notifications for a specific user.

```
USERCFG[john]    NOEMAIL=TRUE
```

Group Credential

The group credential represents an aggregation of users. User-to-group mappings are often specified by the operating system or resource manager and typically map to a user's UNIX group ID. However, user-to-group mappings may also be provided by a security and identity management service, or you can specify such directly within Moab.

With many resource managers such as TORQUE, PBSPro, and LSF, the group associated with a job is either the user's active primary group as specified within the operating system or a group that is explicitly requested at job submission time. When a secondary group is requested, the user's default group and associated policies are not taken into account. Also note that a job may only run under one group. If more constraining policies are required for these systems, an alternate aggregation scheme such as the use of Account or QOS credentials is recommended.

To submit a job as a secondary group, refer to your local resource manager's job submission options. For TORQUE users, see the `group_list=g_list` option of the qsub -W command.

Account Credential

The account credential is also referred to as the project. This credential is generally associated with a group of users along the lines of a particular project for accounting and billing purposes. User-to-accounting mapping may be obtained from a resource manager or allocation manager, or you can configure it directly within Moab. Access to an account can be controlled via the **ALIST** and **ADEF** credential attributes specified via the Identity Manager or the `moab.cfg` file.

The MANAGERS attribute (applicable only to the account and class credentials) allows an administrator to assign a user the ability to manage jobs inside the credential, as if the user is the job owner.

Example 3-7: MANAGERS Attribute

```
ACCOUNTCFG[general]  MANAGERS=ops
ACCOUNTCFG[special]  MANAGERS=stevep
```

If a user is able to access more than one account, the desired account can be specified at job submission time using the resource-manager specific attribute. For example, with TORQUE this is accomplished using the -A argument to the qsub command.

Example 3-8: Enforcing Account Usage

Job-to-account mapping can be enforced using the **ALIST** attribute and the ENFORCEACCOUNTACCESS parameter.

```
USERCFG[john]    ALIST=proj1,proj3
USERCFG[steve]   ALIST=proj2,proj3,proj4
USERCFG[brad]    ALIST=proj1
USERCFG[DEFAULT] ALIST=proj2
ENFORCEACCOUNTACCESS TRUE
...
```

Class Credential

- Class Job Defaults
- Per Job Min/Max Limits
- Resource Access
- Class Membership Constraints

- Attributes Enabling Class Access to Other Credentials
- Special Class Attributes (such as Managers and Job Prologs)
- Setting Default Classes
- Creating a Remap Class
- Class Attribute Overview
- Enabling Queue Complex Functionality

The concept of the class credential is derived from the resource manager class or queue object. Classes differ from other credentials in that they more directly impact job attributes. In standard HPC usage, a user submits a job to a class and this class imposes a number of factors on the job. The attributes of a class may be specified within the resource manager or directly within Moab. Class attributes include the following:

- Job Defaults
- Per Job Min/Max Limits
- Resource Access Constraints
- Class Membership Constraints
- Attributes Enabling Class Access to Other Credentials
- Special Class Attributes

When using SLURM, Moab classes have a one-to-one relationship with SLURM partitions of the same name.

For all classes configured in Moab, a resource manager queue with the same name should be created.

When TORQUE reports a new queue to Moab a class of the same name is automatically applied to all nodes.

Class Job Defaults

Classes can be assigned to a default job template that can apply values to job attributes not explicitly specified by the submitter. Additionally, you can specify shortcut attributes from the table that follows:

Attribute	Description
DEFAULT.ATTR	Job Attribute
DEFAULT.DISK	Required Disk (in MB)

Attribute	Description
DEFAULT.EXT	Job RM Extension
DEFAULT.FEATURES	Required Node Features/Properties
DEFAULT.GRES	Required Consumable Generic Resources
DEFAULT.MEM	Required Memory/RAM (in MB)
DEFAULT.NODESET	Node Set Specification
DEFAULT.PROC	Required Processor Count
DEFAULT.TPN	Tasks Per Node
DEFAULT.WCLIMIT	Wallclock Limit

Defaults set in a class/queue of the resource manager will override the default values of the corresponding class/queue specified in Moab.

RESOURCELIMITPOLICY must be configured in order for the **CLASSCFG** limits to take effect.

Example 3-9:

```
CLASSCFG[batch] DEFAULT.DISK=200MB DEFAULT.FEATURES=prod DEFAULT.WCLIMIT=1:00:00
CLASSCFG[debug] DEFAULT.FEATURES=debug DEFAULT.WCLIMIT=00:05:00
```

Per Job Min/Max Limits

Classes can be assigned a minimum and a maximum job template that constrains resource requests. Jobs submitted to a particular queue must meet the resource request constraints of these templates. If a job submission exceeds these limits, the entire job submission fails.

Limit	Description
MAX.ARRAYSUBJOBS	Max Allowed Jobs in an Array
MAX.CPUTIME	Max Allowed Utilized CPU Time
MAX.NODE	Max Allowed Node Count

Limit	Description
MAX.PROC	Max Allowed Processor Count
MAX.PS	Max Requested Processor-Seconds
MIN.NODE	Min Allowed Node Count
MIN.PROC	Min Allowed Processor Count
MIN.PS	Min Requested Processor-Seconds
MIN.TPN	Min Tasks Per Node
MIN.WCLIMIT	Min Requested Wallclock Limit
MAX.WCLIMIT	Max Requested Wallclock Limit

The parameters listed in the preceding table are for classes and PARCFG only, not users, accounts, groups or QoSes, and they function on a per-job basis. The **MAX.*** and **MIN.*** parameters are different from the **MAXJOB**, **MAXNODE**, and **MAXMEM** parameters described earlier in Credential Usage Limits.

Resource Access

Classes may be associated with a particular set of compute resources. Consequently, jobs submitted to a given class may only use listed resources. This may be handled at the resource manager level or via the CLASSCFG HOSTLIST attribute.

Class Membership Constraints

Classes may be configured at either the resource manager or scheduler level to only allow select users and groups to access them. Jobs that do not meet these criteria are rejected. If specifying class membership/access at the resource manager level, see the respective resource manager documentation. Moab automatically detects and enforces these constraints. If specifying class membership/access at the scheduler level, use the **REQUIREDUSERLIST** or **EXCLUDEUSERLIST** attributes of the CLASSCFG parameter.

Under most resource managers, jobs must always be a member of one and only one class.

Attributes Enabling Class Access to Other Credentials

Classes may be configured to allow jobs to access other credentials such as QoSs and Accounts. This is accomplished using the QDEF, QLIST, **ADEF**, and **ALIST** attributes.

Special Class Attributes

The class object also possesses a few unique attributes including JOBPROLOG, **JOBEPILOG**, RESFAILPOLICY, and DISABLEAM attributes described in what follows:

MANAGERS

Users listed via the **MANAGERS** parameter are granted full control over all jobs submitted to or running within the specified class.

```
# allow john and steve to cancel and modify all jobs submitted to the class/queue
special
CLASSCFG[special] MANAGERS=john,steve
```

In particular, a class manager can perform the following actions on jobs within a class/queue:

- view/diagnose job (checkjob)
- cancel, requeue, suspend, resume, and checkpoint job (mjobctl)
- modify job (mjobctl)

JOBPROLOG

The **JOBPROLOG** class performs a function similar to the resource manager level job prolog feature; however, there are some key differences:

- Moab prologs execute on the head node; resource manager prologs execute on the nodes allocated to the job.
- Moab prologs execute as the primary Moab administrator, resource manager prologs execute as root.
- Moab prologs can incorporate cluster environment information into their decisions and actions. (See Valid Variables.)
- Unique Moab prologs can be specified on a per class basis.
- Job start requests are not sent to the resource manager until the Moab job prolog is successfully completed.
- Error messages generated by a Moab prolog are attached to jobs and associated objects; stderr from prolog script is attached to job.
- Moab prologs have access to Moab internal and peer services.

Valid epilog and prolog variables are:

Variable	Description
$TIME	Time that the trigger launches
$HOME	Moab home directory

Variable	Description
$USER	User name the job is running under
$JOBID	Unique job identifier
$HOSTLIST	Entire host list for job
$MASTERHOST	Master host for job

The **JOBPROLOG** class attribute allows a site to specify a unique per-class action to take before a job is allowed to start. This can be used for environmental provisioning, pre-execution resource checking, security management, and other functions. Sample uses may include enabling a VLAN, mounting a global file system, installing a new application or virtual node image, creating dynamic storage partitions, or activating job specific software services.

A prolog is considered to have failed if it returns a negative number. If a prolog fails, the associated job will not start.

If a prolog executes successfully, the associated epilog is guaranteed to start, even if the job fails for any reason. This allows the epilog to undo any changes made to the system by the prolog.

Job Prolog Examples

```
# explicitly specify prolog arguments for special epilog
CLASSCFG[special] JOBPROLOG='$TOOLSDIR/specialprolog.pl $JOBID $HOSTLIST'
# use default prolog arguments for batch prolog
CLASSCFG[batch]   JOBPROLOG=$TOOLSDIR/batchprolog.pl
```

JOBEPILOG

The Moab epilog is nearly identical to the prolog in functionality except that it runs after the job completes within the resource manager but before the scheduler releases the allocated resources for use by subsequent jobs. It is commonly used for job clean-up, file transfers, signaling peer services, and undoing other forms of resource customization.

An epilog is considered to have failed if it returns a negative number. If an epilog fails, the associated job will be annotated and a message will be sent to administrators.

RESFAILPOLICY

This policy allows specification of the action to take on a per-class basis when a failure occurs on a node allocated to an actively running job. See the Node Availability Overview for more information.

DISABLEAM

You can disable allocation management for jobs in specific classes by setting the **DISABLEAM** class attribute to *TRUE*. For all jobs outside of the specified classes, allocation enforcement will continue to be enforced.

```
# do not enforce allocations on low priority and debug jobs
CLASSCFG[lowprio]  DISABLEAM=TRUE
CLASSCFG[debug]    DISABLEAM=TRUE
```

Setting Default Classes

In many cases, end-users do not want to be concerned with specifying a job class/queue. This is often handled by defining a default class. Whenever a user does not explicitly submit a job to a particular class, a default class, if specified, is used. In resource managers such as TORQUE, this can be done at the resource manager level and its impact is transparent to the scheduler. The default class can also be enabled within the scheduler on a per resource manager or per user basis. To set a resource manager default class within Moab, use the **DEFAULTCLASS** attribute of the RMCFG parameter. For per user defaults, use the **CDEF** attribute of the USERCFG parameter.

Creating a Remap Class

If a single default class is not adequate, Moab provides more flexible options with the REMAPCLASS parameter. If this parameter is set and a job is submitted to the remap class, Moab attempts to determine the final class to which a job belongs based on the resources requested. If a remap class is specified, Moab compares the job's requested nodes, processors, memory, and node features with the class's corresponding minimum and maximum resource limits. Classes are searched in the order in which they are defined; when the first match is found, Moab assigns the job to that class.

Because Moab remaps at job submission, updates you make to job requirements after submission will not cause any class changes. Moab does not restart the process.

In order to use **REMAPCLASS**, you must specify a **DEFAULTCLASS**. For example:

```
RMCFG[internal] DEFAULTCLASS=batch
```

In the example that follows, a job requesting 4 processors and the node feature *fast* are assigned to the class *quick*.

```
# You must specify a default class in order to use remap classes
RMCFG[internal]   DEFAULTCLASS=batch

# Jobs submitted to "batch" should be remapped
REMAPCLASS        batch

# stevens only queue
CLASSCFG[stevens] REQ.FEATURES=stevens REQUIREDUSERLIST=stevens,stevens2

# Special queue for I/O nodes
CLASSCFG[io]      MAX.PROC=8 REQ.FEATURES=io

# General access queues
CLASSCFG[quick]   MIN.PROC=2 MAX.PROC=8 REQ.FEATURES=fast|short
CLASSCFG[medium]  MIN.PROC=2 MAX.PROC=8
CLASSCFG[DEFAULT] MAX.PROC=64
...
```

The following parameters can be used to remap jobs to different classes:

- **MIN.PROC**
- **MAX.PROC**
- **MIN.WCLIMIT**
- **MAX.WCLIMIT**
- **REQ.FEATURES**
- **REQ.FLAGS=INTERACTIVE**
- **REQUIREDUSERLIST**

If the parameter REMAPCLASSLIST is set, then only the listed classes are searched and they are searched in the order specified by this parameter. If none of the listed classes are valid for a particular job, that job retains its original class.

> The remap class only works with resource managers that allow dynamic modification of a job's assigned class/queue.

> If default credentials are specified on a remap class, a job submitted to that class will inherit those credentials. If the destination class has different defaults credentials, the new defaults override the original settings. If the destination class does not have default credentials, the job maintains the defaults inherited from the remap class.

Class Attribute Overview

The following table enumerates the different parameters for **CLASSCFG**.

> Setting DEFAULT.* on a class does not assign resources or features to that class. Rather, it specifies resources that jobs will inherit when they are submitted to the class without their own resource requests. To configure features, use NODECFG.

DEFAULT.ATTR	
Format	<ATTRIBUTE>[,<ATTRIBUTE>]...
Description	One or more comma-delimited generic job attributes.
Example	---

DEFAULT.DISK	
Format	<INTEGER>
Description	Default amount of requested disk space.
Example	---

DEFAULT.EXT	
Format	<STRING>
Description	Default job RM extension.
Example	---

DEFAULT.FEATURESDEFAULT.EXT	
Format	Comma-delimited list of features.
Description	Default list of requested node features (a.k.a, node properties). This only applies to compute resource reqs.
Example	---

DEFAULT.GRES	
Format	<STRING>[<COUNT>][,<STRING>[<COUNT>]]...
Description	Default list of per task required consumable generic resources.

DEFAULT.GRES	
Example	`CLASSCFG[viz] DEFAULT.GRES=viz:2`

DEFAULT.MEM	
Format	<INTEGER> (in MB)
Description	Default amount of requested memory.
Example	---

DEFAULT.NODE	
Format	<INTEGER>
Description	Default required node count.
Example	`CLASSCFG[viz] DEFAULT.NODE=5` When a user submits a job to the `viz` class without a specified node count, the job is assigned 5 nodes.

DEFAULT.NODESET	
Format	<SETTYPE>:<SETATTR>[:<SETLIST>[,<SETLIST>]...]
Description	Default node set.
Example	`CLASSCFG[amd] DEFAULT.NODESET=ONEOF:FEATURE:ATHLON,OPTERON`

DEFAULT.PROC	
Format	<INTEGER>
Description	Default number of requested processors.
Example	---

DEFAULT.TPN	
Format	<INTEGER>
Description	Default number of tasks per node.
Example	---

DEFAULT.WCLIMIT	
Format	<INTEGER>
Description	Default wallclock limit.
Example	---

EXCL.FEATURES	
Format	Comma- or pipe-delimited list of node features.
Description	Set of excluded (disallowed) features. If delimited by commas, reject job if all features are requested; if delimited by the pipe symbol (|), reject job if at least one feature is requested.
Example	`CLASSCFG[intel] EXCL.FEATURES=ATHLON,AMD`

EXCL.FLAGS	
Format	Comma-delimited list of job flags.
Description	Set of excluded (disallowed) job flags. Reject job if any listed flags are set.
Example	`CLASSCFG[batch] EXCL.FLAGS=INTERACTIVE`

EXCLUDEUSERLIST	
Format	Comma-delimited list of users.

EXCLUDEUSERLIST	
Description	List of users not permitted access to class.
Example	---

FORCENODEACCESSPOLICY	
Format	one of *SINGLETASK*, *SINGLEJOB*, *SINGLEUSER*, or *SHARED*
Description	Node access policy associated with queue. If set, this value overrides any per job settings specified by the user at the job level. (See Node Access Policy overview for more information.)
Example	`CLASSCFG[batch] FORCENODEACCESSPOLICY=SINGLEJOB`

FSCAP	
Format	<DOUBLE>[%]
Description	See fairshare policies specification.
Example	---

FSTARGET	
Format	<DOUBLE>[%]
Description	See fairshare policies specification.
Example	---

HOSTLIST	
Format	Host expression, or comma-delimited list of hosts or host ranges.
Description	List of hosts associated with a class. If specified, Moab constrains the availability of a class to only nodes listed in the class host list.

HOSTLIST	
Example	`CLASSCFG[batch] HOSTLIST=r:abs[45-113]`

JOBEPILOG	
Format	<STRING>
Description	Scheduler level job epilog to be run after job is completed by resource manager. (See special class attributes.)
Example	---

JOBFLAGS	
Format	Comma-delimited list of job flags.
Description	See the flag overview for a description of legal flag values.
Example	`CLASSCFG[batch] JOBFLAGS=restartable`

JOBPROLOG	
Format	<STRING>
Description	Scheduler level job prolog to be run before job is started by resource manager. (See special class attributes.)
Example	---

MANAGERS	
Format	<USER>[,<USER>]...
Description	Users allowed to control, cancel, preempt, and modify jobs within class/queue. (See special class attributes.)
Example	`CLASSCFG[fast] MANAGERS=root,kerry,e43`

MAXJOB	
Format	<INTEGER>
Description	Maximum number of jobs allowed in the class.
Example	---

MAXPROCPERNODE	
Format	<INTEGER>
Description	Maximum number of processors requested per node. May optionally include node names to articulate which nodes have a specific limit.
Example	`CLASSCFG[cpu] MAXPROCPERNODE=20  # When using this class, limit 20 for all nodes` `CLASSCFG[cpu] MAXPROCPERNODE[n1,n2]=20 MAXPROCPERNODE[n3]=10  # When using this class, limit 20 for n1 & n2 and limit 10 for n3` `CLASSCFG[cpu] MAXPROCPERNODE[n1,n2]=20 MAXPROCPERNODE=10 # When using this class, limit 20 for n1 & n2 and limit 10 for all other nodes`

MAX.CPUTIME	
Format	<INTEGER>
Description	Maximum allowed utilized CPU time.
Example	---

MAX.NODE	
Format	<INTEGER>
Description	Maximum number of requested nodes per job. (Also used when REMAPCLASS is set to correctly route the job.)

MAX.NODE	
Example	`CLASSCFG[batch] MAX.NODE=64` Deny jobs requesting over 64 nodes access to the class *batch*.

MAX.PROC	
Format	<INTEGER>
Description	Maximum number of requested processors per job. (Also used when REMAPCLASS is set to correctly route the job.)
Example	`CLASSCFG[small] MAX.PROC[USER]=3,6`

MAX.PS	
Format	<INTEGER>
Description	Maximum requested processor-seconds.
Example	---

MAX.WCLIMIT	
Format	[[[DD:]HH:]MM:]SS
Description	Maximum allowed wallclock limit per job. (Also used when REMAPCLASS is set to correctly route the job.)
Example	`CLASSCFG[long] MAX.WCLIMIT=96:00:00`

MIN.NODE	
Format	<INTEGER>
Description	Minimum number of requested nodes per job. (Also used when REMAPCLASS is set to correctly route the job.)

MIN.NODE	
Example	`CLASSCFG[dev] MIN.NODE=16` Jobs must request at least 16 nodes to be allowed to access the class.

MIN.PROC	
Format	<INTEGER>
Description	Minimum number of requested processors per job. (Also used when REMAPCLASS is set to correctly route the job.)
Example	`CLASSCFG[dev] MIN.PROC=32` Jobs must request at least 32 processors to be allowed to access the class.

MIN.PS	
Format	<INTEGER>
Description	Minimum requested processor-seconds.
Example	---

MIN.TPN	
Format	<INTEGER>
Description	Minimum required tasks per node per job.
Example	---

MIN.WCLIMIT	
Format	[[[DD:]HH:]MM:]SS
Description	Minimum required wallclock limit per job. (Also used when REMAPCLASS is set to correctly route the job.)

MIN.WCLIMIT	
Example	---

NODEACCESSPOLICY	
Format	one of *SINGLETASK*, *SINGLEJOB*, *SINGLEUSER*, or *SHARED*
Description	Default node access policy associated with queue. This value will be overridden by any per job settings specified by the user at the job level. (See Node Access Policy overview.)
Example	`CLASSCFG[batch] NODEACCESSPOLICY=SINGLEJOB`

PARTITION	
Format	<STRING>
Description	Partition name where jobs associated with this class must run.
Example	`CLASSCFG[batch] PARTITION=p12`

PRIORITY	
Format	<INTEGER>
Description	Priority associated with the class. (See Priority overview.)
Example	`CLASSCFG[batch] PRIORITY=1000`

QDEF	
Format	<QOSID>
Description	Default QoS for jobs submitted to this class. You may specify a maximum of four **QDEF** entries per credential. Any QoSes specified after the fourth will not be accepted. In addition to classes, you may also specify **QDEF** for accounts, groups, and users.

QDEF	
Example	`CLASSCFG[batch] QDEF=base` Jobs submitted to class *batch* that do not explicitly request a QoS will have the QoS *base* assigned.

QLIST	
Format	<QOSID>[,<QOSID>]...
Description	List of accessible QoSs for jobs submitted to this class.
Example	`CLASSCFG[batch] QDEF=base QLIST=base,fast,special,bigio`

REQ.FEATURES	
Format	Comma- or pipe-delimited list of node features.
Description	Set of required features. If delimited by commas, all features are required; if delimited by the pipe symbol (|), at least one feature is required.
Example	`CLASSCFG[amd] REQ.FEATURES=ATHLON,AMD`

REQ.FLAGS	
Format	REQ.FLAGS can be used with only the INTERACTIVE flag.
Description	Sets the INTERACTIVE flag on jobs in this class.
Example	`CLASSCFG[orion] REQ.FLAGS=INTERACTIVE`

REQUIREDACCOUNTLIST	
Format	Comma-delimited list of accounts.
Description	List of accounts allowed to access and use a class (analogous to ***LIST** for other credentials).

REQUIREDACCOUNTLIST	
Example	`CLASSCFG[jasper] REQUIREDACCOUNTLIST=testers,development`

REQUIREDUSERLIST	
Format	Comma-delimited list of users.
Description	List of users allowed to access and use a class (analogous to ***LIST** for other credentials).
Example	`CLASSCFG[jasper] REQUIREDUSERLIST=john,u13,steve,guest`

REQUIREDQOSLIST	
Format	Comma-delimited list of QoSs
Description	List of QoSs allowed to access and use a class (analogous to ***LIST** for other credentials). The number of unique QoSs is limited by the Moab Maximum ACL limit, which defaults to 32.
Example	`CLASSCFG[jasper] REQUIREDQOSLIST=hi,lo`

SYSPRIO	
Format	<INTEGER>
Description	Value of system priority applied to every job submitted to this class.
Example	`CLASSCFG[special] SYSPRIO=100`

WCOVERRUN	
Format	[[[DD:]HH:]MM:]SS

WCOVERRUN	
Description	Tolerated amount of time beyond the specified wallclock limit.
Example	---

Enabling Queue Complex Functionality

Queue complexes allow an organization to build a hierarchy of queues and apply certain limits and rules to collections of these queues. Moab supports this functionality in two ways. The first way, queue mapping, is very simple but limited in functionality. The second method provides very rich functionality but requires more extensive configuration using the Moab hierarchical fairshare facility.

Queue Mapping

Queue mapping allows collections of queues to be mapped to a parent credential object against which various limits and policies can be applied, as in the following example.

```
QOSCFG[general]   MAXIJOB[USER]=14  PRIORITY=20
QOSCFG[prio]      MAXIJOB[USER]=8   PRIORITY=2000
# group short, med, and long jobs into 'general' QOS
CLASSCFG[short]   QDEF=general FSTARGET=30
CLASSCFG[med]     QDEF=general FSTARGET=40
CLASSCFG[long]    QDEF=general FSTARGET=30 MAXPROC=200
# group interactive and debug jobs into 'prio' QOS
CLASSCFG[inter]   QDEF=prio
CLASSCFG[debug]   QDEF=prio
CLASSCFG[premier] PRIORITY=10000
```

QoS Credential

The concept of a quality of service (QoS) credential is unique to Moab and is not derived from any underlying concept or peer service. In most cases, the QoS credential is used to allow a site to set up a selection of service levels for end-users to choose from on a long-term or job-by-job basis. QoSs differ from other credentials in that they are centered around special access where this access may allow use of additional services, additional resources, or improved responsiveness. Unique to this credential, organizations may also choose to apply different charge rates to the varying levels of service available within each QoS. As QoS is an internal credential, all QoS configuration occurs within Moab.

QoS access and QoS defaults can be mapped to users, groups, accounts, and classes, allowing limited service offering for key users. As mentioned, these services focus around increasing access to special scheduling capabilities & additional resources and improving job responsiveness. At a high level, unique QoS attributes can be broken down into the following:

- Usage Limit Overrides
- Service Targets
- Privilege Flags

- Charge Rate
- Access Controls

QoS Usage Limit Overrides

All credentials allow specification of job limits. In such cases, jobs are constrained by the most limiting of all applicable policies. With QoS override limits, however, jobs are limited by the override, regardless of other limits specified.

QoS Service Targets

Service targets cause the scheduler to take certain job-related actions as various responsiveness targets are met. Targets can be set for either job queue time or job expansion factor and cause priority adjustments, reservation enforcement, or preemption activation. In strict service centric organizations, Moab can be configured to trigger various events and notifications in the case of failure by the cluster to meet responsiveness targets.

QoS Privilege Flags

QoSs can provide access to special capabilities. These capabilities include preemption, job deadline support, backfill, next to run priority, guaranteed resource reservation, resource provisioning, dedicated resource access, and many others. See the complete list in the QoS Facility Overview section.

QoS Charge Rate

Associated with the QoSs many privileges is the ability to assign end-users costs for the use of these services. This charging can be done on a per-QoS basis and may be specified for both dedicated and use-based resource consumption. The Per QoS Charging section covers more details on QoS level costing configuration while the Charging and Allocation Management section provides more details regarding general single cluster and multi-cluster charging capabilities.

QoS Access Controls

QoS access control can be enabled on a per QoS basis using the MEMBERULIST attribute or specified on a *per-requestor* basis using the **QDEF** and **QLIST** attributes of the USERCFG, GROUPCFG, ACCOUNTCFG, and CLASSCFG parameters. See Managing QoS Access for more detail.

Related topics

- Identity Manager Interface
- Usage Limits

Job Attributes/Flags Overview

Job Attributes

FLAGS	
Format:	<FLAG>[:<FLAG>]...
Default:	---
Description:	Specifies job specific flags.
Example:	`FLAGS=ADVRES:RESTARTABLE` *The job can restart and should only utilize reserved resources.*

PLIST*	
Format:	<PARTITION_NAME>[^|&] [:<PARTITION_NAME>[^|&]]...
Default:	*[ALL]*
Description:	Specifies the list of partitions the object can access. If no partition list is specified, the object is granted default access to all partitions.
Example:	`PLIST=OldSP:Cluster1:O3K` *The object can access resources located in the OldSP, Cluster1, and/or O3K partitions.*

QDEF	
Format:	<QOS_NAME>
Default:	*[DEFAULT]*
Description:	Specifies the default QOS associated with the object.

QDEF	
Example:	`QDEF=premium` *The object is assigned the default QOS* *premium.*

QLIST*	
Format:	<QOS_NAME>[^|&] [:<QOS_NAME>[^|&]]...
Default:	*<QDEF>*
Description:	Specifies the list of QoSs the object can access. If no QOS list is specified, the object is granted access only to its default partition.
Example:	`QLIST=premium:express:bottomfeeder` *The object can access any of the 3 QoSs listed.*

By default, jobs may access QoSs based on the 'logical or' of the access lists associated with all job credentials. For example, a job associated with user "John," group "staff," and class "batch" may utilize QoSs accessible by any of the individual credentials. Thus the job's QOS access list, or QLIST, equals the 'or' of the user, group, and class QLIST's. (i.e., JOBQLIST = USERQLIST | GROUPQLIST | CLASSQLIST). If the ampersand symbol, '&', is associated with any list, this list is logically and'd with the other lists. If the carat symbol, '^', is associated with any object QLIST, this list is exclusively set, regardless of other object access lists using the following order of precedence user, group, account, QOS, and class. These special symbols affect the behavior of both QOS and partition access lists.

Job Flags

ADVRES	
Format:	ADVRES[:<RESID>]
Default:	Use available resources where ever found, whether inside a reservation or not.
Description:	Specifies the job may only utilize accessible, reserved resources. If <RESID> is specified, only resources in the specified reservation may be utilized.

ADVRES	
Example:	`FLAGS=ADVRES:META.1` *The job may only utilize resources located in the META.1 reservation.*

ARRAYJOBPARLOCK	
Format:	---
Default:	---
Description:	Specifies that the job array being submitted should not span across multiple partitions. This locks all sub jobs of the array to a single partition. If you want to lock all job arrays to a single partition, specify the ARRAYJOBPARLOCK parameter in `moab.cfg` to force this behavior on a global scale.
Example:	`> msub -t moab.[1-5]%3 -l walltime=30,flags=arrayjobparlock`

ARRAYJOBPARSPAN	
Format:	---
Default:	---
Description:	Specifies that the job array being submitted should span across multiple partitions. This is the default behavior in Moab, unless the ARRAYJOBPARLOCK parameter is specified in `moab.cfg`. This job flag overrides the **ARRAYJOBPARLOCK** parameter so that job arrays can be allowed to span multiple partitions at submit time.
Example:	`> msub -t moab.[1-5]%3 -l walltime=30,flags=arrayjobparspan`

GRESONLY	
Format:	GRESONLY
Default:	False
Description:	Uses no compute resources such as processors, memory, and so forth; uses only generic resources.

GRESONLY	
Example:	`> msub -l gres=matlab,walltime=300`

IGNIDLEJOBRSV	
Format:	IGNIDLEJOBRSV
Default:	N/A
Description:	Only applies to QOS. **IGNIDLEJOBRSV** allows jobs to start without a guaranteed walltime. Instead, it overlaps the idle reservations of real jobs and is preempted 2 minutes before the real job starts.
Example:	`QOSCFG[standby] JOBFLAGS=IGNIDLEJOBRSV`

NOQUEUE	
Format:	NOQUEUE
Default:	Jobs remain queued until they are able to run
Description:	Specifies that the job should be removed it is unable to allocate resources and start execution immediately.
Example:	`FLAGS=NOQUEUE` *The job should be removed unless it can start running at submit time.* This functionality is identical to the resource manager extension QUEUEJOB:FALSE.

NORMSTART	
Format:	NORMSTART
Default:	Moab passes jobs to a resource manager to schedule.
Description:	Specifies that the job is an internal system job and will not be started via an RM.

NORMSTART	
Example:	`FLAGS=NORMSTART` *The job begins running in Moab without a corresponding RM job.*

NOVMMIGRATE	
Format	NOVMMIGRATE
Default	Moab can migrate the VM associated with the job.
Description	Specifies that Moab may not migrate the VM that the job sets up.
Example	`msub -l walltime=INFINITY,template=VMTracking,os=linux,nodes=h3,jobflags=novmmigrate` *Moab will not migrate the new VM.*

PREEMPTEE	
Format:	PREEMPTEE
Default:	Jobs may not be preempted by other jobs
Description:	Specifies that the job may be preempted by other jobs which have the PREEMPTOR flag set.
Example:	`FLAGS=PREEMPTEE` *The job may be preempted by other jobs which have the PREEMPTOR flag set.*

PREEMPTOR	
Format:	PREEMPTOR
Default:	Jobs may not preempt other jobs
Description:	Specifies that the job may preempt other jobs which have the **PREEMPTEE** flag set .

PREEMPTOR	
Example:	`FLAGS=PREEMPTOR` *The job may preempt other jobs which have the PREEMPTEE flag set.*

PURGEONSUCCESSONLY	
Format	PURGEONSUCCESSONLY
Default	Completed jobs are sent to a queue for a short period of time before Moab purges them from the system.
Description	Specifies that Moab should only purge the job from the completed queue if it completed successfully. If the job failed, Moab will keep it in the queue indefinitely to allow you to restart it at any time. This flag is particularly useful for setup and take down jobs in job workflows. See **Creating workflows with job templates** on page 836 for more information.
Example	`FLAGS=PURGEONSUCCESSONLY` *If the job fails, Moab will not purge it from the completed job queue.*

RESTARTABLE	
Format:	RESTARTABLE
Default:	Jobs may not be restarted if preempted.
Description:	Specifies jobs can be *requeued* and later restarted if preempted.
Example:	`FLAGS=RESTARTABLE` *The associated job can be preempted and restarted at a later date.*

SUSPENDABLE	
Format:	SUSPENDABLE

SUSPENDABLE	
Default:	Jobs may not be suspended if preempted.
Description:	Specifies jobs can be *suspended* and later resumed if preempted.
Example:	`FLAGS=SUSPENDABLE` *The associated job can be suspended and resumed at a later date.*

SYSTEMJOB	
Format:	SYSTEMJOB
Default:	N/A
Description:	Creates an internal system job that does not require resources.
Example:	`FLAGS=SYSTEMJOB`

WIDERSVSEARCHALGO	
Format:	<BOOLEAN>
Default:	---
Description:	When Moab is determining when and where a job can run, it either searches for the most resources or the longest range of resources. In almost all cases searching for the longest range is ideal and returns the soonest starttime. In some rare cases, however, a particular job may need to search for the most resources. In those cases this flag can be used to have the job find the soonest starttime. The flag can be specified at submit time, or you can use mjobctl -m to modify the job after it has been submitted. See the RSVSEARCHALGO parameter.
Example:	`> msub -l flags=widersvsearchalgo` `> mjobctl -m flags+=widersvsearchalgo job.1`

Related topics

- Setting Per-Credential Job Flags

Scheduler Commands

Moab Commands

Command	Description
checkjob	Provide detailed status report for specified job
checknode	Provide detailed status report for specified node
mcredctl	Controls various aspects about the credential objects within Moab
mdiag	Provide diagnostic reports for resources, workload, and scheduling
mjobctl	Control and modify job
mnodectl	Control and modify nodes
moab	Control the Moab daemon
mrmctl	Query and control resource managers
mrsvctl	Create, control and modify reservations
mschedctl	Modify scheduler state and behavior
mshow	Displays various diagnostic messages about the system and job queues
mshow -a	Query and show available system resources
msub	Scheduler job submission
mvcctl	Create, modify, and delete VCs
mvmctl	Create, control and modify VMs
showbf	Show current resource availability
showhist.moab.pl	Show past job information

Command	Description
showq	Show queued jobs
showres	Show existing reservations
showstart	Show estimates of when job can/will start
showstate	Show current state of resources
showstats	Show usage statistics
showstats -f	Show various tables of scheduling/system performance

Moab command options

For many Moab commands, you can use the following options to specify that Moab will run the command in a different way or different location from the configured default. These options do not change your settings in the configuration file; they override the settings for this single instance of the command.

Option	Description
--about	Displays build and version information and the status of your Moab license
--help	Displays usage information about the command
--host=*<server-HostName>*	Causes Moab to run the client command on the specified host
--loglevel=*<logLevel>*	Causes Moab to write log information to STDERR as the client command is running. For more information, see **Logging Overview** on page 676.
--msg=*<message>*	Causes Moab to annotate the action in the event log
--port=*<server-Port>*	Causes Moab to run the command using the port specified
--timeout=*<seconds>*	Sets the maximum time that the client command will wait for a response from the Moab server

Option	Description
--version	Displays version information
--xml	Causes Moab to return the command output in XML format

Commands Providing Maui Compatibility

The following commands are deprecated. Click the link for respective deprecated commands to see the updated replacement command for each.

Command	Description
canceljob	Cancel job
changeparam	Change in memory parameter settings
diagnose	Provide diagnostic report for various aspects of resources, workload, and scheduling
releasehold	Release job defers and holds
releaseres	Release reservations
runjob	Force a job to run immediately
sethold	Set job holds
setqos	Modify job QOS settings
setres	Set an admin/user reservation
setspri	Adjust job/system priority of job
showconfig	Show current scheduler configuration

Status Commands

The status commands organize and present information about the current state and historical statistics of the scheduler, jobs, resources, users, and accounts. The following table presents the primary status commands and flags.

Command	Description
checkjob	Displays detailed job information such as job state, resource requirements, environment, constraints, credentials, history, allocated resources, and resource utilization.
checknode	Displays detailed node information such as node state, resources, attributes, reservations, history, and statistics.
mdiag -f	Displays summarized fairshare information and any unexpected fairshare configuration.
mdiag -j	Displays summarized job information and any unexpected job state.
mdiag -n	Displays summarized node information and any unexpected node state.
mdiag -p	Displays summarized job priority information.
mschedctl -f	Resets internal statistics.
showstats -f	Displays various aspects of scheduling performance across a job duration/job size matrix.
showq [-r|-i]	Displays various views of currently queued active, idle, and non-eligible jobs.
showstats -g	Displays current and historical usage on a per group basis.
showstats -u	Displays current and historical usage on a per user basis.
showstats -v	Displays high level current and historical scheduling statistics.

Job Management Commands

Moab shares job management tasks with the resource manager. Typically, the scheduler only modifies scheduling relevant aspects of the job such as partition access, job priority, charge account, and hold state. The following table covers the available job management commands. The Commands Overview lists all available commands.

Command	Description
canceljob	Cancels existing job.
checkjob	Displays job state, resource requirements, environment, constraints, credentials, history, allocated resources, and resource utilization.
mdiag -j	Displays summarized job information and any unexpected job state.
releasehold -a	Removes job holds or deferrals.
runjob	Starts job immediately, if possible.
sethold	Sets hold on job.
setqos	Sets/modifies QoS of existing job.
setspri	Adjusts job/system priority of job.

Related topics

- Job State Definitions

Reservation Management Commands

Moab exclusively controls and manages all advance reservation features including both standing and administrative reservations. The following table covers the available reservation management commands.

Command	Description
mdiag -r	Displays summarized reservation information and any unexpected state.
mrsvctl	Reservation control.
mrsvctl -r	Removes reservations.
mrsvctl -c	Creates an administrative reservation.
showres	Displays information regarding location and state of reservations.

Policy/Configuration Management Commands

Moab allows dynamic modification of most scheduling parameters allowing new scheduling policies, algorithms, constraints, and permissions to be set at any time. Changes made via Moab client commands are temporary and are overridden by values specified in Moab configuration files the next time Moab is shut down and restarted. The following table covers the available configuration management commands.

Command	Description
mschedctl -l	Displays triggers, messages, and settings of all configuration parameters.
mschedctl	Controls the scheduler (behavior, parameters, triggers, messages).
mschedctl -m	Modifies system values.

End-user Commands

While the majority of Moab commands are tailored for use by system administrators, a number of commands are designed to extend the knowledge and capabilities of end-users. The following table covers the commands available to end-users.

When using Active Directory as a central authentication mechanism, all nodes must be reported with a different name when booted in both Linux and Windows (for instance, `node01-l` for Linux and `node01` for Windows). If a machine account with the same name is created for each OS, the most recent OS will remove the previously-joined machine account. The nodes must report to Moab with the same hostname. This can be done by using aliases (adding all node names to the `/etc/hosts` file on the system where Moab is running) and ensuring that the Linux resource manager reports the node with its global name rather than the Linux-specific one (`node01` rather than `node01-l`).

Command	Description
canceljob	Cancels existing job.
checkjob	Displays job state, resource requirements, environment, constraints, credentials, history, allocated resources, and resource utilization.
msub	Submit a new job.
releaseres	Releases a user reservation.

Command	Description
setres	Create a user reservation.
showbf	Shows resource availability for jobs with specific resource requirements.
showq	Displays detailed prioritized list of active and idle jobs.
showstart	Shows estimated start time of idle jobs.
showstats	Shows detailed usage statistics for users, groups, and accounts, to which the end-user has access.

Related topics

- Commands Overview

Commands

checkjob

Synopsis

checkjob [exact:*jobid*] [-l *policylevel*] [-n *nodeid*] [-q *qosid*] [-r *reservationid*] [-v] [--flags=future] [--blocking] *jobid*

Overview

checkjob displays detailed job state information and diagnostic output for a specified job. Detailed information is available for queued, blocked, active, and recently completed jobs. The checkjob command shows the master job of an array as well as a summary of array sub-jobs, but does not display all sub-jobs. Use checkjob -v to display all job-array sub-jobs.

Access

This command can be run by level 1-3 Moab administrators for any job. Also, end users can use checkjob to view the status of their own jobs.

Arguments

--blocking	
Format	`--blocking`
Description	Do not use cache information in the output. The `--blocking` flag retrieves results exclusively from the scheduler.
Example	`> checkjob -v --blocking 1234` *Display real time data about job* `1234`*.*

--flags	
Format	`--flags=future`
Description	Evaluates future eligibility of job (ignore current resource state and usage limitations).
Example	`> checkjob -v --flags=future 6235` *Display reasons why idle job is blocked ignoring node state and current node utilization constraints.*

exact	
Format	exact:*<JOBID>*
Description	Searches for and returns the exact job ID
Example	`> checkjob exact:1.job_dependency1`

-l (Policy level)	
Format	*<POLICYLEVEL>* *HARD*, *SOFT*, or *OFF*

-l (Policy level)	
Description	Reports job start eligibility subject to specified throttling policy level.
Example	`> checkjob -l SOFT 6235` `> checkjob -l HARD 6235`

-n (NodeID)	
Format	*<NODEID>*
Description	Checks job access to specified node and preemption status with regards to jobs located on that node.
Example	`> checkjob -n node113 6235`

-q (QoS)	
Format	*<QOSID>*
Description	Checks job access to specified QoS *<QOSID>*.
Example	`> checkjob -q special 6235`

-r (Reservation)	
Format	*<RSVID>*
Description	Checks job access to specified reservation *<RSVID>*.
Example:	`> checkjob -r orion.1 6235`

-v (Verbose)	
Description	Sets verbose mode. If the job is part of an array, the `-v` option shows pertinent array information before the job-specific information (see Example 2 and Example 3 for differences between standard output and -v output). Specifying the double verbose (`-v -v`) displays additional information about the job. See the Output table for details.
Example	`> checkjob -v 6235`

Details

This command allows any Moab administrator to check the detailed status and resource requirements of an active, queued, or recently completed job. Additionally, this command performs numerous diagnostic checks and determines if and where the job could potentially run. Diagnostic checks include policy violations, reservation constraints, preemption status, and job to resource mapping. If a job cannot run, a text reason is provided along with a summary of how many nodes are and are not available. If the `-v` flag is specified, a node by node summary of resource availability will be displayed for idle jobs.

Job Eligibility

If a job cannot run, a text reason is provided along with a summary of how many nodes are and are not available. If the `-v` flag is specified, a node by node summary of resource availability will be displayed for idle jobs. For job level eligibility issues, one of the following reasons will be given:

Reason	Description
job has hold in place	one or more job holds are currently in place
insufficient idle procs	there are currently not adequate processor resources available to start the job
idle procs do not meet requirements	adequate idle processors are available but these do not meet job requirements
start date not reached	job has specified a minimum *start date* which is still in the future
expected state is not idle	job is in an unexpected state
state is not idle	job is not in the idle state

Reason	Description
dependency is not met	job depends on another job reaching a certain state
rejected by policy	job start is prevented by a throttling policy

If a job cannot run on a particular node, one of the following 'per node' reasons will be given:

Reason	Description
Class	Node does not allow required job class/queue
CPU	Node does not possess required processors
Disk	Node does not possess required local disk
Features	Node does not possess required node features
Memory	Node does not possess required real memory
Network	Node does not possess required network interface
State	Node is not Idle or Running

Reservation Access

The -r flag can be used to provide detailed information about job access to a specific reservation

Preemption Status

If a job is marked as a preemptor and the -v and -n flags are specified, checkjob will perform a job by job analysis for all jobs on the specified node to determine if they can be preempted.

Output

The checkjob command displays the following job attributes:

Attribute	Value	Description
Account	*<STRING>*	Name of account associated with job

Attribute	Value	Description
Actual Run Time	[[[DD:]HH:]MM:]SS	Length of time job actually ran. This info is only displayed in simulation mode.
Allocated Nodes	Square bracket delimited list of node and processor ids	List of nodes and processors allocated to job
Applied Nodeset**	*<STRING>*	Nodeset used for job's node allocation
Arch	*<STRING>*	Node architecture required by job
Attr	square bracket delimited list of job attributes	Job Attributes (i.e. `[BACKFILL][PREEMPTEE]`)
Available Memory**	*<INTEGER>*	The available memory requested by job. Moab displays the relative or exact value by returning a comparison symbol (>, <, >=, <=, or ==) with the value (i.e. `Available Memory <= 2048`).
Available Swap**	*<INTEGER>*	The available swap requested by job. Moab displays the relative or exact value by returning a comparison symbol (>, <, >=, <=, or ==) with the value (i.e. `Available Swap >= 1024`).
Average Utilized Procs*	*<FLOAT>*	Average load balance for a job
Avg Util Resources Per Task*	*<FLOAT>*	
BecameEligible	*<TIMESTAMP>*	The date and time when the job moved from Blocked to Eligible.
Bypass	*<INTEGER>*	Number of times a lower priority job with a later submit time ran before the job
CheckpointStartTime**	`[[[DD:]HH:]MM:]SS`	The time the job was first checkpointed

Attribute	Value	Description
Class	[*<CLASS NAME>* *<CLASS COUNT>*]	Name of class/queue required by job and number of class initiators required per task.
Dedicated Resources Per Task*	Space-delimited list of *<STRING>* :*<INTEGER>*	Resources dedicated to a job on a per-task basis
Disk	*<INTEGER>*	Amount of local disk required by job (in MB)
Estimated Walltime	[[[DD:]HH:]MM:]SS	The scheduler's estimated walltime. In simulation mode, it is the actual walltime.
EnvVariables**	Comma-delimited list of *<STRING>*	List of environment variables assigned to job
Exec Size*	*<INTEGER>*	Size of job executable (in MB)
Executable	*<STRING>*	Name of command to run
Features	Square bracket delimited list of *<STRING>*s	Node features required by job
Flags		
Group	*<STRING>*	Name of UNIX group associated with job
Holds	Zero or more of User, System, and Batch	Types of job holds currently applied to job
Image Size	*<INTEGER>*	Size of job data (in MB)
IWD (Initial Working Directory)	*<DIR>*	Directory to run the executable in
Job Messages**	*<STRING>*	Messages attached to a job
Job Submission**	*<STRING>*	Job script submitted to RM

Attribute	Value	Description
Memory	*<INTEGER>*	Amount of real memory required per node (in MB)
Max Util Resources Per Task*	*<FLOAT>*	
NodeAccess*		
Nodecount	*<INTEGER>*	Number of nodes required by job
Opsys	*<STRING>*	Node operating system required by job
Partition Mask	ALL or colon delimited list of partitions	List of partitions the job has access to
PE	*<FLOAT>*	Number of processor-equivalents requested by job
Per Partition Priority**	Tabular	Table showing job template priority for each partition
Priority Analysis**	Tabular	Table showing how job's priority was calculated: `Job PRIORITY* Cred( User:Group:Class) Serv (QTime)`
QOS	*<STRING>*	Quality of Service associated with job
Reservation	*<RSVID>* (*<TIME1>* - *<TIME2>* Duration: *<TIME3>*)	RESID specifies the reservation id, TIME1 is the relative start time, TIME2 the relative end time, TIME3 the duration of the reservation
Req	[*<INTEGER>*] TaskCount: *<INTEGER>* Partition: *<partition>*	A job requirement for a single type of resource followed by the number of tasks instances required and the appropriate partition
StageIn	*<SOURCE>*%*<DESTINATION>*	The *<SOURCE>* is the username, hostname, directory and file name of origin for the file(s) that Moab will stage in for this job. The *<DESTINATION>* is the username, hostname, directory and file name where Moab will place the file during this job. See **About data staging** on page 880 for more information.

Attribute	Value	Description
StageInSize	*<INTEGER><UNIT>*	The size of the file Moab will stage in for this job. *<UNIT>* can be KB, MB, GB, or TB. See **About data staging** on page 880 for more information.
StageOut	*<SOURCE>* %*<DESTINATION>*	The *<SOURCE>* is the username, hostname, directory and file name of origin for the file(s) that Moab will stage out for this job. The *<DESTINATION>* is the username, hostname, directory and file name where Moab will place the file during this job. See **About data staging** on page 880 for more information.
StageOutSize	*<INTEGER><UNIT>*	The size of the file Moab will stage out for this job. *<UNIT>* can be KB, MB, GB, or TB. See **About data staging** on page 880 for more information.
StartCount	*<INTEGER>*	Number of times job has been started by Moab
StartPriority	*<INTEGER>*	Start priority of job
StartTime	*<TIME>*	Time job was started by the resource management system
State	One of Idle, Starting, Running, etc. See **Job States** on page 107 for all possible values.	Current Job State
SubmitTime	*<TIME>*	Time job was submitted to resource management system
Swap	*<INTEGER>*	Amount of swap disk required by job (in MB)
Task Distribution*	Square bracket delimited list of nodes	
Time Queued		
Total Requested Nodes**	*<INTEGER>*	Number of nodes the job requested
Total Requested Tasks	*<INTEGER>*	Number of tasks requested by job

Attribute	Value	Description
User	*<STRING>*	Name of user submitting job
Utilized Resources Per Task*	*<FLOAT>*	
WallTime	[[[DD:]HH:]MM:]SS of [[[DD:]HH:]MM:]SS	Length of time job has been running out of the specified limit

In the above table, fields marked with an asterisk (*) are only displayed when set or when the `-v` flag is specified. Fields marked with two asterisks (**) are only displayed when set or when the `-v -v` flag is specified.

Example 3-10: checkjob 717

```
> checkjob 717
job 717
State: Idle
Creds:  user:jacksond  group:jacksond  class:batch
WallTime: 00:00:00 of 00:01:40
SubmitTime: Mon Aug 15 20:49:41
  (Time Queued  Total: 3:12:23:13  Eligible: 3:12:23:11)
TerminationDate:   INFINITY  Sat Oct 24 06:26:40
Total Tasks: 1
Req[0]  TaskCount: 1  Partition: ALL
Network: ---  Memory >= 0  Disk >= 0  Swap >= 0
Opsys: ---  Arch: ---  Features: ---

IWD:            /home/jacksond/moab/moab-4.2.3
Executable:     STDIN
Flags:          RESTARTABLE,NORMSTART
StartPriority:  5063
Reservation '717' (  INFINITY ->   INFINITY  Duration: 00:01:40)
Note:  job cannot run in partition base (idle procs do not meet requirements : 0 of 1
procs found)
idle procs:   4  feasible procs:   0
Rejection Reasons: [State        :    3][ReserveTime  :    1]
cannot select job 717 for partition GM (partition GM does not support requested class
batch)
```

The example job cannot be started for two different reasons.

- It is temporarily blocked from partition *base* because of node state and node reservation conflicts.
- It is permanently blocked from partition *GM* because the requested class *batch* is not supported in that partition.

Example 3-11: Using checkjob (no -v) on a job array master job:

```
checkjob array.1
job array.1

AName: array
Job Array Info:
  Name: array.1

Sub-jobs:         10
  Active:          6 ( 60.0%)
  Eligible:        2 ( 20.0%)
  Blocked:         2 ( 20.0%)
  Complete:        0 (  0.0%)
```

Example 3-12: Using checkjob -v on a job array master job:

```
$ checkjob -v array.1
job array.1

AName: array
Job Array Info:
  Name: array.1
  1 : array.1.1 : Running
  2 : array.1.2 : Running
  3 : array.1.3 : Running
  4 : array.1.4 : Running
  5 : array.1.5 : Running
  6 : array.1.6 : Running
  7 : array.1.7 : Idle
  8 : array.1.8 : Idle
  9 : array.1.9 : Blocked
  10 : array.1.10 : Blocked

Sub-jobs:         10
  Active:          6 ( 60.0%)
  Eligible:        2 ( 20.0%)
  Blocked:         2 ( 20.0%)
  Complete:        0 (  0.0%)
```

Example 3-13: Using checkjob -v on a data staging job

```
$ checkjob -v moab.14.dsin
job moab.14.dsin

AName: moab.14.dsin
State: Running
Creds:  user:fred  group:company
WallTime:   00:00:00 of 00:01:01
SubmitTime: Wed Apr 16 10:07:19
 (Time Queued  Total: 00:00:00  Eligible: 00:00:00)

StartTime: Wed Apr 16 10:07:19
TemplateSets:  dsin
Triggers: 78$start+0@0.000000:exec@/opt/moab/tools/datastaging/ds_move_rsync --
stagein:FALSE
Total Requested Tasks: 1

Req[0]  TaskCount: 1  Partition: SHARED
Dedicated Resources Per Task: bandwidth: 1
NodeAccess: SHARED

Allocated Nodes:
[GLOBAL:1]

Job Group:  moab.14
SystemID:   moab
SystemJID:  moab.14.dsin
Task Distribution: GLOBAL
IWD:            $HOME/test/datastaging
SubmitDir:      $HOME/test/datastaging
StartCount:     1
Parent VCs:     vc11
User Specified Partition List:   local
Partition List: local
SrcRM:          internal
Flags:          NORMSTART,GRESONLY,TEMPLATESAPPLIED
Attr:           dsin
StageInSize:    386MB
StageOutSize:   100MB
StageIn:        fred@remotelab:/home/fred/input1/%fred@scratch:/home/fred/input1/
StageIn:        fred@remotelab:/home/fred/input2/%fred@scratch:/home/fred/input2/
StageIn:        fred@remotelab:/home/fred/input3/%fred@scratch:/home/fred/input3/
StageOut:       fred@scratch:/home/fred/output/%fred@remotelab:/home/fred/output/
StartPriority:  1
  SJob Type:             datastaging
  Completion Policy:     datastaging
PE:             0.00
Reservation 'moab.14.dsin' (-00:00:06 -> 00:00:55  Duration: 00:01:01)
```

Related topics

- showhist.moab.pl - explains how to query for past job information
- Moab Client Installation - explains how to distribute this command to client nodes
- mdiag -j command - display additional detailed information regarding jobs
- showq command - showq high-level job summaries
- JOBCPURGETIME parameter - specify how long information regarding completed jobs is maintained
- diagnosing job preemption

checknode

Synopsis

checknode *options nodeID*
ALL

Overview

This command shows detailed state information and statistics for nodes that run jobs.

The following information is returned by this command:

Name	Description
Disk	Disk space available
Memory	Memory available
Swap	Swap space available
State	Node state
Opsys	Operating system
Arch	Architecture
Adapters	Network adapters available
Features	Features available
Classes	Classes available
StateTime	Time node has been in current state in HH:MM:SS notation
Downtime	Displayed only if downtime is scheduled
Load	CPU Load (Berkley one-minute load average)
TotalTime	Total time node has been detected since statistics initialization expressed in *HH:MM:SS* notation

Name	Description
UpTime	Total time node has been in an available (Non-Down) state since statistics initialization expressed in *HH:MM:SS* notation (percent of time up: UpTime/TotalTime)
ActiveTime	Total time node has been busy (allocated to active jobs) since statistics initialization expressed in *HH:MM:SS* notation (percent of time busy: BusyTime/TotalTime)
EffNodeAccessPolicy	Configured effective node access policy

After displaying this information, some analysis is performed and any unusual conditions are reported.

Access

By default, this command can be run by any Moab Administrator (see ADMINCFG).

Parameters

Name	Description
NODE	Node name you want to check. Moab uses regular expressions to return any node that contains the provided argument. For example, if you ran checknode node1, Moab would return information about `node1`, `node10`, `node100`, etc. If you want to limit the results to `node1` only, you would run checknode "^node1$".

Flags

Name	Description
ALL	Returns checknode output on all nodes in the cluster.
-h	Help for this command.
-v	Returns verbose output.
--xml	Output in XML format. Same as mdiag -n --xml.

Example 3-14: checknode

```
> checknode P690-032
node P690-032

State:      Busy  (in current state for 11:31:10)
Configured Resources: PROCS: 1  MEM: 16G  SWAP: 2000M  DISK: 500G
Utilized   Resources: PROCS: 1
Dedicated  Resources: PROCS: 1
Opsys:      AIX       Arch:      P690
Speed:      1.00      CPULoad:   1.000
Network:    InfiniBand,Myrinet
Features:   Myrinet
Attributes: [Batch]
Classes:    [batch]

Total Time: 5:23:28:36  Up: 5:23:28:36 (100.00%)  Active: 5:19:44:22 (97.40%)

Reservations:
  Job '13678'(x1)  10:16:12:22 -> 12:16:12:22 (2:00:00:00)
  Job '13186'(x1)  -11:31:10 -> 1:12:28:50 (2:00:00:00)
Jobs:  13186
```

Example 3-15: checknode ALL

```
> checknode ALL
node ahe

State:      Idle  (in current state for 00:00:30)
Configured Resources: PROCS: 12  MEM: 8004M  SWAP: 26G  DISK: 1M
Utilized   Resources: PROCS: 1  SWAP: 4106M
Dedicated  Resources: ---
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   1.400
Flags:      rmdetected
Classes:    [batch]
RM[ahe]*:   TYPE=PBS
EffNodeAccessPolicy: SHARED

Total Time: 00:01:44  Up: 00:01:44 (100.00%)  Active: 00:00:00 (0.00%)

Reservations:  ---
node ahe-ubuntu32

State:   Running  (in current state for 00:00:05)
Configured Resources: PROCS: 12  MEM: 2013M  SWAP: 3405M  DISK: 1M
Utilized   Resources: PROCS: 6  SWAP: 55M
Dedicated  Resources: PROCS: 6
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   2.000
Flags:      rmdetected
Classes:    [batch]
RM[ahe]*:   TYPE=PBS
EffNodeAccessPolicy: SHARED

Total Time: 00:01:44  Up: 00:01:44 (100.00%)  Active: 00:00:02 (1.92%)

Reservations:
  6x2  Job:Running  -00:00:07 -> 00:01:53 (00:02:00)
  7x2  Job:Running  -00:00:06 -> 00:01:54 (00:02:00)
  8x2  Job:Running  -00:00:05 -> 00:01:55 (00:02:00)
Jobs:        6,7,8
node ahe-ubuntu64

State:      Busy  (in current state for 00:00:06)
Configured Resources: PROCS: 12  MEM: 2008M  SWAP: 3317M  DISK: 1M
Utilized   Resources: PROCS: 12  SWAP: 359M
Dedicated  Resources: PROCS: 12
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   0.000
Flags:      rmdetected
Classes:    [batch]
RM[ahe]*:   TYPE=PBS
EffNodeAccessPolicy: SHARED

Total Time: 00:01:44  Up: 00:01:44 (100.00%)  Active: 00:00:55 (52.88%)

Reservations:
  0x2  Job:Running  -00:01:10 -> 00:00:50 (00:02:00)
  1x2  Job:Running  -00:00:20 -> 00:01:40 (00:02:00)
  2x2  Job:Running  -00:00:20 -> 00:01:40 (00:02:00)
  3x2  Job:Running  -00:00:17 -> 00:01:43 (00:02:00)
  4x2  Job:Running  -00:00:13 -> 00:01:47 (00:02:00)
  5x2  Job:Running  -00:00:07 -> 00:01:53 (00:02:00)
```

```
Jobs:         0,1,2,3,4,5
ALERT:  node is in state Busy but load is low (0.000)
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mdiag -n
- showstate

mcredctl

Synopsis

mcredctl [-d *credtype*[:*credid*]] [-h *credtype*:*credid*] [-l *credtype*] [-q {role|limit|profile|accessfrom|accessto|policies} *credtype*[:*credid*]] [--format=xml] [-r {stats|credits|fairshare} *credtype*[:*credid*]] [-t *<STARTTIME>*[,*<ENDTIME>*]

Overview

The mcredctl command controls various aspects about the credential objects within Moab. It can be used to display configuration, limits, roles, and relationships for various Moab credential objects.

Arguments

In all cases *<CREDTYPE>* is one of `acct`, `group`, `user`, `class`, or `qos`.

In most cases it is necessary to use the `--format=xml` flag in order to print the output (see examples below for specific syntax requirements).

-d - DESTROY	
Format	*<TYPE>*:*<VAL>*
Description	Purge a credential from `moab.cfg` (does not delete credential from memory).
Example	`> mcredctl -d user:john` *All references to* ***USERCFG****[john] will be commented out of* `moab.cfg`*)*

-h - HOLD	
Format	*<TYPE>*:*<VAL>*
Description	Toggles whether a given credentials' jobs should be place on hold or not.
Example	`> mcredctl -h user:john` *User [john] will be put on hold.*

-l - LIST	
Format	*<TYPE>*
Description	List the various sub-objects of the specified credential.
Example	`> mcredctl -l user --format=xml` *List all users within Moab in XML.* `> mcredctl -l group --format=xml` *List all groups within Moab in XML.*

-q - QUERY	
Format	{role | accessfrom | accessto | limit| profile | policies} limit *<TYPE>* policies *<TYPE>* role *<USER>*:*<USERID>* profile *<TYPE>*[:*<VAL>*] accessfrom *<TYPE>*[:*<VAL>*] accessto *<TYPE>*[:*<VAL>*]
Description:	Display various aspects of a credential (formatted in XML)

-q - QUERY	
Example:	`> mcredctl -q role user:bob --format=xml` *View user bob's administrative role within Moab in XML* `> mcredctl -q limit acct --format=xml` *Display limits for all accounts in XML* `> mcredctl -q policies user:bob` *View limits organized by credential for user bob on each partition and resource manager*

-r - RESET	
Format	*<TYPE>*
Description	Resets the credential within Moab.
Example	`> mcredctl -r user:john` *Resets the credential of user john*

-t - TIMEFRAME	
Format	*<STARTTIME>*[,*<ENDTIME>*]
Description	Can be used in conjunction with the `-q` profile option to display profiling information for the specified timeframe.
Example	`> mcredctl -q profile user -t 14:30_06/20`

Credential Statistics XML Output

Credential statistics can be requested as XML (via the `--format=xml` argument) and will be written to STDOUT in the following format:

```
> mcredctl -q profile user --format=xml -o time:1182927600,1183013999
<Data>
  <user ...>
    <Profile ...>
    </Profile>
  </user>
</Data>
```

Example 3-16: Deleting a group

```
> mcredctl -d group:john
GROUPCFG[john] Successfully purged from config files
```

Example 3-17: List users in XML format

```
> mcredctl -l user --format=xml
<Data><user ID="john"</user><user ID="john"></user><user ID="root"></user><user
ID="dev"></user></Data>
```

Example 3-18: Display information about a user

```
> mcredctl -q role user:john --format=xml
<Data><user ID="test" role="admin5"></user></Data>
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes

mdiag

Synopsis

mdiag -a [*accountid*]

mdiag -b [-l *policylevel*] [-t *partition*]

mdiag -c [*classid*]

mdiag -C [*configfile*] // diagnose config file syntax

mdiag -e [-w <*starttime*>|<*endtime*>|<*eventtypes*>|<*oidlist*>|<*eidlist*>|<*objectlist*>] --xml

mdiag -f [-o user|group|acct|qos|class] [-v]

mdiag -g [*groupid*]

mdiag -G [*Green*]

mdiag -j [*jobid*] [-t <*partition*>] [-v] [--blocking]

mdiag -L [-v] // diagnose usage limits

mdiag -n [-A <*creds*>] [-t *partition*] [*nodeid*] [-v]

mdiag -p [-t *partition*] [-v] // diagnose job priority

mdiag -q [*qosid*]

mdiag -r [*reservationid*] [-v] [-w type=<*type*>] [--blocking]

mdiag -R [*resourcemanagername*] [-v]

mdiag -s [*standingreservationid*] [--blocking]

mdiag -S [-v] // diagnose scheduler

mdiag -t [-v] // diagnose partitions

mdiag -T [*triggerid*] [-v] [--blocking]

mdiag -u [*userid*]

mdiag [--format=xml]

Overview

The mdiag command is used to display information about various aspects of the cluster and the results of internal diagnostic tests. In summary, it provides the following:

- current object health and state information
- current object configuration (resources, policies, attributes, etc)
- current and historical performance/utilization information
- reports on recent failure
- object messages

Some mdiag options gather information from the Moab cache which prevents them from interrupting the scheduler, but the `--blocking` option can be used to bypass the cache and interrupt the scheduler.

Arguments

Argument	Description
-a [accountid]	Display account information
-b	Display information on jobs blocked by policies, holds, or other factors. If blocked job diagnostics are specified, the `-t` option is also available to constrain the report to analysis of particular partition. Also, with blocked job diagnosis, the `-l` option can be used to specify the analysis policy level.
-c [classid]	Display class information

Argument	Description
-C [file]	With the vast array of options in the configuration file, the `-C` option does not validate function, but it does analyze the configuration file for syntax errors including use of invalid parameters, deprecated parameters, and some illegal values. If you start Moab with the -e flag, Moab evaluates the configuration file at startup and quits if an error exists.
-e	Moab will do a query for all events whose *eventtime* starts at *<starttime>* and matches the search criteria. This works only when Moab is configured with ODBC MySQL. The syntax is: mdiag -e[-w *<starttime>*|*<eventtypes>*| *<oidlist>*|*<eidlist>*|*<objectlist>*] --xml • *starttime* default is - • • *eventtypes* default is command delimited, the default is all event types (possible values can be found in the EventType table in the Moab database) • *oidlist* is a comma-delimited list of object ids, the default is all objects ids • *eidlist* is a comma-delimited list of specific event ids, the default is all event ids • *objectlist* is a comma-delimited list of object types, the default is all object types (possible values can be found in the ObjectType table in the Moab database)
-f	Display fairshare information
-g [groupid]	display group information
-G [Green]	display power management information
-j [jobid]	display job information
-L	display limits
-n [nodeid]	display nodes If node diagnostics are specified, the `-t` option is also available to constrain the report to a particular partition.
-p	display job priority. If priority diagnostics are specified, the `-t` option is also available to constrain the report to a particular partition.
-q [qosid]	display qos information

Argument	Description
-r [reservationid]	display reservation information
-R [rmid]	display resource manager information
-s [srsv]	display standing reservation information
-S	display general scheduler information
-t	display configuration, usage, health, and diagnostic information about partitions maintained by Moab
-T [triggerid]	display trigger information
-u [userid]	display user information
--format=xml	display output in XML format

XML Output

Information for most of the options can be reported as XML as well. This is done with the command `mdiag -<option> <CLASS_ID> --format=xml`. For example, XML-based class information will be written to STDOUT in the following format:

```
<Data>
  <class <ATTR>="<VAL>" ... >
    <stats <ATTR>="<VAL>" ... >
      <Profile <ATTR>="<VAL>" ... >
      </Profile>
    </stats>
  </class>
<Data>
  ...
</Data>
```

Of the mdiag options, only `-G` and `-L` cannot be reported as XML.

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- checkjob
- checknode

mdiag -a

Synopsis

mdiag -a [*accountid*]

Overview

The mdiag -a command provides detailed information about the accounts (aka projects) Moab is currently tracking. This command also allows an administrator to verify correct throttling policies and access provided to and from other credentials.

Example 3-19: Generating information about accounts

```
> mdiag -a
evaluating acct information
Name          Priority        Flags         QDef      QOSList*
PartitionList Target  Limits
engineering       100           -         high      high,urgent,low              [A]
[B]      30.00  MAXJOB=50,75  MAXPROC=400,500
marketing           1           -          low      low                          [A]
       5.00  MAXJOB=100,110  MAXPS=54000,54500
it                 10           -       DEFAULT      DEFAULT,high,urgent,low     [A]
     100.00  MAXPROC=100,1250 MAXPS=12000,12500
  FSWEIGHT=1000
development       100           -        high       high,urgent,low              [A]
[B]      30.00  MAXJOB=50,75 MAXNODE=100,120
research          100           -        high       DEFAULT,high,low             [A]
[B]      30.00  MAXNODE=400,500 MAXPS=900000,1000000
DEFAULT             0           -           -      -                             -
       0.00  -
```

Related topics

- Account credential

mdiag -b

Synopsis

mdiag -b [-l *policylevel*] [-t *partition*]

Overview

The mdiag -b command returns information about blocked jobs.

mdiag -c

Synopsis

mdiag -c [-v] [*classid*]

Overview

The mdiag -c command provides detailed information about the classes Moab is currently tracking. This command also allows an administrator to verify correct throttling policies and access provided to and from other credentials.

> The term class is used interchangeably with the term queue and generally refers to resource manager queue.

XML Attributes

Name	Description
ADEF	Accounts a class has access to.
CAPACITY	Number of procs available to the class.
DEFAULT.ATTR	Default attributes attached to a job.
DEFAULT.DISK	Default required disk attached to a job.
DEFAULT.FEATURES	Default required node features attached to a job.
DEFAULT.GRES	Default generic resources attached to a job.
DEFAULT.MEM	Default required memory attached to a job.
DEFAULT.NODESET	Default specified nodeset attached to a job.
DEFAULT.WCLIMIT	Default wallclock limit attached to a job.
EXCL.FEATURES	List of excluded (disallowed) node features.
EXCL.FLAGS	List of excluded (disallowed) job flags.
FSTARGET	The class' fairshare target.
HOLD	If TRUE this credential has a hold on it, FALSE otherwise.
HOSTLIST	The list of hosts in this class.

Name	Description
JOBEPILOG	Scheduler level job epilog to be run after job is completed by resource manager (script path).
JOBFLAGS	Default flags attached to jobs in the class.
JOBPROLOG	Scheduler level job prolog to be run before job is started by resource manager (script path).
ID	The unique ID of this class.
LOGLEVEL	The log level attached to jobs in the class.
MAX.PROC	The max processors per job in the class.
MAX.PS	The max processor-seconds per job in the class.
MAX.WCLIMIT	The max wallclock limit per job in the class.
MAXIJOB	The max idle jobs in the class.
MAXIPROC	The max idle processors in the class.
MAXJOBPERUSER	The max jobs per user.
MAXNODEPERJOB	The max nodes per job.
MAXNODEPERUSER	The max nodes per user.
MAXPROCPERJOB	The max processors per job.
MAXPROCPERNODE	The max processors per node.
MAXPROCPERUSER	The max processors per user.
MIN.NODE	The minimum nodes per job in the class.
MIN.PROC	The minimum processors per job in the class.

Name	Description
MIN.WCLIMIT	The minimum wallclock limit per job in the class.
NODEACCESSPOLICY	The node access policy associated with jobs in the class.
OCDPROCFACTOR	Dedicated processor factor.
OCNODE	Overcommit node.
PRIORITY	The class' associated priority.
PRIORITYF	Priority calculation function.
REQ.FEATURES	Required features for a job to be considered in the class.
REQ.FLAGS	Required flags for a job to be considered in the class.
REQ.IMAGE	Required image for a job to be considered in the class.
REQUIREDUSERLIST	The list of users who have access to the class.
RM	The resource manager reporting the class.
STATE	The class' state.
WCOVERRUN	Tolerated amount of time beyond the specified wallclock limit.

Example 3-20: Generating information about classes

```
> mdiag -c
Class/Queue Status
ClassID        Priority Flags        QDef               QOSList* PartitionList
Target Limits
DEFAULT               0 ---          ---                    ---  ---
0.00  ---
batch                 1 ---          ---                    ---  [A][B]
70.00  MAXJOB=33:200,250
  MAX.WCLIMIT=10:00:00  MAXPROCPERJOB=128
long                  1 ---          low                    low  [A]
10.00  MAXJOB=3:100,200
  MAX.WCLIMIT=1:00:00:00  MAXPROCPERJOB=128
fast                100 ---          high                  high  [B]
10.00  MAXJOB=8:100,150
  MAX.WCLIMIT=00:30:00  MAXPROCPERJOB=128
bigmem                1 ---          low,high               low  ---
10.00  MAXJOB=1:100,200
  MAXPROCPERJOB=128
```

*In the example above, class fast has **MAXJOB** soft and hard limits of 100 and 150 respectively and is currently running 8 jobs.*

> *The `Limits` column will display limits in the following format:*
>
> *<USAGE>:<HARDLIMIT>[,<SOFTLIMIT>]*

Related topics

- showstats command - display general statistics

mdiag -f

Synopsis

mdiag -f [-o user|group|acct|qos|class] [--flags=relative] [-w par=*<PARTITIONID>*]

Overview

The mdiag -f command is used to display at a glance information about the fairshare configuration and historic resource utilization. The fairshare usage may impact job prioritization, job eligibility, or both based on the credential **FSTARGET** and **FSCAP** attributes and by the fairshare priority weights as described in the Job Prioritization Overview. The information presented by this command includes fairshare configuration and credential fairshare usage over time.

The command hides information about credentials which have no fairshare target and no fairshare cap.

If an object type (*<OTYPE>*) is specified, then only information for that credential type (user, group, acct, class, or qos) will be displayed. If the `relative` flag is set, then per user fairshare usage will be displayed relative to each non-user credential (see the second example below).

> Relative output is only displayed for credentials which have user mappings. For example, if there is no association between classes and users, no relative per user fairshare usage class breakdown will be provided.

Example 3-21: Standard Fairshare Output

```
> mdiag -f
FairShare Information
Depth: 6 intervals   Interval Length: 00:20:00   Decay Rate: 0.50
FS Policy: DEDICATEDPES
System FS Settings:  Target Usage: 0.00
FSInterval         %     Target       0       1       2       3       4       5
FSWeight       ------- -------  1.0000  0.5000  0.2500  0.1250  0.0625  0.0312
TotalUsage      100.00 -------    85.3   476.1   478.9   478.5   475.5   482.8
USER
-------------
mattp             2.51 -------    2.20    2.69    2.21    2.65    2.65    3.01
jsmith           12.82 -------   12.66   15.36   10.96    8.74    8.15   13.85
kyliem            3.44 -------    3.93    2.78    4.36    3.11    3.94    4.25
tgh               4.94 -------    4.44    5.12    5.52    3.95    4.66    4.76
walex             1.51 -------    3.14    1.15    1.05    1.61    1.22    1.60
jimf              4.73 -------    4.67    4.31    5.67    4.49    4.93    4.92
poy               4.64 -------    4.43    4.61    4.58    4.76    5.36    4.90
mjackson          0.66 -------    0.35    0.78    0.67    0.77    0.55    0.43
tfw              17.44 -------   16.45   15.59   19.93   19.72   21.38   15.68
gjohn             2.81 -------    1.66    3.00    3.16    3.06    2.41    3.33
ljill            10.85 -------   18.09    7.23   13.28    9.24   14.76    6.67
kbill            11.10 -------    7.31   14.94    4.70   15.49    5.42   16.61
stevei            1.58 -------    1.41    1.34    2.09    0.75    3.30    2.15
gms               1.54 -------    1.15    1.74    1.63    1.40    1.38    0.90
patw              5.11 -------    5.22    5.11    4.85    5.20    5.28    5.78
wer               6.65 -------    5.04    7.03    7.52    6.80    6.43    2.83
anna              1.97 -------    2.29    1.68    2.27    1.80    2.37    2.17
susieb            5.69 -------    5.58    5.55    5.57    6.48    5.83    6.16
GROUP
-------------
dallas           13.25  15.00    14.61   12.41   13.19   13.29   15.37   15.09
sanjose*          8.86  15.00     6.54    9.55    9.81    8.97    8.35    4.16
seattle          10.05  15.00     9.66   10.23   10.37    9.15    9.94   10.54
austin*          30.26  15.00    29.10   30.95   30.89   28.45   29.53   29.54
boston*           3.44  15.00     3.93    2.78    4.36    3.11    3.94    4.25
orlando*         26.59  15.00    29.83   26.77   22.56   29.49   25.53   28.18
newyork*          7.54  15.00     6.33    7.31    8.83    7.54    7.34    8.24
ACCT
-------------
engineering      31.76  30.00    32.25   32.10   31.94   30.07   30.74   31.14
marketing         8.86   5.00     6.54    9.55    9.81    8.97    8.35    4.16
it                9.12   5.00     7.74    8.65   10.92    8.29   10.64   10.40
development*     24.86  30.00    24.15   24.76   25.00   24.84   26.15   26.78
research         25.40  30.00    29.32   24.94   22.33   27.84   24.11   27.53
QOS
-------------
DEFAULT*          0.00  50.00  ------- ------- ------- ------- ------- -------
high*            83.69  90.00    86.76   83.20   81.71   84.35   83.19   88.02
urgent            0.00   5.00  ------- ------- ------- ------- ------- -------
low*             12.00   5.00     7.34   12.70   14.02   12.51   12.86    7.48
CLASS
-------------
batch*           51.69  70.00    53.87   52.01   50.80   50.38   48.67   52.65
long*            18.75  10.00    16.54   18.36   20.89   18.36   21.53   16.28
fast*            15.29  10.00    18.41   14.98   12.58   16.80   15.15   18.21
bigmem           14.27  10.00    11.17   14.65   15.73   14.46   14.65   12.87
```

An asterisk (*) next to a credential name indicates that that credential has exceeded its fairshare target.

Example 3-22: Grouping User Output by Account

```
> mdiag -f -o acct --flags=relative
FairShare Information
Depth: 6 intervals   Interval Length: 00:20:00   Decay Rate: 0.50
FS Policy: DEDICATEDPES
System FS Settings:  Target Usage: 0.00
FSInterval        %     Target       0       1       2       3       4       5
FSWeight       ------- -------  1.0000  0.5000  0.2500  0.1250  0.0625  0.0312
TotalUsage      100.00 -------    23.8   476.1   478.9   478.5   475.5   482.8
ACCOUNT
-------------
dallas           13.12  15.00    15.42   12.41   13.19   13.29   15.37   15.09
  mattp          19.47 -------   15.00   21.66   16.75   19.93   17.26   19.95
  walex           9.93 -------   20.91    9.28    7.97   12.14    7.91   10.59
  stevei         12.19 -------    9.09   10.78   15.85    5.64   21.46   14.28
  anna           14.77 -------   16.36   13.54   17.18   13.55   15.44   14.37
  susieb         43.64 -------   38.64   44.74   42.25   48.74   37.92   40.81
sanjose*          9.26  15.00     8.69    9.55    9.81    8.97    8.35    4.16
  mjackson        7.71 -------    6.45    8.14    6.81    8.62    6.54   10.29
  gms            17.61 -------   21.77   18.25   16.57   15.58   16.51   21.74
  wer            74.68 -------   71.77   73.61   76.62   75.80   76.95   67.97
seattle          10.12  15.00    10.16   10.23   10.37    9.15    9.94   10.54
  tgh            49.56 -------   46.21   50.05   53.26   43.14   46.91   45.13
  patw           50.44 -------   53.79   49.95   46.74   56.86   53.09   54.87
austin*          30.23  15.00    25.58   30.95   30.89   28.45   29.53   29.54
  jsmith         42.44 -------   48.77   49.62   35.47   30.70   27.59   46.90
  tfw            57.56 -------   51.23   50.38   64.53   69.30   72.41   53.10
boston*           3.38  15.00     3.78    2.78    4.36    3.11    3.94    4.25
  kyliem        100.00 -------  100.00  100.00  100.00  100.00  100.00  100.00
orlando*         26.20  15.00    30.13   26.77   22.56   29.49   25.53   28.18
  poy            17.90 -------   16.28   17.22   20.30   16.15   20.98   17.39
  ljill          37.85 -------   58.60   26.99   58.87   31.33   57.79   23.67
  kbill          44.25 -------   25.12   55.79   20.83   52.52   21.23   58.94
newyork*          7.69  15.00     6.24    7.31    8.83    7.54    7.34    8.24
  jimf           61.42 -------   69.66   58.94   64.20   59.46   67.21   59.64
  gjohn          38.58 -------   30.34   41.06   35.80   40.54   32.79   40.36
```

Related topics

- Fairshare Overview

mdiag -g

Synopsis

mdiag -g [*groupid*]

Overview

The mdiag -g command is used to present information about groups.

mdiag -j

Synopsis

mdiag -j [*jobid*] [-t <*partition*>] [-v] [-w] [--flags=policy] [--xml] [--blocking]

Overview

The mdiag -j command provides detailed information about the state of jobs Moab is currently tracking. This command also performs a large number of sanity and state checks. The job configuration and status information, as well as the results of the various checks, are presented by this command. The command gathers information from the Moab cache which prevents it from interrupting the scheduler, but the `--blocking` option can be used to bypass the cache and interrupt the scheduler. If the `-v` (verbose) flag is specified, additional information about less common job attributes is displayed. If `--flags=policy` is specified, information about job templates is displayed.

If used with the `-t <partition>` option on a running job, the only thing mdiag -j shows is if the job is running on the specified partition. If used on job that is not running, it shows if the job is able to run on the specified partition.

The `-w` flag enables you to specify specific job states (Such as Running, Completed, Idle, or ALL. See **Job States** on page 107 for all valid options.) or jobs associated with a given credential (user, acct, class, group, qos). For example:

```
mdiag -j -w user=david              # Displays only David's jobs
mdiag -j -w state=Idle,Running      # Displays only idle or running jobs
```

> The mdiag -j command does not show all subjobs of an array unless you use `mdiag -j --xml`. In the XML, the master job element contains a child element called `ArraySubJobs` that contains the subjobs in the array. Using `mdiag -j -v --xml` shows the completed sub-jobs as well.

XML Output

If XML output is requested (via the --format=xml argument), XML based node information will be written to STDOUT in the following format:

```
<Data>
  <job ATTR="VALUE" ... > </job>
  ...
</Data>
```

For information about legal attributes, refer to the XML Attributes table.

> To show jobs in XML, use `mdiag -j --xml -w [completed=true|system=true|ALL=true]` to limit or filter jobs. This is for XML use only.

Related topics

- checkjob
- mdiag

mdiag -n

Synopsis

mdiag -n [-t *partitionid*] [-A *creds*] [-w <CONSTRAINT>] [-v] [--format=xml] [*nodeid*]

Overview

The mdiag -n command provides detailed information about the state of nodes Moab is currently tracking. This command also performs a large number of sanity and state checks. The node configuration and status information as well as the results of the various checks are presented by this command.

Arguments

Flag	Argument	Description
[-A]	{user|group|account|qos|class|job}: `<OBJECTID>`	report if each node is accessible by requested job or credential
[-t]	`<partitionid>`	report only nodes from specified partition
[-v]	---	show verbose output (do not truncate columns and add columns for additional node attributes)
[-w]	`nodestate=drained`	display only jobs associated with the specified constraint: `nodestate` (See DISPLAYFLAGS for more information.)

Output

This command presents detailed node information in whitespace-delineated fields.

The output of this command can be extensive and the values for a number of fields may be truncated. If truncated, the `-v` flag can be used to display full field content.

Column	Format
Name	*<NODE NAME>*
State	*<NODE STATE>*
Procs	*<AVAILABLE PROCS>*:*<CONFIGURED PROCS>*
Memory	*<AVAILABLE MEMORY>*:*<CONFIGURED MEMORY>*
Disk	*<AVAILABLE DISK>*:*<CONFIGURED DISK>*
Swap	*<AVAILABLE SWAP>*:*<CONFIGURED SWAP>*
Speed	*<RELATIVE MACHINE SPEED>*

Column	Format
Opsys	*<NODE OPERATING SYSTEM>*
Arch	*<NODE HARDWARE ARCHITECTURE>*
Par	*<PARTITION NODE IS ASSIGNED TO>*
Load	*<CURRENT 1 MINUTE BSD LOAD>*
Rsv	*<NUMBER OF RESERVATIONS ON NODE>*
Classes	*<CLASS NAME>*
Network	*<NETWORK NAME>...*
Features	*<NODE FEATURE>...*

Examples

Example 3-23:

```
> mdiag -n

compute node summary
Name                    State   Procs      Memory         Opsys

opt-001                  Busy    0:2      2048:2048        SuSE
opt-002                  Busy    0:2      2048:2048        SuSE
opt-003                  Busy    0:2      2048:2048        SuSE
opt-004                  Busy    0:2      2048:2048        SuSE
opt-005                  Busy    0:2      2048:2048        SuSE
opt-006                  Busy    0:2      2048:2048        SuSE
WARNING:   swap is low on node opt-006
opt-007                  Busy    0:2      2048:2048        SuSE
opt-008                  Busy    0:2      2048:2048        SuSE
opt-009                  Busy    0:2      2048:2048        SuSE
opt-010                  Busy    0:2      2048:2048        SuSE
opt-011                  Busy    0:2      2048:2048        SuSE
opt-012                  Busy    0:2      2048:2048        SuSE
opt-013                  Busy    0:2      2048:2048        SuSE
opt-014                  Busy    0:2      2048:2048        SuSE
opt-015                  Busy    0:2      2048:2048        SuSE
opt-016                  Busy    0:2      2048:2048        SuSE
x86-001                  Busy    0:1       512:512       Redhat
x86-002                  Busy    0:1       512:512       Redhat
x86-003                  Busy    0:1       512:512       Redhat
x86-004                  Busy    0:1       512:512       Redhat
x86-005                  Idle    1:1       512:512       Redhat
x86-006                  Idle    1:1       512:512       Redhat
x86-007                  Idle    1:1       512:512       Redhat
x86-008                  Busy    0:1       512:512       Redhat
x86-009                  Down    1:1       512:512       Redhat
x86-010                  Busy    0:1       512:512       Redhat
x86-011                  Busy    0:1       512:512       Redhat
x86-012                  Busy    0:1       512:512       Redhat
x86-013                  Busy    0:1       512:512       Redhat
x86-014                  Busy    0:1       512:512       Redhat
x86-015                  Busy    0:1       512:512       Redhat
x86-016                  Busy    0:1       512:512       Redhat
P690-001                 Busy    0:1     16384:16384        AIX
P690-002                 Busy    0:1     16384:16384        AIX
P690-003                 Busy    0:1     16384:16384        AIX
P690-004                 Busy    0:1     16384:16384        AIX
P690-005                 Busy    0:1     16384:16384        AIX
P690-006                 Busy    0:1     16384:16384        AIX
P690-007                 Idle    1:1     16384:16384        AIX
P690-008                 Idle    1:1     16384:16384        AIX
WARNING:   node P690-008 is missing ethernet adapter
P690-009                 Busy    0:1     16384:16384        AIX
P690-010                 Busy    0:1     16384:16384        AIX
P690-011                 Busy    0:1     16384:16384        AIX
P690-012                 Busy    0:1     16384:16384        AIX
P690-013                 Busy    0:1     16384:16384        AIX
P690-014                 Busy    0:1     16384:16384        AIX
P690-015                 Busy    0:1     16384:16384        AIX
P690-016                 Busy    0:1     16384:16384        AIX
-----                     ---    6:64   745472:745472     -----

Total Nodes: 36  (Active: 30  Idle: 5  Down: 1)
```

> Warning messages are interspersed with the node configuration information with all warnings preceded by the keyword WARNING.

XML Output

If XML output is requested (via the --format=xml argument), XML based node information will be written to STDOUT in the following format:

```
mdiag -n --format=xml
<Data>
  <node> <ATTR>="<VAL>" ... </node>
  ...
</Data>
```

XML Attributes

Name	Description
AGRES	Available generic resources
ALLOCRES	Special allocated resources (like vlans)
ARCH	The node's processor architecture.
AVLCLASS	Classes available on the node.
AVLETIME	Time when the node will no longer be available (used in Utility centers)
AVLSTIME	Time when the node will be available (used in Utility centers)
CFGCLASS	Classes configured on the node
ENABLEPROFILING	If true, a node's state and usage is tracked over time.
FEATURES	A list of comma-separated custom features describing a node.
GEVENT	A user-defined event that allows Moab to perform some action.
GMETRIC	A list of comma-separated consumable resources associated with a node.
GRES	generic resources on the node

Name	Description
HOPCOUNT	How many hops the node took to reach this Moab (used in hierarchical grids)
ISDELETED	Node has been deleted
ISDYNAMIC	Node is dynamic (used in Utility centers)
JOBLIST	The list of jobs currently running on a node.
LOAD	Current load as reported by the resource manager
LOADWEIGHT	Load weight used when calculating node priority
MAXJOB	See Node Policies for details.
MAXJOBPERUSER	See Node Policies for details.
MAXLOAD	See Node Policies for details.
MAXPROC	See Node Policies for details.
MAXPROCPERUSER	See Node Policies for details.
NETWORK	The ability to specify which networks are available to a given node is limited to only a few resource managers. Using the **NETWORK** attribute, administrators can establish this node to network connection directly through the scheduler. The NODECFG parameter allows this list to be specified in a comma-delimited list.
NODEID	The unique identifier for a node.
NODESTATE	The state of a node.
OS	A node's operating system.
OSLIST	Operating systems the node can run
OSMODACTION	URL for changing the operating system
OWNER	Credential type and name of owner

Name	Description
PARTITION	The partition a node belongs to. See Node Location for details.
POWER	The state of the node's power. Either ON or OFF.
PRIORITY	The fixed node priority relative to other nodes.
PROCSPEED	A node's processor speed information specified in MHz.
RACK	The rack associated with a node's physical location.
RADISK	The total available disk on a node.
RAMEM	The total available memory available on a node.
RAPROC	The total number of processors available on a node.
RASWAP	The total available swap on a node.
RCMEM	The total configured memory on a node.
RCPROC	The total configured processors on a node.
RCSWAP	The total configured swap on a node.
RESCOUNT	Number of reservations on the node
RSVLIST	List of reservations on the node
RESOURCES	Deprecated (use GRES)
RMACCESSLIST	A comma-separated list of resource managers who have access to a node.
SIZE	The number of slots or size units consumed by the node.
SLOT	The first slot in the rack associated with the node's physical location.
SPEED	A node's relative speed.
SPEEDWEIGHT	speed weight used to calculate node's priority

Name	Description
STATACTIVETIME	Time node was active
STATMODIFYTIME	Time node's state was modified
STATTOTALTIME	Time node has been monitored
STATUPTIME	Time node has been up
TASKCOUNT	The number of tasks on a node.

Related topics

- checknode

mdiag -t

Synopsis

mdiag -t [-v] [-v] [*partitionid*]

Overview

The mdiag -t command is used to present configuration, usage, health, and diagnostic information about partitions maintained by Moab. The information presented includes partition name, limits, configured and available resources, allocation weights and policies.

Examples

Example 3-24: Standard partition diagnostics

```
> mdiag -t
Partition Status
...
```

mdiag -p

Synopsis

mdiag -p [-t *partition*] [-v]

Overview

The mdiag -p command is used to display at a glance information about the job priority configuration and its effects on the current eligible jobs. The information presented by this command includes priority weights, priority components, and the percentage contribution of each component to the total job priority.

The command hides information about priority components which have been deactivated (i.e. by setting the corresponding component priority weight to 0). For each displayed priority component, this command gives a small amount of context sensitive information. The following table documents this information. In all cases, the output is of the form *<PERCENT>(<CONTEXT INFO>)* where *<PERCENT>* is the percentage contribution of the associated priority component to the job's total priority.

> By default, this command only shows information for jobs which are eligible for immediate execution. Jobs which violate soft or hard policies, or have holds, job dependencies, or other job constraints in place will not be displayed. If priority information is needed for any of these jobs, use the -v flag or the checkjob command.

Format

Flag	Name	Format	Default	Description	Example
-v	VERBOSE	---	---	Display verbose priority information. If specified, display priority breakdown information for blocked, eligible, and active jobs. By default, only information for eligible jobs is displayed. To view blocked jobs in addition to eligible, run mdiag -p -v -v.	`> mdiag -p -v` *Display priority summary information for eligible and active jobs*

Output

Priority Component	Format	Description
Target	*<PERCENT>*()	
QOS	*<PERCENT>*(*<QOS>*:*<QOSPRI>*)	*QOS* — QOS associated with job *QOSPRI* — Priority assigned to the QOS

Priority Component	Format	Description
FairShare	*<PERCENT>* (*<USR>*:*<GRP>*:*<ACC>*:*<QOS>*:*<CLS>*)	*USR* — user fs usage - user fs target *GRP* — group fs usage - group fs target *ACC* — account fs usage - account fs target *QOS* — QOS fs usage - QOS fs target *CLS* — class fs usage - class fs target
Service	*<PERCENT>*(*<QT>*:*<XF>*:*<Byp>*)	*QTime* — job queue time which is applicable towards priority (in minutes) *XF* — current theoretical minimum XFactor is job were to start immediately *Byp* — number of times job was bypassed by lower priority jobs via backfill
Resource	*<PERCENT>* (*<NDE>*:*<PE>*:*<PRC>*:*<MEM>*)	*NDE* — nodes requested by job *PE* — Processor Equivalents as calculated by all resources requested by job *PRC* — processors requested by job *MEM* — real memory requested by job

Examples

Example 3-25: mdiag -p

```
diagnosing job priority information (partition: ALL)

Job                    PRIORITY*   Cred(  QOS)    FS(Accnt)  Serv(QTime)
Weights   --------       1(    1)      1(    1)     1(    1)

13678                      1321*    7.6(100.0)   0.2(  2.7)  92.2(1218.)
13698                       235*   42.6(100.0)   1.1(  2.7)  56.3(132.3)
13019                      8699     0.6( 50.0)   0.3( 25.4)  99.1(8674.)
13030                      8699     0.6( 50.0)   0.3( 25.4)  99.1(8674.)
13099                      8537     0.6( 50.0)   0.3( 25.4)  99.1(8512.)
13141                      8438     0.6( 50.0)   0.2( 17.6)  99.2(8370.)
13146                      8428     0.6( 50.0)   0.2( 17.6)  99.2(8360.)
13153                      8360     0.0(  1.0)   0.1( 11.6)  99.8(8347.)
13177                      8216     0.0(  1.0)   0.1( 11.6)  99.8(8203.)
13203                      8127     0.6( 50.0)   0.3( 25.4)  99.1(8102.)
13211                      8098     0.0(  1.0)   0.1( 11.6)  99.8(8085.)
...
13703                       137    36.6( 50.0)  12.8( 17.6)  50.6( 69.2)
13702                        79     1.3(  1.0)   5.7(  4.5)  93.0( 73.4)

Percent Contribution   --------     0.9(  0.9)   0.4(  0.4)  98.7( 98.7)

* indicates system prio set on job
```

The mdiag -p command only displays information for priority components actually utilized. In the above example, QOS, Account Fairshare, and QueueTime components are utilized in determining a job's priority. Other components, such as Service Targets, and Bypass are not used and thus are not displayed. (See the Priority Overview for more information) The output consists of a header, a job by job analysis of jobs, and a summary section.

The header provides column labeling and provides configured priority component and subcomponent weights. In the above example, **QOSWEIGHT** *is set to 1000 and* **FSWEIGHT** *is set to 100. When configuring fairshare, a site also has the option of weighting the individual components of a job's overall fairshare, including its user, group, and account fairshare components. In this output, the QoS and account fairshare weights are set to 1.*

The job by job analysis displays a job's total priority and the percentage contribution to that priority of each of the priority components. In this example, job `13019` has a total priority of 8699. Both QOS and Fairshare contribute to the job's total priority although these factors are quite small, contributing `0.6%` and `0.3%` respectively with the fairshare factor being contributed by an account fairshare target. For this job, the dominant factor is the `service` subcomponent `qtime` which is contributing `99.1%` of the total priority since the job has been in the queue for approximately 8600 minutes.

At the end of the job by job description, a Totals line is displayed which documents the average percentage contributions of each priority component to the current idle jobs. In this example, the QOS, Fairshare, and Service components contributed an average of `0.9%`, `0.4%`, and `98.7%` to the jobs' total priorities.

Related topics

- Job Priority Overview
- Moab Cluster Manager - Priority Manager

mdiag -q

Synopsis

mdiag -q [*qosid*]

Overview

The mdiag -q command is used to present information about each QOS maintained by Moab. The information presented includes QOS name, membership, scheduling priority, weights and flags.

Examples

Example 3-26: Standard QOS Diagnostics

```
> mdiag -q
QOS Status
System QOS Settings:  QList: DEFAULT (Def: DEFAULT)  Flags: 0
Name                * Priority QTWeight QTTarget XFWeight XFTarget     QFlags
JobFlags Limits
DEFAULT                    1        1        3        1     5.00   PREEMPTEE
[NONE] [NONE]
  Accounts:  it research
  Classes:  batch
[ALL]                      0        0        0        0     0.00      [NONE]
[NONE] [NONE]
high                    1000        1        2        1    10.00   PREEMPTOR
[NONE] [NONE]
  Accounts:  engineering it development research
  Classes:  fast
urgent                 10000        1        1        1     7.00   PREEMPTOR
[NONE] [NONE]
  Accounts:  engineering it development
low                      100        1        5        1     1.00   PREEMPTEE
[NONE] [NONE]
  Accounts:  engineering marketing it development research
  Classes:  long bigmem
```

mdiag -r

Synopsis

mdiag -r [*reservationid*] [-v] [-w type=<*type*>]

Overview

The mdiag -r command allows administrators to look at detailed reservation information. It provides the name, type, partition, starttime and endtime, proc and node counts, as well as actual utilization figures. It also provides detailed information about which resources are being used, how many nodes, how much memory, swap, and processors are being associated with each task. Administrators can also view the Access Control Lists for each reservation as well as any flags that may be active in the reservation. The command gathers information from the Moab cache which prevents it from waiting for the scheduler, but the `--blocking` option can be used to bypass the cache and allow waiting for the scheduler.

The `-w` flag filters the output according to the type of reservation. The allowable reservation types are Job, and User.

Examples

Example 3-27:

```
> mdiag -r
Diagnosing Reservations
RsvID                      Type Par   StartTime     EndTime     Duration Node Task
Proc
-----                      ---- ---   ---------     -------     -------- ---- ---- ---
-
engineer.0.1               User   A    -6:29:00    INFINITY     INFINITY    0    0
7
    Flags: STANDINGRSV IGNSTATE OWNERPREEMPT
    ACL:   CLASS==batch+:==long+:==fast+:==bigmem+ QOS==low-:==high+ JATTR==PREEMPTEE+
    CL:    RSV==engineer.0.1
    Task Resources: PROCS: [ALL]
    Attributes (HostExp='fr10n01 fr10n03 fr10n05 fr10n07 fr10n09 fr10n11 fr10n13
fr10n15')
    Active PH: 43.77/45.44 (96.31%)
    SRAttributes (TaskCount: 0  StartTime: 00:00:00  EndTime: 1:00:00:00  Days: ALL)
research.0.2               User   A    -6:29:00    INFINITY     INFINITY    0    0
8
    Flags: STANDINGRSV IGNSTATE OWNERPREEMPT
    ACL:   CLASS==batch+:==long+:==fast+:==bigmem+ QOS==high+:==low- JATTR==PREEMPTEE+
    CL:    RSV==research.0.2
    Task Resources: PROCS: [ALL]
    Attributes (HostExp='fr3n01 fr3n03 fr3n05 fr3n07 fr3n07 fr3n09 fr3n11 fr3n13
fr3n15')
    Active PH: 51.60/51.93 (99.36%)
    SRAttributes (TaskCount: 0  StartTime: 00:00:00  EndTime: 1:00:00:00  Days: ALL)
fast.0.3                   User   A    00:14:05     5:14:05      5:00:00    0    0
16
    Flags: STANDINGRSV IGNSTATE OWNERPREEMPT
    ACL:   CLASS==fast+ QOS==high+:==low+:==urgent+:==DEFAULT+ JATTR==PREEMPTEE+
    CL:    RSV==fast.0.3
    Task Resources: PROCS: [ALL]
    Attributes (HostExp='fr12n01 fr12n02 fr12n03 fr12n04 fr12n05 fr12n06 fr12n07
fr12n08 fr12n09 fr12n10 fr12n11 fr12n12 fr12n13 fr12n14 fr12n15 fr12n16')
    SRAttributes (TaskCount: 0  StartTime: 00:00:00  EndTime: 5:00:00  Days:
Mon,Tue,Wed,Thu,Fri)
fast.1.4                   User   A  1:00:14:05  1:05:14:05      5:00:00    0    0
16
    Flags: STANDINGRSV IGNSTATE OWNERPREEMPT
    ACL:   CLASS==fast+ QOS==high+:==low+:==urgent+:==DEFAULT+ JATTR==PREEMPTEE+
    CL:    RSV==fast.1.4
    Task Resources: PROCS: [ALL]
    Attributes (HostExp='fr12n01 fr12n02 fr12n03 fr12n04 fr12n05 fr12n06 fr12n07
fr12n08 fr12n09 fr12n10 fr12n11 fr12n12 fr12n13 fr12n14 fr12n15 fr12n16')
    SRAttributes (TaskCount: 0  StartTime: 00:00:00  EndTime: 5:00:00  Days:
Mon,Tue,Wed,Thu,Fri)
job2411                     Job   A   -00:01:00    00:06:30     Each tile contains a
summary information about the service it represents, including the following:
    ACL:   JOB==job2411=
    CL:    JOB==job2411 USER==jimf GROUP==newyork ACCT==it CLASS==bigmem QOS==low
JATTR==PREEMPTEE DURATION==00:07:30 PROC==6 PS==2700
job1292                     Job   A    00:00:00    00:07:30     00:07:30    0    0
4
    ACL:   JOB==job1292=
    CL:    JOB==job1292 USER==jimf GROUP==newyork ACCT==it CLASS==batch QOS==DEFAULT
JATTR==PREEMPTEE DURATION==00:07:30 PROC==4 PS==1800
```

Example 3-28:

With the `-v` option, a nodes line is included for each reservation and shows how many nodes are in the reservation as well as how many tasks are on each node.

```
> mdiag -r -v
Diagnosing Reservations
RsvID                      Type Par   StartTime     EndTime     Duration Node Task
Proc
-----                      ---- ---   ---------     -------     -------- ---- ---- ---
-
Moab.6                      Job   B   -00:01:05    00:00:35     00:01:40    1    1
1
    Flags: ISACTIVE
    ACL:   JOB==Moab.6=
    CL:    JOB==Moab.6 USER==tuser1 GROUP==tgroup1 CLASS==fast QOS==starter
JPRIORITY<=0 DURATION==00:01:40 PROC==1 PS==100
    SubType: JobReservation
    Nodes='node002:1'
    Rsv-Group: Moab.6

Moab.4                      Job   B   -00:01:05    00:00:35     00:01:40    1    1
1
    Flags: ISACTIVE
    ACL:   JOB==Moab.4=
    CL:    JOB==Moab.4 USER==tuser1 GROUP==tgroup1 CLASS==batch QOS==starter
JPRIORITY<=0 DURATION==00:01:40 PROC==1 PS==100
    SubType: JobReservation
    Nodes='node002:1'
    Rsv-Group: Moab.4

Moab.5                      Job   A   -00:01:05    00:00:35     00:01:40    3    3
6
    Flags: ISACTIVE
    ACL:   JOB==Moab.5=
    CL:    JOB==Moab.5 USER==tuser1 GROUP==tgroup1 ACCT==marketing CLASS==long
QOS==low JPRIORITY<=0 DURATION==00:01:40 PROC==6 PS==600
    Task Resources: PROCS: [ALL]
    SubType: JobReservation
    Nodes='node008:1,node007:1,node006:1'
    Rsv-Group: Moab.5

Moab.7                      Job   A   -00:01:04    00:00:36     00:01:40    1    1
1
    Flags: ISACTIVE
    ACL:   JOB==Moab.7=
    CL:    JOB==Moab.7 USER==tuser1 GROUP==tgroup1 CLASS==bigmen QOS==starter
JPRIORITY<=0 DURATION==00:01:40 PROC==1 PS==100
    SubType: JobReservation
    Nodes='node005:1'
    Rsv-Group: Moab.7

Moab.2                      Job   A   -00:01:07     3:58:53      4:00:00    1    2
2
    Flags: ISACTIVE
    ACL:   JOB==Moab.2=
    CL:    JOB==Moab.2 USER==tuser1 GROUP==tgroup1 QOS==starter JPRIORITY<=0
DURATION==4:00:00 PROC==2 PS==28800
    SubType: JobReservation
    Nodes='node009:1'
    Rsv-Group: Moab.2

Moab.8                      Job   A     3:58:53     7:58:53      4:00:00    8   16
16
    Flags: PREEMPTEE
    ACL:   JOB==Moab.8=
    CL:    JOB==Moab.8 USER==tuser1 GROUP==tgroup1 ACCT==development CLASS==bigmen
```

```
QOS==starter JPRIORITY<=0 DURATION==4:00:00 PROC==16 PS==230400
    SubType: JobReservation

Nodes='node009:1,node008:1,node007:1,node006:1,node005:1,node004:1,node003:1,node001:
1'
    Attributes (Priority=148)
    Rsv-Group: idle

system.3                   User bas   -00:01:08    INFINITY     INFINITY    1    1
2
    Flags: ISCLOSED,ISACTIVE
    ACL:   RSV==system.3=
    CL:    RSV==system.3
    Accounting Creds:  User:root
    Task Resources: PROCS: [ALL]
    SubType: Other
    Nodes='node254:1'
    Attributes (HostExp='node254')
    Active PH: 0.00/0.01 (0.00%)
    History: 1322773208:PROCS=2


system.2                   User bas   -00:01:08    INFINITY     INFINITY    1    1
2
    Flags: ISCLOSED,ISACTIVE
    ACL:   RSV==system.2=
    CL:    RSV==system.2
    Accounting Creds:  User:root
    Task Resources: PROCS: [ALL]
    SubType: Other
    Nodes='node255:1'
    Attributes (HostExp='node255')
    Active PH: 0.00/0.01 (0.00%)
    History: 1322773208:PROCS=2


system.1                   User bas   -00:01:08    INFINITY     INFINITY    1    1
2
    Flags: ISCLOSED,ISACTIVE
    ACL:   RSV==system.1=
    CL:    RSV==system.1
    Accounting Creds:  User:root
    Task Resources: PROCS: [ALL]
    SubType: Other
    Nodes='node256:1'
    Attributes (HostExp='node256')
    Active PH: 0.00/0.01 (0.00%)
    History: 1322773208:PROCS=2
```

mdiag -S

Synopsis

mdiag -S [-v] [-v]

Overview

The mdiag -S command is used to present information about the status of the scheduler and grid interface.

This command will report on the following aspects of scheduling:

- General Scheduler Configuration
  - Reports short and long term scheduler load
  - Reports detected overflows of node, job, reservation, partition, and other scheduler object tables
- High Availability
  - Configuration
  - Reports health of HA primary
  - Reports health of HA backup
- Scheduling Status
  - Reports if scheduling is paused
  - Reports if scheduling is stopped
- System Reservation Status
  - Reports if global system reservation is active
- Message Profiling/Statistics Status
- Moab scheduling activities (only with mdiag -S -v -v)
  - Activity[JobStart]: Time Moab spends telling the RM to start a job and waiting for a response.
  - Activity[RMResourceLoad]: Time Moab spends querying license managers and nodes.
  - Activity[RMWorkloadLoad]: Time Moab spends querying resource managers about jobs (as opposed to nodes)
  - Activity[Schedule]: Time Moab spends prioritizing jobs and scheduling them onto nodes.
  - Activity[UIProcess]: Time Moab spends handling client commands.

Examples

Example 3-29:

```
> mdiag -S
Moab Server running on orion-1:43225  (Mode: NORMAL)
  Load(5m)  Sched: 12.27%  RMAction: 1.16%  RMQuery: 75.30%  User: 0.29%  Idle: 10.98%
  Load(24h) Sched: 10.14%  RMAction: 0.93%  RMQuery: 74.02%  User: 0.11%  Idle: 13.80%
  HA Fallback Server:  orion-2:43225  (Fallback is Ready)
  Note:  system reservation blocking all nodes
  Message:  profiling enabled (531 of 600 samples/5:00 interval)
```

mdiag -s

Synopsis

mdiag -s [*reservationid*] [-v]>]

Overview

The mdiag -s command allows administrators to look at detailed standing reservation information. It provides the name, type, partition, starttime and endtime, period, task count, host list, and a list of child instances.

Examples

Example 3-30:

```
> mdiag -s
standing reservation overview
RsvID                  Type    Par     StartTime        EndTime          Duration         Period
-----                  ----    ---     ---------        -------          --------         ------

TestSR                 User    ---     00:00:00             ---          00:00:00            DAY
  Days:                ALL
  Depth:       2
  RsvList:     testSR.1,testSR.2,testSR.3
  HostExp:     'node1,node2,node4,node8'

test2                  User    ---     00:00:00             ---          00:00:00            DAY
  Days:        ALL
  TaskCount:   4
  Depth:       1
  RsvList:     test2.4,test2.5
```

mdiag -T

Synopsis

mdiag -T [*triggerid*] [-v] [--blocking]

Overview

The mdiag -T command is used to present information about each Trigger. The information presented includes Name, State, Action, Event Time. The command gathers information from the Moab cache which prevents it from waiting for the scheduler, but the --blocking option can be used to bypass the cache and allow waiting for the scheduler.

Examples

Example 3-31:

```
> mdiag -T
TrigID                 Object ID                Event   AType            ActionDate
State
---------------------- -------------------- -------- ------ -------------------- ------
-----
sched_trig.0           sched:Moab                 end   exec                      -
Blocked
3                      node:node010         threshol   exec                      -
Blocked
5                      job:Moab.7            preempt   exec                      -
Blocked
6                      job:Moab.8            preempt   exec                      -
Blocked
4*                     job:Moab.5              start   exec            -00:00:36
Failure
* indicates trigger has completed
```

Example 3-32:

```
> mdiag -T -v
TrigID                Object ID                Event  AType           ActionDate
State
--------------------- -------------------- --------- ------ -------------------- ------
-----
sched_trig.0          sched:Moab                 end   exec                    -
Blocked
  Name:          sched_trig
  Flags:         globaltrig
  BlockUntil:    INFINITY  ActiveTime:  ---
  Action Data:   date
  NOTE:  trigger can launch

3                     node:node010         threshol   exec                    -
Blocked
  Flags:         globaltrig
  BlockUntil:    INFINITY  ActiveTime:  ---
  Threshold:     CPULoad > 3.00  (current value: 0.00)
  Action Data:   date
  NOTE:  trigger cannot launch - threshold not satisfied - threshold type not
supported

5                     job:Moab.7            preempt   exec                    -
Blocked
  Flags:         user,globaltrig
  BlockUntil:    INFINITY  ActiveTime:  ---
  Action Data:   $HOME/tools/preemptnotify.pl $OID $OWNER $HOSTNAME

6                     job:Moab.8            preempt   exec                    -
Blocked
  Flags:         user,globaltrig
  BlockUntil:    INFINITY  ActiveTime:  ---
  Action Data:   $HOME/tools/preemptnotify.pl $OID $OWNER $HOSTNAME
  NOTE:  trigger cannot launch - parent job Moab.8 is in state Idle

4*                    job:Moab.5              start   exec  Mon Jan 16 12:33:00
Failure
  Launch Time:   -00:02:17
  Flags:         globaltrig
  Last Execution State: Failure (ExitCode: 0)
  BlockUntil:    00:00:00  ActiveTime:  00:00:00
  Action Data:   $HOME/tools/preemptnotify.pl $OID $OWNER $HOSTNAME
  ALERT:  trigger failure detected
  Message:       'exec '/usr/test/moab/tools/preemptnotify.pl' cannot be located or is
not executable'

* indicates trigger has completed
```

mdiag -u

Synopsis

mdiag -u [*userid*]

Overview

The mdiag -u command is used to present information about user records maintained by Moab. The information presented includes user name, UID, scheduling priority, default job flags, default QOS level, List of accessible QOS levels, and list of accessible partitions.

Examples

Example 3-33:

```
> mdiag -u
evaluating user information
Name          Priority        Flags         QDef      QOSList*        PartitionList
Target  Limits

jvella              0       [NONE]       [NONE]       [NONE]                [NONE]
0.00   [NONE]
  ALIST=Engineering
  Message:  profiling enabled (597 of 3000 samples/00:15:00 interval)
[NONE]              0       [NONE]       [NONE]       [NONE]                [NONE]
0.00   [NONE]
reynolds            0       [NONE]       [NONE]       [NONE]                [NONE]
0.00   [NONE]
  ALIST=Administration
  Message:  profiling enabled (597 of 3000 samples/00:15:00 interval)
mshaw               0       [NONE]       [NONE]       [NONE]                [NONE]
0.00   [NONE]
  ALIST=Test
  Message:  profiling enabled (584 of 3000 samples/00:15:00 interval)
kforbes             0       [NONE]       [NONE]       [NONE]                [NONE]
0.00   [NONE]
  ALIST=Shared
  Message:  profiling enabled (597 of 3000 samples/00:15:00 interval)
gastor              0       [NONE]       [NONE]       [NONE]                [NONE]
0.00   [NONE]
  ALIST=Engineering
  Message:  profiling enabled (597 of 3000 samples/00:15:00 interval)
```

Note that only users which have jobs which are currently queued or have been queued since Moab was most recently started are listed.

Related topics

- showstats command (display user statistics)

mjobctl

Synopsis

mjobctl -c *jobexp*

mjobctl -c -w *attr=val*

mjobctl -C *jobexp*

mjobctl -e jobid

mjobctl -F *jobexp*

mjobctl -h [User|System|Batch|Defer|All] *jobexp*

mjobctl -m *attr*{+=|=|-=}*valjobexp*

mjobctl -N [<SIGNO>] *jobexp*

mjobctl -n <JOBNAME>

mjobctl -p <PRIORITY> *jobexp*

mjobctl -q {diag|starttime|hostlist} *jobexp*

mjobctl -r *jobexp*

mjobctl -R *jobexp*

mjobctl -s *jobexp*

mjobctl -u *jobexp*

mjobctl -w *attr*{+=|=|-=}*valjobexp*

mjobctl -x [-w flags=*val*] *jobexp*

Overview

The mjobctl command controls various aspects of jobs. It is used to submit, cancel, execute, and checkpoint jobs. It can also display diagnostic information about each job. The mjobctl command enables the Moab administrator to control almost all aspects of job behavior. See General Job Administration for more details on jobs and their attributes.

Format

-c - Cancel	
Format	JOBEXP
Description	Cancel a job. Use -w (following a -c flag) to specify job cancellation according to given credentials or job attributes. See -c -w for more information.
Example:	`> mjobctl -c job1045` *Cancel job* `job1045`.

-c -w - Cancel Where	
Format	*<ATTR>=<VALUE>* where *<ATTR>*=[user | account | qos | class | reqreservation(RsvName) | state (JobState) | job-name(JobName, not job ID)] | partition

-c -w - Cancel Where

Description	Cancel a job based on a given credential or job attribute. Use `-w` following a -c flag to specify job cancellation according to credentials or job attributes. (See examples.) See **Job States** on page 107 for a list of all valid job states. Also, you can cancel jobs from given partitions using `-w partition=<PAR1>[<PAR2>...]];` however, you must also either use another `-w` flag to specify a job or use the standard job expression.
Example	`> mjobctl -c -w state=USERHOLD` *Cancels all jobs that currently have a `USERHOLD` on them.* `> mjobctl -c -w user=user1 -w acct=acct1` *Cancels all jobs assigned to `user1` or `acct1`.*

-C - Checkpoint

Format	JOBEXP
Description	Checkpoint a job. See **Checkpoint/Restart Facilities** on page 528 for more information.
Example	`> mjobctl -C job1045` *Checkpoint job `job1045`.*

-e - Rerun

Format	JOBID
Description	Rerun the completed TORQUE job. This works only for jobs that are completed and show up in TORQUE as completed. This flag does not work with other resource managers.
Example	`> mjobctl -e job1045` *Rerun job `job1045`.*

-F - Force Cancel	
Format	JOBEXP
Description	Forces a job to cancel and ignores previous cancellation attempts.
Example	`> mjobctl -F job1045` *Force cancel job* `job1045`.

-h - Hold	
Format	*<HOLDTYPE><JOBEXP>* *<HOLDTYPE>* = { user | batch | system | defer | ALL }
Default	user
Description	Set or release a job hold See **Job Holds** on page 526 for more information
Example	`> mjobctl -h user job1045` *Set a user hold on job* `job1045`. `> mjobctl -u all job1045` *Unset all holds on job* `job1045`.

-m - Modify	
Format	`<ATTR>{ += | =| -= } <VAL>` *<ATTR>*={ account | arraylimit | awduration| class | cpuclock | deadline | depend | eeduration | env | features | feature | flags | gres | group | hold | hostlist | jobdisk | jobmem | jobname | job-swap | loglevel | messages | minstarttime | nodecount | notificationaddress | partition | priority | queue | qos | reqreservation | rmxstring | reqattr | reqawduration | sysprio | trig | trigvar | userprio | var | wclimit}

-m - Modify	
Description	Modify a specific job attribute. If an mjobctl -m attribute can affect how a job starts, then it generally cannot affect a job that is already running. For example, it is not feasible to change the **hostlist** of a job that is already running. The userprio attribute allows you to specify user priority. For job priority, use the '-p' flag. Modification of the job dependency is also communicated to the resource manager in the case of SLURM and PBS/TORQUE. Adding `--flags=warnifcompleted` causes a warning message to print when a job completes. To define values for `awduration`, `eeduration`, `minstarttime` (Note that the `minstarttime` attribute performs the same function as msub -a.), `reqawduration`, and `wclimit`, use the time spec format. A non-active job's partition list can be modified by adding or subtracting partitions. Note, though, that when adding or subtracting multiple partitions, each partition must have its own `-m partition{+= | = | -=}name` on the command line. (See example for adding multiple partitions.) To modify a job's generic resources, use the following format: `gres{ += | = | -= }` *`<gresName>`*`[:`*`<count>`*`]`. *<gresName>* is a single resource, not a list. *<count>* is an integer that, if not specified, is assumed to be 1. Modifying a job's generic resources causes Moab to append the new gres (+=), subtract the specified gres (-=), or clear out all existing generic resources attached to the job and override them with the newly-specified one (=).

-m - Modify	
Example	`> mjobctl -m reqawduration+=600 1664` *Add 10 minutes to the job walltime.* `> mjobctl -m eeduration=-1 1664` *Reset job's effective queue time, to when the job was submitted.* `> mjobctl -m var=Flag1=TRUE 1664` *Set the job variable `Flag1` to `TRUE`.* `> mjobctl -m notificationaddress="name@server.com"` *Sets the notification e-mail address associated with a job to `name@server.com`.* `> mjobctl -m partition+=p3 -m partition+=p4 Moab.5` *Adds multiple partitions (p3 and p4) to job `Moab.5`.* `> mjobctl -m arraylimit=10 sim.25` *Changes the concurrently running sub-job limit to 10 for array `sim.25`.* `> mjobctl -m gres=matlab:1 job0201` *Overrides all generic resources applied to job `job0201` and replaces them with 1 `matlab`.* `> mjobctl -m userprio-=100 Moab.4` *Reduces the user priority of `Moab.4` by 100.*

-N - Notify	
Format	[signal=]<*SIGID*>JOBEXP
Description	Send a signal to all jobs matching the job expression.

-N - Notify	
Example	`> mjobctl -N INT 1664` *Send an interrupt signal to job 1664.* `> mjobctl -N 47 1664` *Send signal 47 to job* `1664`*.*

-n - Name	
Format	
Description	Select jobs by job name.
Example	

-p - Priority	
Format	[+|+=|-=]*<VAL><JOBID>* [--flags=relative]
Description	Modify a job's system priority.

-p - Priority

Example	Priority is the job priority plus the system priority. Each format affects the job and system priorities differently. Using the format `<VAL><JOBID>` or `+<VAL><JOBID>` will set the system priority to the maximum system priority plus the specified value. Using `+=<VAL><JOBID>` or `<VAL><JOBID> --flags=relative` will relatively increase the job's priority and set the system priority. Using the format `-=<VAL> <JOBID>` sets the system priority to 0, and does not change priority based on *<VAL>* (it will not decrease priority by that number). For the following example, `job1045` has a priority of 10, which is composed of a job priority of 10 and a system priority of 0. `> mjobctl -p +1000 job1045` *The system priority changes to the max system priority plus `1000` points, ensuring that this job will be higher priority than all normal jobs. In this case, the job priority of 10 is not added, so the priority of `job1045` is now 1000001000.* `> mjobctl -p -=1 job1045` *The system priority of `job1045` resets to 0. The job priority is still 10, so the overall priority becomes 10.* `> mjobctl -p 3 job1045 --flags=relative` *Adds 3 points to the relative system priority. The priority for `job1045` changes from 10 to 13.*

-q - Query

Format	[diag(ALL)| hostlist | starttime| template] *<JOBEXP>*
Description	Query a job.

-q - Query	
Example	`> mjobctl -q diag job1045` *Query job `job1045`.* `> mjobctl -q diag ALL --format=xml` *Query all jobs and return the output in machine-readable XML.* `> mjobctl -q starttime job1045` *Query starttime of job `job1045`.* `> mjobctl -q template <job>` *Query job templates. If the <job> is set to ALL or empty, it will return information for all job templates.* `> mjobctl -q wiki <jobName>` *Query a job with the output displayed in a WIKI string. The job's name may be replaced with `ALL`.* `--flags=completed` will only work with the `diag` option.

-r - Resume	
Format	JOBEXP
Description	Resume a job.
Example	`> mjobctl -r job1045` *Resume job `job1045`.*

-R - Requeue	
Format	JOBEXP

-R - Requeue	
Description	Requeue a job. Adding `--flags=unmigrate` causes Moab to pull a grid job back to the central scheduler for further evaluation on all valid partitions.
Example	`> mjobctl -R job1045` *Requeue job `job1045`.*

-s - Suspend	
Format	JOBEXP
Description	Suspend a job. For more information, see Suspend/Resume Handling.
Example	`> mjobctl -s job1045` *Suspend job `job1045`.*

-u - Unhold	
Format	[<*TYPE*>[,<*TYPE*>]]JOBEXP <*TYPE*> = [user | system | batch | defer | ALL]
Default	`ALL`
Description	Release a hold on a job See **Job Holds** on page 526 for more information.
Example	`> mjobctl -u user,system scrib.1045` *Release user and system holds on job `scrib.1045`.*

-w - Where	
Format	[CompletionTime | StartTime][<= | = | >=]<*EPOCH_TIME*>

-w - Where	
Description	Add a where constraint clause to the current command. As it pertains to `CompletionTime` | `StartTime`, the where constraint only works for completed jobs. CompletionTime filters according to the completed jobs' completion times; StartTime filters according to the completed jobs' start times.
Example	`> mjobctl -q diag ALL --flags=COMPLETED --format=xml -w CompletionTime>=1246428000 -w CompletionTime<=1254376800` *Prints all completed jobs still in memory that completed between July 1, 2009 and October 1, 2009.*

-x - Execute	
Format	JOBEXP
Description	Execute a job. The `-w` option allows flags to be set for the job. Allowable flags are, `ignore-policies`, `ignorenodestate`, and `ignorersv`.
Example	`> mjobctl -x job1045` *Execute job `job1045`.* `> mjobctl -x -w flags=ignorepolicies job1046` *Execute job `job1046` and ignore policies, such as MaxJobPerUser.*

Parameters

JOB EXPRESSION	
Format	*<STRING>*

JOB EXPRESSION

Description	The name of a job or a regular expression for several jobs. The flags that support job expressions can use node expression syntax as described in Node Selection. Using `x:` indicates the following string is to be interpreted as a regular expression, and using `r:` indicates the following string is to be interpreted as a range. Job expressions do not work for array sub-jobs. Moab uses regular expressions conforming to the POSIX 1003.2 standard. This standard is somewhat different than the regular expressions commonly used for filename matching in Unix environments (see man 7 regex). To interpret a job expression as a regular expression, use `x:` or in the Moab configuration file (`moab.cfg`), set the parameter **USEJOBREGEX** to *TRUE* (and take note of the following caution). If you set **USEJOBREGEX** to *TRUE*, Moab treats all mjobctl job expressions as regular expressions regardless of whether wildcards are specified. This should be used with extreme caution since there is high potential for unintended consequences. For example, specifying `canceljob m.1` will not only cancel `m.1`, but also `m.11`,`m.12`,`m.13`, and so on. In most cases, it is necessary to quote the job expression (for example, `job13[5-9]`) to prevent the shell from intercepting and interpreting the special characters. The mjobctl command accepts a comma delimited list of job expressions. Example usage might be `mjobctl -r job[1-2],job4` or `mjobctl -c job1,job2,job4`.
Example:	see below

```
> mjobctl -c "x:80.*"
job '802' cancelled
job '803' cancelled
job '804' cancelled
job '805' cancelled
job '806' cancelled
job '807' cancelled
job '808' cancelled
job '809' cancelled
```

Cancel all jobs starting with `80`.

```
> mjobctl -m priority+=200 "x:74[3-5]"
job '743' system priority modified
job '744' system priority modified
job '745' system priority modified
```

```
> mjobctl -h x:17.*
# This puts a hold on any job that has a 17 that is followed by an unlimited amount
of any
# character and includes jobs 1701, 17mjk10, and 17DjN_JW-07

> mjobctl -h r:1-17
# This puts a hold on jobs 1 through 17.
```

XML Output

mjobctl information can be reported as XML as well. This is done with the command `mjobctl -q diag` *<JOB_ID>*.

XML Attributes

Name	Description
Account	The account assigned to the job
AllocNodeList	The nodes allocated to the job
Args	The job's executable arguments
AWDuration	The active wall time consumed
BlockReason	The block message index for the reason the job is not eligible
Bypass	Number of times the job has been bypassed by other jobs
Calendar	The job's timeframe constraint calendar
Class	The class assigned to the job
CmdFile	The command file path
CompletionCode	The return code of the job as extracted from the RM
CompletionTime	The time of the job's completion
Cost	The cost of executing the job relative to an allocation manager
CPULimit	The CPU limit for the job
Depend	Any dependencies on the status of other jobs
DRM	The master destination RM
DRMJID	The master destination RM job ID
EEDuration	The duration of time the job has been eligible for scheduling

Name	Description
EFile	The stderr file
Env	The job's environment variables set for execution
EnvOverride	The job's overriding environment variables set for execution
EState	The expected state of the job
EstHistStartTime	The estimated historical start time
EstPrioStartTime	The estimated priority start time
EstRsvStartTime	The estimated reservation start time
ExcHList	The excluded host list
Flags	Command delimited list of Moab flags on the job
GAttr	The requested generic attributes
GJID	The global job ID
Group	The group assigned to the job
Hold	The hold list
Holdtime	The time the job was put on hold
HopCount	The hop count between the job's peers
HostList	The requested host list
IFlags	The internal flags for the job
IsInteractive	If set, the job is interactive
IsRestartable	If set, the job is restartable
IsSuspendable	If set, the job is suspendable

Name	Description
IWD	The directory where the job is executed
JobID	The job's batch ID.
JobName	The user-specified name for the job
JobGroup	The job ID relative to its group
LogLevel	The individual log level for the job
MasterHost	The specified host to run primary tasks on
Messages	Any messages reported by Moab regarding the job
MinPreemptTime	The minimum amount of time the job must run before being eligible for preemption
Notification	Any events generated to notify the job's user
OFile	The stdout file
OldMessages	Any messages reported by Moab in the old message style regarding the job
OWCLimit	The original wallclock limit
PAL	The partition access list relative to the job
QueueStatus	The job's queue status as generated this iteration
QOS	The QoS assigned to the job
QOSReq	The requested QoS for the job
ReqAWDuration	The requested active walltime duration
ReqCMaxTime	The requested latest allowed completion time
ReqMem	The total memory requested/dedicated to the job
ReqNodes	The number of requested nodes for the job

Name	Description
ReqProcs	The number of requested procs for the job
ReqReservation	The required reservation for the job
ReqRMType	The required RM type
ReqSMinTime	The requested earliest start time
RM	The master source resource manager
RMXString	The resource manager extension string
RsvAccess	The list of reservations accessible by the job
RsvStartTime	The reservation start time
RunPriority	The effective job priority
Shell	The execution shell's output
SID	The job's system ID (parent cluster)
Size	The job's computational size
STotCPU	The average CPU load tracked across all nodes
SMaxCPU	The max CPU load tracked across all nodes
STotMem	The average memory usage tracked across all nodes
SMaxMem	The max memory usage tracked across all nodes
SRMJID	The source RM's ID for the job
StartCount	The number of the times the job has tried to start
StartPriority	The effective job priority
StartTime	The most recent time the job started executing

Name	Description
State	The state of the job as reported by Moab
StatMSUtl	The total number of memory seconds utilized
StatPSDed	The total number of processor seconds dedicated to the job
StatPSUtl	The total number of processor seconds utilized by the job
StdErr	The path to the stderr file
StdIn	The path to the stdin file
StdOut	The path to the stdout file
StepID	StepID of the job (used with LoadLeveler systems)
SubmitHost	The host where the job was submitted
SubmitLanguage	The RM language that the submission request was performed
SubmitString	The string containing the entire submission request
SubmissionTime	The time the job was submitted
SuspendDuration	The amount of time the job has been suspended
SysPrio	The admin specified job priority
SysSMinTime	The system specified min. start time
TaskMap	The allocation taskmap for the job
TermTime	The time the job was terminated
User	The user assigned to the job
UserPrio	The user specified job priority
UtlMem	The utilized memory of the job

Name	Description
UtlProcs	The number of utilized processors by the job
Variable	
VWCTime	The virtual wallclock limit

Examples

Example 3-34:

```
> mjobctl -q diag ALL --format=xml
<Data><job AWDuration="346" Class="batch" CmdFile="jobsleep.sh" EEDuration="0"
EState="Running" Flags="RESTARTABLE" Group="test" IWD="/home/test" JobID="11578"
QOS="high"
RMJID="11578.lolo.icluster.org" ReqAWDuration="00:10:00" ReqNodes="1" ReqProcs="1"
StartCount="1"
StartPriority="1" StartTime="1083861225" StatMSUtl="903.570" StatPSDed="364.610"
StatPSUtl="364.610"
State="Running" SubmissionTime="1083861225" SuspendDuration="0" SysPrio="0"
SysSMinTime="00:00:00"
User="test"><req AllocNodeList="hana" AllocPartition="access" ReqNodeFeature="[NONE]"
ReqPartition="access"></req></job><job AWDuration="346" Class="batch"
CmdFile="jobsleep.sh"
EEDuration="0" EState="Running" Flags="RESTARTABLE" Group="test" IWD="/home/test"
JobID="11579"
QOS="high" RMJID="11579.lolo.icluster.org" ReqAWDuration="00:10:00" ReqNodes="1"
ReqProcs="1"
StartCount="1" StartPriority="1" StartTime="1083861225" StatMSUtl="602.380"
StatPSDed="364.610"
StatPSUtl="364.610" State="Running" SubmissionTime="1083861225" SuspendDuration="0"
SysPrio="0"
SysSMinTime="00:00:00" User="test"><req AllocNodeList="lolo" AllocPartition="access"
ReqNodeFeature="[NONE]" ReqPartition="access"></req></job></Data>
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- setspri
- canceljob
- runjob

mnodectl

Synopsis

mnodectl -m *attr*{=|-=}*val* *nodeexp*
mnodectl -q [*cat*|*diag*|*profile*|*wiki*] *nodeexp*

Overview

Change specified attributes for a given node expression.

Access

By default, this command can be run by any Moab Administrator.

Format

-m - Modify	
Format	<*ATTR*>{=|-=|+=}<*VAL*> Where <*ATTR*> is one of the following: FEATURES GEVENT, GMETRIC, MESSAGE, OS, POWER, STATE, VARIABLE and −=, except when used for features, clears the attribute instead of decrementing the attribute's value and = indicates that you are specifying a new value to replace the old one(s), if any. When the −= option is used to modify features, it removes the specified features from the node. The += option, which is only available for features, allows you to append additional features to the current list rather than replacing the current list entirely. Changing OS and POWER require a Moab Adaptive Computing Suite license and a provisioning resource manager.
Description	Modify the state or attribute of specified node(s)
Example	`> mnodectl -m features+=fastio,highmem node1` `> mnodectl -m gevent=cpufail:'cpu02 has failed w/ec:0317' node1` `> mnodectl -m gmetric=temp:131.2 node1` `> mnodectl -m message='cpufailure:cpu02 has failed w/ec:0317' node1` `> mnodectl -m OS=RHAS30 node1` `> mnodectl -m power=off node1` `> mnodectl -m state=idle node1` `> mnodectl -m variable=IP=10.10.10.100,Location=R1S2 node1`

-q - Query	
Format	{cat | diag | profile | wiki}

-q - Query	
Description	Query node categories or node profile information (see ENABLEPROFILING for nodes). The `diag` and `profile` options must use `--xml`.
Example	See examples below.

```
> mnodectl -q cat ALL
node categorization stats from Mon Jul 10 00:00:00 to Mon Jul 10 15:30:00
Node: moab
 Categories:
                    busy: 96.88%
                    idle: 3.12%
Node: maka
 Categories:
                    busy: 96.88%
                    idle: 3.12%
Node: pau
 Categories:
                    busy: 96.88%
                    idle: 3.12%
Node: maowu
 Categories:
                    busy: 96.88%
                 down-hw: 3.12%
Cluster Summary:
                    busy: 96.88%
                 down-hw: 0.78%
                    idle: 2.34%
```

```
> mnodectl -v -q profile
...
```

```
> mnodectl -q wiki <ALL>
GLOBAL STATE=Idle PARTITION=SHARED
n0 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n1 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n2 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n3 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n4 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n5 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n6 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n7 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n8 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n9 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
```

Query a node with the output displayed in a WIKI string.

Parameters

FEATURES	
Format	*<STRING>* One of the following: • a comma-delimited list of features • [NONE] (to clear features on the node)
Description	Sets the features on a node. These node features will be overwritten when an RM reports features.
Example	`mnodectl -m features=fastio,highmem node1` `mnodectl -m features=[NONE] node1`

GEVENT	
Format	*<EVENT>*:*<MESSAGE>*
Description	Creates a generic event on the node to which Moab may respond (see Enabling Generic Events).
Example	`mnodectl -m gevent=powerfail:'power has failed' node1`

GMETRIC	
Format	*<ATTR>*:*<VALUE>*
Description	Sets the value for a generic metric on the node (see Enabling Generic Metrics). When a gmetric set in Moab conflicts with what the resource manager reports, Moab uses the set gmetric until the next time the resource manager reports a different number.
Example	`mnodectl -m gmetric=temp:120 node1`

MESSAGE	
Format	'*<MESSAGE>*'

MESSAGE	
Description	Sets a message to be displayed on the node.
Example	`mnodectl -m message='powerfailure: power has failed' node1`

NODEEXP	
Format	*<STRING>* Where *<NODEEXP>* is a node name, regex or ALL
Description	Identifies one or more nodes.
Example	`node1` — applies only to `node1` `fr10n*` - all nodes starting with `fr10n` `ALL` - all known nodes

OS	
Format	*<STRING>*
Description	Operating System (see Resource Provisioning).
Example	`mnodectl node1 -m OS=RHELAS30`

POWER	
Format	{off|on}

POWER	
Description	Set the power state of a node. Action will NOT be taken if the node is already in the specified state. If you power off a node, a green policy will try to turn it back on. If you want the node to remain powered off, you must associate a reservation with it. If you request to power off a node that has active work on it, Moab returns a status indicating that the node is busy (with a job or VM) and will not be powered off. You will see one of these messages: • `Ignoring node <name>: power ON in process` (indicates node is currently powering on) • `Ignoring node <name>: power OFF in process` (indicates node is currently powering off) • `Ignoring node <name>: has active VMs running` (indicates the node is currently running active VMs) • `Ignoring node <name>: has active jobs running` (indicates the node is currently running active jobs) Once you resolve the activity on the node (by preempting or migrating the jobs or VMs, for example), you can attempt to power the node off again. You can use the `--flags=force` option to cause a force override. However, doing this will power off the node regardless of whether or not its jobs get migrated or preempted (i.e., you run the risk of losing the VMs/jobs entirely). For example: `> mnodectl node1 -m power=off --flags=force`
Example	`> mnodectl node1 -m power=off`

STATE	
Format	{drained|idle}
Description	Remove (drained) or add (idle) a node from scheduling.
Example	`mnodectl node1 -m state=drained` *Moab ignores `node1` when scheduling.*

VARIABLE	
Format	*<name>*[=*<value>*],*<name>*[=*<value>*]...

VARIABLE	
Description	Set a list of variables for a node.
Example	`> mnodectl node1 -m variable=IP=10.10.10.100,Location=R1S2`

Related topics

- Moab Client Installation — explains how to distribute this command to client nodes
- mdiag -n
- showres -n
- checknode
- showstats -n — report current and historical node statistics

moab

Synopsis

moab --about --help --loglevel=<LOGLEVEL> --version [-c <CONFIG_FILE>] [-C] [-d] [-e] [-h] [-P [<PAUSEDURATION>]] [-R <RECYCLEDURATION>] [-s] [-S [<STOPITERATION>]] [-v]

Parameters

Parameter	Description
--about	Displays build environment and version information.
--loglevel	Sets the server loglevel to the specified value.
--version	Displays version information.
-c	Configuration file the server should use.
-C	Clears checkpoint files (`.moab.ck`, `.moab.ck.1`).
-d	Debug mode (does not background itself).

Parameter	Description
-e	Forces Moab to exit if there are any errors in the configuration file, if it can't connect to the configured database, or if it can't find these directories: • `statdir` • `logdir` • `spooldir` • `toolsdir`
-P	Starts Moab in a paused state for the duration specified.
-R	Causes Moab to automatically recycle every time the specified duration transpires.
-s	Starts Moab in the state that was most recently checkpointed.
-S	Suspends/stops scheduling at specified iteration (or at startup if no iteration is specified).
-v	Same as --version.

mrmctl

Synopsis

mrmctl -f [*fobject*] {*rmName*|am:[*amid*]} mrmctl -l [*rmid*|am:[*amid*]] mrmctl -m <attr>=<value> [*rmid*|am:[*amid*]] mrmctl -p {*rmid*|am:[*amid*]} mrmctl -R {AM|ID}[:*RMID*]}}

Overview

mrmctl allows an admin to query, list, modify, and ping the resource managers and allocation managers in Moab. mrmctl also allows for a queue (often referred to as a class) to be created for a resource manager.

Access

By default, this command can be run by level 1 and level 2 Moab administrators (see ADMINCFG).

Format

-f - Flush Statistics	
Format	[<*fobject*>] where *fobject* is optional and one of messages or stats.

-f - Flush Statistics	
Default	If no *fobject* is specified, then reported failures and performance data will be flushed. If no resource manager id is specified, the first resource manager will be flushed.
Description	Clears resource manager statistics. If messages is specified, then reported failures, performance data, and messages will be flushed.
Example	`> mrmctl -f base` *Moab will clear the statistics for RM* `base`.

-l - List	
Format	N/A
Default	All RMs and AMs (when no RM/AM is specified)
Description	List Resource and Allocation Manager(s)
Example	`> mrmctl -l` *Moab will list all resource and allocation managers.*

-m - Modify	
Format	N/A
Default	All RMs and AMs (when no RM/AM is specified).
Description	Modify Resource and Allocation Manager(s).
Example	`> mrmctl -m state=disabled peer13`

-p - Ping	
Format	N/A

-p - Ping	
Default	First RM configured.
Description	Ping Resource Manager.
Example	`> mrmctl -p base` *Moab will ping RM* `base`*.*

-R - Reload	
Format	{AM|ID}[:*RMID*]}}
Description	Dynamically reloads server information for the identity manager service if `ID` is specified; if `AM` is specified, reloads the allocation manager service.
Example	`> mrmctl -R ID` `Reloads the identity manager on demand.`

Resource manager interfaces can be enabled/disabled using the modify operation to change the resource manager state as in the following example:

```
# disable active resource manager interface
> mrmctl -m state=disabled torque
# restore disabled resource manager interface
> mrmctl -m state=enabled torque
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mdiag -R
- mdiag -c

mrsvctl

Synopsis

mrsvctl -c [-a *acl*] [-b*subtype*] [-d *duration*] [-D *description*] [-e *endtime*] [-E] [-f *features*] [-F *flags*] [-g *rsvgroup*] [-h *hostexp*] [-n *name*] [-o *owner*] [-p *partition*] [-P *profile*] [-R *resources*] [-s *starttime*] [-S *setvalue*] [-t *tasks*] [-T *trigger*] [-V *variable*] [-x joblist] mrsvctl -C [-g *standing_reservationid*] {*reservationid*}

mrsvctl -l [{*reservationid* | -i *index*}]

mrsvctl -m <duration|endtime|reqtaskcount|starttime>{=|+=|-=}<VAL> <hostexp>{+=|-=}<VAL> <variable>{+=KEY=VAL|-=KEY_TO_REMOVE} {*reservationid* | -i *index*}

mrsvctl -q {*reservationid* | -i *index*} [--blocking]

mrsvctl -r {*reservationid* | -i *index*}

Overview

mrsvctl controls the creation, modification, querying, and releasing of reservations.

The timeframe covered by the reservation can be specified on either an absolute or relative basis. Only jobs with credentials listed in the reservation's access control list can utilize the reserved resources. However, these jobs still have the freedom to utilize resources outside of the reservation. The reservation will be assigned a name derived from the ACL specified. If no reservation ACL is specified, the reservation is created as a system reservation and no jobs will be allowed access to the resources during the specified timeframe (valuable for system maintenance, etc.). See the Reservation Overview for more information.

Reservations can be viewed using the `-q` flag and can be released using the `-r` flag.

By default, reservations are not exclusive and may overlap with other reservations and jobs. Use the '-E' flag to adjust this behavior.

Access

By default, this command can be run by level 1 and level 2 Moab administrators (see ADMINCFG).

Format

-a	
Name	ACL

-a	
Format	*<TYPE>*==*<VAL>*[,*<TYPE>*==*<VAL>*]... Where *<TYPE>* is one of the following: ACCT, CLASS, DURATION, GROUP, JATTR, PROC, QOS, or USER
Description	List of limitations for access to the reserved resources (See also: ACL Modifiers).
Example	`> mrsvctl -c -h node01 -a USER==john+,CLASS==batch-` *Moab will make a reservation on `node01` allowing access to user john and restricting access from class batch when other resources are available to class batch* `> mrsvctl -m -a USER-=john system.1` *Moab will remove user john from the `system.1` reservation*

-a	
Notes	• When you specify multiple credentials, a user must only match one of them in order to access the reservation. To require one or more of the listed limitations for reservation access, each required specification must end with an asterisk (*). If a user meets the required limitation(s), he or she has access to the reservation (without meeting any that are not marked required). • There are three different assignment operators that can be used for modifying most credentials in the ACL. The operator == will reassess the list for that particular credential type. The += operator will append to the list for that credential type, and -= will remove from the list. Two other operators are used to specify DURATION and PROC: >= (greater than) and <= (less than). • To add multiple credentials of the same type with one command, use a colon to separate them. To separate lists of different credential types, use commas. For example, to reassign the user list to consist of users Joe and Bob, and to append the group MyGroup to the groups list on the `system.1` reservation, you could use the command `mrsvctl -m -a USER==Joe:Bob,GROUP+=MyGroup system.1`. • Any of the ACL modifiers may be used. When using them, it is often useful to put single quotes on either side of the assignment command. For example, `mrsvctl -m -a 'USER==&Joe' system.1`. • Some flags are mutually exclusive. For example, the ! modifier means that the credential is blocked from the reservation and the & modifier means that the credential must run on that reservation. Moab will take the most recently parsed modifier. Modifiers may be placed on either the left or the right of the argument, so `USER==&JOE` and `USER==JOE&` are equivalent. Moab parses each argument starting from right to left on the right side of the argument, then from left to right on the left side. So, if the command was `USER==!Joe&`, Moab would keep the equivalent of `USER==!Joe` because the `!` would be the last one parsed. • You can set a reservation to have a time limit for submitted jobs using DURATION and the * modifier. For example, `mrsvctl -m -a 'DURATION<=*1:00:00' system.1` would cause the `system.1` reservation to not accept any jobs with a walltime greater than one hour. Similarly, you can set a reservation to have a processor limit using PROC and the * modifier. `mrsvctl -a 'PROC>=2*' system.2` would cause the `system.2` reservation to only allow jobs requesting more than 2 procs to run on it. • You can verify the ACL of a reservation using the mdiag -r command. `mrsvctl -m -a 'USER==Joe:Bob,GROUP-=BadGroup,ACCT+=GoodAccount,DURATION<=*1:00:00' system.1` *Moab will reassign the `USER` list to be `Joe` and `Bob`, will remove `BadGroup` from the `GROUP` list, append `GoodAccount` to the `ACCT` list, and only allow jobs that have a submitted walltime of an hour or less on the `system.1` reservation.*

-a	
	`mrsvctl -m -a 'USER==Joe,USER==Bob' system.1` *Moab will assign the `USER` list to `Joe`, and then reassign it again to `Bob`. The final result will be that the `USER` list will just be `Bob`. To add `Joe` and `Bob`, use `mrsvctl -m -a USER==Joe:Bob system.1` or `mrsvctl -m -a USER==Joe,USER+=Bob system.1`.*

-b	
Name	SUBTYPE
Format	One of the node category values or node category shortcuts.
Description	Add subtype to reservation.
Example	`> mrsvctl -c -b SoftwareMaintenance -t ALL` *Moab will associate the reserved nodes with the node category SoftwareMaintenance.*

-c	
Name	CREATE
Format	*<ARGUMENTS>*
Description	Creates a reservation. The `-x` flag, when used with `-F ignjobrsv`, lets users create reservations but exclude certain nodes from being part of the reservation because they are running specific jobs. The `-F` flag instructs mrsvctl to still consider nodes with current running jobs.

-c	
Examples	`> mrsvctl -c -t ALL` *Moab will create a reservation across all system resources.* `> mrsvctl -c -t 5 -F ignjobrsv -x moab.5,moab.6` *Moab will create the reservation while assigning the nodes. Nodes running jobs `moab5` and `moab6` will not be assigned to the reservation.* `> mrsvctl -c -d INFINITY` *Moab will create an infinite reservation.*

-C	
Name	CLEAR
Format	`<RSVID> | -g <SRSVID>`
Description	Clears any disabled time slots from standing reservations and allows the recreation of disabled reservations
Example	`> mrsvctl -C -g testing` *Moab will clear any disabled timeslots from the standing reservation `testing`.*

-d	
Name	DURATION
Format	`[[[DD:]HH:]MM:]SS`
Default	INFINITY
Description	Duration of the reservation (not needed if ENDTIME is specified)

-d	
Example	`> mrsvctl -c -h node01 -d 5:00:00` *Moab will create a reservation on* `node01` *lasting 5 hours.* `mrsvctl -c -d INFINITY` *Moab will create a reservation with a duration of INFINITY (no endtime).*

-D	
Name	DESCRIPTION
Format	*<STRING>*
Description	Human-readable description of reservation or purpose
Example	`> mrsvctl -c -h node01 -d 5:00:00 -D 'system maintenance to test network'` *Moab will create a reservation on* `node01` *lasting 5 hours.*

-e	
Name	ENDTIME
Format	[HH[:MM[:SS]]][_MO[/DD[/YY]]] or +[[[DD:]HH:]MM:]SS
Default	INFINITY
Description	Absolute or relative time reservation will end (not required if Duration specified). ENDTIME also supports an epoch timestamp.

-e	
Example	`> mrsvctl -c -h node01 -e +3:00:00` *Moab will create a reservation on* `node01` *ending in 3 hours.*

-E	
Name	EXCLUSIVE
Description	When specified, Moab will only create a reservation if there are no other reservations (exclusive or otherwise) which would conflict with the time and space constraints of this reservation. If exceptions are desired, the rsvaccesslist attribute can be set or the ignrsv flag can be used.
Example	`> mrsvctl -c -h node01 -E` *Moab will only create a reservation on* `node01` *if no conflicting reservations are found.* This flag is only used at the time of reservation creation. Once the reservation is created, Moab allows jobs into the reservation based on the ACL. Also, once the exclusive reservation is created, it is possible that Moab will overlap it with jobs that match the ACL.

-f	
Name	FEATURES
Format	`<STRING>[:<STRING>]...`
Description	List of node features which must be possessed by the reserved resources. You can use a backslash and pipe to delimit features to indicate that Moab can use one or the other.
Example	`> mrsvctl -c -h node[0-9] -f fast|slow` *Moab will create a reservation on nodes matching the expression and which also have either the feature* `fast` *or the feature* `slow`*.*

-F	
Name	FLAGS
Format	`<flag>[[,<flag>]...]`
Description	Comma-delimited list of flags to set for the reservation (see Managing Reservations for flags).
Example	`> mrsvctl -c -h node01 -F ignstate` *Moab will create a reservation on* `node01` *ignoring any conflicting node states.*

-g	
Name	RSVGROUP
Format	*<STRING>*
Description	For a create operation, create a reservation in this reservation group. For list and modify operations, take actions on all reservations in the specified reservation group. The `-g` option can also be used in conjunction with the -r option to release a reservation associated with a specified group. See Reservation Group for more information.
Example	`> mrsvctl -c -g staff -h 'node0[1-9]'` *Moab will create a reservation on nodes matching the node expression given and assign it to the reservation group* `staff`*.*

-h	
Name	HOSTLIST
Format	class:*<classname>*[,*<classname>*]... or *<STRING>* or 'r:*<nodeNameStart>*[*<beginRange>*-*<endRange>*]' or ALL

-h	
Description	Host expression or a class mapping indicating the nodes which the reservation will allocate. When you specify a *<STRING>*, the HOSTLIST attribute is always treated as a regular expression. `foo10` will map to `foo10`, `foo101`, `foo1006`, etc. To request an exact host match, the expression can be bounded by the carat and dollar op expression markers as in `^foo10$`.
Example	`> mrsvctl -c -h 'r:node0[1-9]'` *Moab will create a reservation on nodes* `node01`, `node02`, `node03`, `node04`, `node05`, `node06`, `node07`, `node08`, *and* `node09`. `> mrsvctl -c -h class:batch` *Moab will create a reservation on all nodes which support class/queue* `batch`.

-i	
Name	INDEX
Format	*<STRING>*
Description	Use the reservation index instead of full reservation ID.
Example	`> mrsvctl -m -i 1 starttime=+5:00` *Moab will create a reservation on nodes matching the expression given.*

-l	
Name	LIST
Format	*<RSV_ID>* or ALL *RSV_ID* can be the name of a reservation or a regular expression.

-l	
Default	ALL
Description	List reservation(s).
Example	`> mrsvctl -l system*` *Moab will list all of the reservations whose names start with* `system`.

-m	
Name	MODIFY
Format	<*ATTR*>=<*VAL*>[-m <*ATTR2*>=<*VAL2*>]... Where <*ATTR*> is one of the following: **flags** **duration**: duration{+=|-=|=}<*RELTIME*> **endtime**: endtime{+=|-=}<*RELTIME*> or endtime=<*ABSTIME*> **hostexp**: hostexp[+=|-=]<*node*>[,<*node*>] **variable**: variable[+=key1=val1|-=key_to_remove] **reqtaskcount**: reqtaskcount{+=|-=|=}<*TASKCOUNT*> **starttime**: starttime{+=|-=}<*RELTIME*> or starttime=<*ABSTIME*>
Description	Modify aspects of a reservation. Moab is constantly scheduling and updating reservations. Before modifying a reservation it is recommended that you first stop the scheduler (mschedctl -s) so that the scheduler and reservation are in a stable and steady state. Once the reservation has been modified, resume the scheduler with mschedctl -r.

-m	
Example	`> mrsvctl -m duration=2:00:00 system.1` *Moab sets the duration of reservation `system.1` to be exactly two hours, thus modifying the endtime of the reservation.* `> mrsvctl -m starttime+=5:00:00 system.1` *Moab advances the starttime of `system.1` five hours from its current starttime (without modifying the duration of the reservation).* `> mrsvctl -m endtime-=5:00:00 system.1` *Moab moves the endtime of reservation `system.1` ahead five hours from its current endtime (without modifying the starttime; thus, this action is equivalent to modifying the duration of the reservation).* `> mrsvctl -m starttime=15:00:00_7/6/08 system.1` *Moab sets the starttime of reservation `system.1` to 3:00 p.m. on July 6, 2008.* `> mrsvctl -m starttime-=5:00:00 system.1` *Moab moves the starttime of reservation `system.1` ahead five hours.* `> mrsvctl -m starttime+=5:00:00 system.1` *Moab moves the starttime of reservation `system.1` five hours from the current time.* `> mrsvctl -m -duration+=5:00:00 system.1` *Moab extends the duration of `system.1` by five hours.* `> mrsvctl -m flags+=ADVRES system.1` *Moab adds the flag `ADVRES` to reservation `system.1`.*

-m	
	`> mrsvctl -m variable+key1=val1 system.1` *Moab adds the variable `key1` with the value `key2` to `system.1`.* `> mrsvctl -m variable+=key1=val1 variable+=key2=val2 system.1` *Moab adds the variable `key1` with the value `val1`, and variable `key2` with `val2` to `system.1`. (Note that each variable flag requires a distinct `-m` entry.)* `> mrsvctl -m variable-=key1 system.1` *Moab deletes the variable `key1` from `system.1`.* `> mrsvctl -m variable-=key1 -m variable-=key2 system.1` *Moab deletes the variables `key1` and `key2` from `system.1`.*

-m	
Notes:	• Modifying the starttime does not change the duration of the reservation, so the endtime changes as well. The starttime can be changed to be before the current time, but if the change causes the endtime to be before the current time, the change is not allowed. • Modifying the endtime changes the duration of the reservation as well (and vice versa). An endtime *cannot* be placed before the starttime or before the current time. • Duration cannot be negative. • The += and −= operators operate on the time of the reservation (`starttime+=5` adds five seconds to the current reservation starttime), while + and − operate on the current time (`starttime+5` sets the starttime to five seconds from now). • If the starttime or endtime specified is before the current time without a date specified, it is set to the next time that fits the command. To force the date, add the date as well. For the following examples, assume that the current time is 9:00 a.m. on March 1, 2007. `> mrsvctl -m starttime=8:00:00_3/1/07 system.1` *Moab moves `system.1`'s starttime to 8:00 a.m., March 1.* `> mrsvctl -m starttime=8:00:00 system.1` *Moab moves `system.1`'s starttime to 8:00 a.m., March 2.* `> mrsvctl -m endtime=7:00:00 system.1` *Moab moves `system.1`'s endtime to 7:00 a.m., March 3. This happens because the endtime must also be after the starttime, so Moab continues searching until it has found a valid time that is in the future and after the starttime.* `> mrsvctl -m endtime=7:00:00_3/2/07 system.1` *Moab will return an error because the endtime cannot be before the starttime.*

-n	
Name	NAME
Format	*<STRING>*

-n	
Description	Name for new reservation. If no name is specified, the reservation name is set to first name listed in ACL or `SYSTEM` if no ACL is specified. Reservation names may not contain whitespace.
Example	`mrsvctl -c -h node01 -n John` *Moab will create a reservation on node01 with the name* `John`.

-o	
Name	OWNER
Format	*<CREDTYPE>:<CREDID>*
Description	Specifies the owner of a reservation. See Reservation Ownership for more information.
Example	`mrsvctl -c -h node01 -o USER:user1` *Moab creates a reservation on* `node01` *owned by* `user1`.

-p	
Name	PARTITION
Format	*<STRING>*
Description	Only allocate resources from the specified partition
Example	`mrsvctl -c -p switchB -t 14` *Moab will allocate 14 tasks from the* `switchB` *partition.*

-P	
Name	PROFILE
Format	*<STRING>*
Description	Indicates the reservation profile to load when creating this reservation
Example	`mrsvctl -c -P testing2 -t 14` *Moab will allocate 14 tasks to a reservation defined by the `testing2` reservation profile.*

-q	
Name	QUERY
Format	*<RSV_ID>* — The `-r` option accepts x: node regular expressions and r: node range expressions (asterisks (*) are supported wildcards as well).
Description	Get diagnostic information or list all completed reservations. The command gathers information from the Moab cache which prevents it from interrupting the scheduler, but the `--blocking` option can be used to bypass the cache and interrupt the scheduler.
Example	`mrsvctl -q ALL` *Moab will query reservations.* `mrsvctl -q system.1` *Moab will query the reservation `system.1`.*

-r	
Name	RELEASE
Format	*<RSV_ID>* — The `-r` option accepts x: node regular expressions and r: node range expressions (asterisks (`*`) are supported wildcards as well).

-r	
Description	Releases the specified reservation.
Example	`> mrsvctl -r system.1` *Moab will release reservation* `system.1`. `> mrsvctl -r -g idle` *Moab will release all idle job reservations.*

-R	
Name	RESOURCES
Format	<*tid*> or <*RES*>=<*VAL*>[{,|+|;}<*RES*>=<*VAL*>]... Where <*RES*> is one of the following: PROCS, MEM, DISK, SWAP, GRES
Default	PROCS=-1
Description	Specifies the resources to be reserved per task. (`-1` indicates all resources on node) For GRES resources, <*VAL*> is specified in the format `<GRESNAME>[:<COUNT>]`
Example	`> mrsvctl -c -R MEM=100;PROCS=2 -t 2` *Moab will create a reservation for two tasks with the specified resources.*

-s	
Name	STARTTIME
Format	`[HH[:MM[:SS]]][_MO[/DD[/YY]]]` or `+[[[DD:]HH:]MM:]SS`
Default	[NOW]
Description	Absolute or relative time reservation will start. STARTTIME also supports an epoch timestamp.
Example	`> mrsvctl -c -t ALL -s 3:00:00_4/4/04` *Moab will create a reservation on all system resources at 3:00 am on April 4, 2004* `> mrsvctl -c -h node01 -s +5:00` *Moab will create a reservation in 5 minutes on* `node01` `> mrsvctl -m -s -=5:00 system.1` *This will decrement the start time by 5 minutes.*

-S	
Name	SET ATTRIBUTE
Format	*<ATTR>*=*<VALUE>* where *<ATTR>* is one of aaccount — Accountable account agroup — accountable group aqos — accountable QoS auser — accountable user reqarch — required architecture reqmemory — required node memory - in MB reqos — required operating system rsvaccesslist — comma-delimited list of reservations or reservation groups which can be accessed by this reservation request. Because each reservation can access all other reservations by default, you should make any reservation with a specified rsvaccesslist exclusive by setting the **-E** on page 249 flag. This setting gives the otherwise exclusive reservation access to reservations specified in the list.

-S

Description	Specifies a reservation attribute will be used to create this reservation
Example	`> mrsvctl -c -h node01 -S aqos=high` *Moab will create a reservation on* `node01` *and will use the QOS* `high` *as the accountable credential*

-t

Name	TASKS
Format	*<INTEGER>*[-*<INTEGER>*]
Description	Specifies the number of tasks to reserve. ALL indicates all resources available should be reserved. If the task value is set to ALL, Moab applies the reservation regardless of existing reservations and exclusive issues. If an integer is used, Moab only allocates accessible resources. If a range is specified Moab attempts to reserve the maximum number of tasks, or at least the minimum.
Example	`> mrsvctl -c -t ALL` *Moab will create a reservation on all resources.* `> mrsvctl -c -t 3` *Moab will create a reservation for three tasks.* `> mrsvctl -c -t 3-10 -E` *Moab will attempt to reserve 10 tasks but will fail if it cannot get at least three.*

-T

Name	TRIGGER

-T	
Format	*<STRING>*
Description	Comma-delimited reservation trigger list following format described in the trigger format section of the reservation configuration overview. See **Creating a trigger** on page 727 for more information. To cancel a standing reservation with a trigger, the **SRCFG** parameter's attribute DEPTH must be set to *0*.
Example	`> mrsvctl -c -h node01 -T offset=200,etype=start,atype=exec,action=/tmp/email.sh` *Moab will create a reservation on `node01` and fire the script `/tmp/email.sh` 200 seconds after it starts*

-V	
Name	VARIABLE
Format	*<name>*[=*<value>*][[;*<name>*[=*<value>*]]...]
Description	Semicolon-delimited list of variables that will be set when the reservation is created (See **About trigger variables** on page 754 for more information.). Names with no values will simply be set to *TRUE*.
Example	`> mrsvctl -c -h node01 -V $T1=mac;var2=18.19` *Moab will create a reservation on `node01` and set `$T1` to `mac` and `var2` to `18.19`.* For information on modifying a variable on a reservation, see MODIFY.

-x	
Name	JOBLIST
Format	-x *<jobs to be excluded>*

-x	
Description	The `-x` flag, when used with `-F ignjobrsv`, lets users create reservations but exclude certain nodes that are running the listed jobs. The `-F` flag instructs mrsvctl to still consider nodes with current running jobs. The nodes are not listed directly.
Example	`> mrsvctl -c -t 5 -F ignjobrsv -x moab.5,moab.6` *Moab will create the reservation while assigning the nodes. Nodes running jobs `moab5` and `moab6` will not be assigned to the reservation.*

Parameters

RESERVATION ID	
Format	<STRING>
Description	The name of a reservation or a regular expression for several reservations.
Example	`system*` *Specifies all reservations starting with system.*

Resource Allocation Details

When allocating resources, the following rules apply:

- When specifying tasks, each task defaults to one full compute node unless otherwise specified using the -R specification
- When specifying tasks, the reservation will not be created unless all requested resources can be allocated. (This behavior can be changed by specifying -F besteffort)
- When specifying tasks or hosts, only nodes in an idle or running state will be considered. (This behavior can be changed by specifying -F ignstate)

Reservation Timeframe Modification

Moab supports dynamically modifying the timeframe of existing reservations. This can be accomplished using the mrsvctl -m flag. By default, Moab will perform advanced boundary and resource access to verify that the modification does not result in an invalid scheduler state. However, in certain circumstances administrators may wish to FORCE the modification in spite of any access violations. This can be done using the switch `mrsvctl -m --flags=force` which forces Moab to bypass any access verification and force the change through.

Extending a reservation by modifying the endtime

The following increases the endtime of a reservation using the += tag:

```
$> showres
ReservationID       Type S       Start          End    Duration    N/P    StartTime
system.1            User -    11:35:57  1:11:35:57  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m endtime+=24:00:00 system.1
endtime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start          End    Duration    N/P    StartTime
system.1            User -    11:35:22  2:11:35:22  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

The following increases the endtime of a reservation by setting the endtime to an absolute time:

```
$> showres
ReservationID       Type S       Start          End    Duration    N/P    StartTime
system.1            User -    11:33:18  1:11:33:18  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m endtime=0_11/20 system.1
endtime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start          End    Duration    N/P    StartTime
system.1            User -    11:33:05  2:11:33:05  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

Extending a reservation by modifying the duration

The following increases the duration of a reservation using the += tag:

```
$> showres
ReservationID       Type S       Start          End    Duration    N/P    StartTime
system.1            User -    11:28:46  1:11:28:46  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m duration+=24:00:00 system.1
duration for rsv 'system.1' changed
>$ showres
ReservationID       Type S       Start          End    Duration    N/P    StartTime
system.1            User -    11:28:42  2:11:28:42  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

The following increases the duration of a reservation by setting the duration to an absolute time:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:26:41  1:11:26:41  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m duration=48:00:00 system.1
duration for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:26:33  2:11:26:33  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

Shortening a reservation by modifying the endtime

The following modifies the endtime of a reservation using the -= tag:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:15:51  2:11:15:51  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m endtime-=24:00:00 system.1
endtime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:15:48  1:11:15:48  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

The following modifies the endtime of a reservation by setting the endtime to an absolute time:

```
$ showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:14:00  2:11:14:00  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m endtime=0_11/19 system.1
endtime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:13:48  1:11:13:48  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

Shortening a reservation by modifying the duration

The following modifies the duration of a reservation using the -= tag:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:12:20  2:11:12:20  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m duration-=24:00:00 system.1
duration for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:12:07  1:11:12:07  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

The following modifies the duration of a reservation by setting the duration to an absolute time:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:10:57  2:11:10:57  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m duration=24:00:00 system.1
duration for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:10:50  1:11:10:50  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

Modifying the starttime of a reservation

The following increases the starttime of a reservation using the += tag:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:08:30  2:11:08:30  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m starttime+=24:00:00 system.1
starttime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -  1:11:08:22  3:11:08:22  2:00:00:00    1/2    Sun Nov 19
00:00:00
1 reservation located
```

The following decreases the starttime of a reservation using the -= tag:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:07:04  2:11:07:04  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m starttime-=24:00:00 system.1
starttime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -   -12:53:04  1:11:06:56  2:00:00:00    1/2    Fri Nov 17
00:00:00
1 reservation located
```

The following modifies the starttime of a reservation using an absolute time:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:05:31  2:11:05:31  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m starttime=0_11/19 system.1
starttime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -  1:11:05:18  3:11:05:18  2:00:00:00    1/2    Sun Nov 19
00:00:00
1 reservation located
```

The following modifies the starttime of a reservation using an absolute time:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:04:04  2:11:04:04  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m starttime=0_11/17 system.1
starttime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -   -12:56:02  1:11:03:58  2:00:00:00    1/2    Fri Nov 17
00:00:00
1 reservation located
```

Examples

- **Basic Reservation** on page 266
- **System Maintenance Reservation** on page 266
- **Explicit Task Description** on page 267
- **Dynamic Reservation Modification** on page 267
- **Reservation Modification** on page 267
- **Allocating Reserved Resources** on page 267
- **Modifying an Existing Reservation** on page 267

Example 3-35: Basic Reservation

Reserve two nodes for use by users `john` and `mary` for a period of 8 hours starting in 24 hours

```
> mrsvctl -c -a USER=john,USER=mary -starttime +24:00:00 -duration 8:00:00 -t 2
reservation 'system.1' created
```

Example 3-36: System Maintenance Reservation

Schedule a system wide reservation to allow a system maintenance on Jun 20, 8:00 AM until Jun 22, 5:00 PM.

```
% mrsvctl -c -s 8:00:00_06/20 -e 17:00:00_06/22 -h ALL
reservation 'system.1' created
```

Example 3-37: Explicit Task Description

Reserve one processor and 512 MB of memory on nodes node003 through node node006 for members of the group staff and jobs in the interactive class

```
> mrsvctl -c -R PROCS=1,MEM=512 -a GROUP=staff,CLASS=interactive -h 'node00[3-6]'
reservation 'system.1' created
```

Example 3-38: Dynamic Reservation Modification

Modify reservation john.1 to start in 2 hours, run for 2 hours, and include node02 in the hostlist.

```
> mrsvctl -m starttime=+2:00:00,duration=2:00:00,HostExp+=node02
Note:  hosts added to rsv system.3
```

Example 3-39: Reservation Modification

Remove user John's access to reservation system.1

```
> mrsvctl -m -a USER=John system.1 --flags=unset
successfully changed ACL for rsv system.1
```

Example 3-40: Allocating Reserved Resources

Allocate resources for group dev which are exclusive except for resources found within reservations myrinet.3 or john.6

```
> mrsvctl -c -E -a group=dev,rsv=myrinet.3,rsv=john.6 -h 'node00[3-6]'
reservation 'dev.14' created
```

Create exclusive network reservation on racks 3 and 4

```
> mrsvctl -c -E -a group=ops -g network -f rack3 -h ALL
reservation 'ops.1' created
> mrsvctl -c -E -a group=ops -g network -f rack4 -h ALL
reservation 'ops.2' created
```

Allocate 64 nodes for 2 hours to new reservation and grant access to reservation system.3 and all reservations in the reservation group network

```
> mrsvctl -c -E -d 2:00:00 -a group=dev -t 64 -S rsvaccesslist=system.3,network
reservation 'system.23' created
```

Allocate 4 nodes for 1 hour to new reservation and grant access to idle job reservations

```
> mrsvctl -c -E -d 1:00:00 -t 4 -S rsvaccesslist=idle
reservation 'system.24' created
```

Example 3-41: Modifying an Existing Reservation

Remove user john from reservation ACL

```
> mrsvctl -m -a USER=john system.1 --flags=unset
successfully changed ACL for rsv system.1
```

Change reservation group

```
> mrsvctl -m RSVGROUP=network ops.4
successfully changed RSVGROUP for rsv ops.4
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- Admin Reservation Overview
- showres
- mdiag -r
- mshow -a command to identify available resources
- job to rsv binding

mschedctl

Synopsis

mschedctl -A '<MESSAGE>'

mschedctl -c message *messagestring* [-o *type*:*val*]

mschedctl -c trigger *triggerid* -o *type*:*val*

mschedctl -d trigger:*triggerid*

mschedctl -d message:*index*

mschedctl -f {all|fairshare|usage}

mschedctl -k

mschedctl -l {config|gmetric|gres|message|opsys|trigger|trans} [--flags=verbose] [--xml]

mschedctl -L [*LOGLEVEL*]

mschedctl -m config *string* [-e] [--flags=persistent]

mschedctl -m trigger *triggerid* *attr*=*val*[,*attr*=*val*...]

mschedctl -q mschedctl -q pactions --xml

mschedctl -p

mschedctl -r [*resumetime*]

mschedctl -R

mschedctl -s [*STOPITERATION*]

mschedctl -S [*STEPITERATION*]

Overview

The mschedctl command controls various aspects of scheduling behavior. It is used to manage scheduling activity, shutdown the scheduler, and create resource trace files. It can also evaluate, modify, and create parameters, triggers, and messages.

With many flags, the `--msg=<MSG>` option can be specified to annotate the action in the event log.

Format

-A - ANNOTATE

Format	*<STRING>*
Description	Report the specified parameter modification to the event log and annotate it with the specified message. The **RECORDEVENTLIST** parameter must be set in order for this to work.
Example	`mschedctl -A 'increase logging' -m 'LOGLEVEL 6'` *Adjust the* **LOGLEVEL** *parameter and record an associated message.*

-c - CREATE

Format	One of: • **message** *<STRING>* [-o *<TYPE>*:*<VAL>*] • **trigger**<TRIGSPEC> -o *<OBJECTTYPE>*:*<OBJECTID>* • **gevent** -n *<NAME>* [-m *<message>*] where *<ATTR>* is one of account, duration, ID, messages, profile, reqresources, resources, rsvprofile, starttime, user, or variables
Descrip-tion	Create a message, trigger, or gevent and attach it to the specified object. To create a trigger on a default object, use the Moab configuration file (`moab.cfg`) rather than the mschedctl command.

-c - CREATE	
Example	`mschedctl -c message tell the admin to be nice` *Create a message on the system table.* `mschedctl -c trigger EType=start,AType=exec,Action="/tmp/email $OWNER $TIME" -o rsv:system.1` *Create a trigger linked to* `system.1`. Creating triggers on default objects via `mschedctl -c trigger` does not propagate the triggers to individual objects. To propagate triggers to all objects, the triggers must be created within the `moab.cfg` file; for example: `NODECFG[DEFAULT]TRIGGER`. `mschedctl -c gevent -n diskfailure -m "node=n4"` *Create a gevent indicating a disk failure on the node labeled* `n4`.

-d - DESTROY	
Format	One of: • **trigger**:<*TRIGID*> • **message**:<*INDEX*>
Description	Delete a trigger or message.
Example	`mschedctl -d trigger:3` *Delete trigger 3.* `mschedctl -d message:5` *Delete message with index 5.*

-f - FLUSH

Format	{all|fairshare|usage}
Description	Reset all internally-stored Moab Scheduler statistics to the initial start-up state as of the time the command was executed. Flushing should only be used if you experience corrupt statistics. The best practice is to pause the Moab scheduler with mschedctl -p before running the flush command. After running the flush command, unpause the Moab scheduler with mschedctl -r and the jobs will start flowing again. For all external observers this will be a transparent flush unless they are watching the stats.
Example	`mschedctl -f usage` *Flush usage statistics.*

-k - KILL

Description	Stop scheduling and exit the scheduler
Example	mschedctl -k *Kill the scheduler.*

-l - LIST

Format	{config | gmetric| gres |message|opsys | trans | trigger} [--flags=verbose] [--xml] Using the `--xml` argument with the trans option returns XML that states if the queried TID is valid or not.
Default	config
Description	List the generic metrics, generic resources, scheduler configuration, system messages, operating systems, triggers, or transactions.

-l - LIST	
Example	`mschedctl -l config` *List system parameters.* `mschedctl -l gmetric` *List all configured generic metrics.* `mschedctl -l gres` *List all configured generic resources.* `mschedctl -l message` *List all system messages.* `mschedctl -l opsys` *List all recognized operating systems* `mschedctl -l trans 1` *List transaction id* `1`*.* `mschedctl -l trigger` *List triggers.*

-L - LOG	
Format	*<INTEGER>*
Default	7
Description	Create a temporary log file with the specified loglevel.
Example	`mschedctl -L 7` *Create temporary log file with naming convention* `<logfile>.YYYYMMDDHHMMSS`*.*

-m - MODIFY

Format	One of: • **config** [*<STRING>*] [-e] [--flags=pers] *<STRING>* is any string which would be acceptable in `moab.cfg` ◦ If no string is specified, *<STRING>* is read from STDIN. ◦ If `-e` is specified, the configuration string will be evaluated for correctness but no configuration changes will take place. Any issues with the provided string will be reported to STDERR. ◦ If `--flags=persistent` is specified, the Moab configuration files (`moab.cfg` and `moab.dat`) are modified. • **trigger**:*<TRIGID>* *<ATTR>*=*<VAL>* where *<ATTR>* is one of action, atype, etype, iscomplete, oid, otype, offset, or threshold
Description	Modify a system parameter or trigger.
Example	`mschedctl -m config LOGLEVEL 9` *Change the system loglevel to 9.* `mschedctl -m trigger:2 AType=exec,Offset=200,OID=system.1` *Change aspects of trigger 2.*

-p - PAUSE

Description	Disable scheduling but allow the scheduler to update its cluster and workload state information.
Example	`mschedctl -p`

-q QUERY PENDING ACTIONS

Default	`mschedctl -q pactions --xml`
Description	A way to view pending actions. Only an XML request is valid. Pending actions can be VMs or system jobs.

-q QUERY PENDING ACTIONS

Example	`mschedctl -q pactions --xml`

-R - RECYCLE

Description	Recycle scheduler immediately (shut it down and restart it using the original execution environment and command line arguments).
Example	`mschedctl -R` *Recycle scheduler immediately.* To restart Moab with its last known scheduler state, use: `mschedctl -R savestate`

-r - RESUME

Format	mschedctl -r [[HH:[MM:]]SS]
Default	0
Description	Resume scheduling in the specified amount of time (or immediately if none is specified).
Example	`mschedctl -r` *Resume scheduling immediately.*

-s - STOP

Format	*<INTEGER>*
Default	0
Description	Suspend/stop scheduling at specified iteration (or at the end of the current iteration if none is specified). If the letter I follows *<ITERATION>*, Moab will not process client requests until this iteration is reached.

-s - STOP	
Example	`mschedctl -s 100I` *Stop scheduling at iteration 100 and ignore all client requests until then.*

-S - STEP	
Format	*<INTEGER>*
Default	0
Description	Step the specified number of iterations (or to the next iteration if none is specified) and suspend scheduling If the letter I follows *<ITERATION>*, Moab will not process client requests until this iteration is reached.
Example	`mschedctl -S` *Step to the next iteration and stop scheduling.*

Examples

Example 3-42: Shutting down the Scheduler

```
mschedctl -k
scheduler will be shutdown immediately
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes

mshow

Synopsis

mshow [-a] [-q *jobqueue*=active]

Overview

The mshow command displays various diagnostic messages about the system and job queues.

Arguments

Flag	Description
-a	AVAILABLE RESOURCES
-q [<QUEUENAME>]	Displays the job queues.

Format

AVAILABLE RESOURCES	
Format	Can be combined with `--flags=[tid|verbose|future]` `--format=xml` and/or `-w`
Description	Display available resources.
Example	`> mshow -a -w user=john --flags=tid --format=xml` *Show resources available to john in XML format with a transaction id. See mshow -a for details.*

JOB QUEUE	
Format	*<QUEUENAME>*, where the queue name is one of: active, eligible, or blocked. Job queue names can be delimited by a comma to display multiple queues. If no job queue name is specified, mshow displays all job queues.
Description	Displays the job queues. If a job queue name is specified, mshow shows only that job queue.
Example	`> mshow -q active,blocked` `[Displays all jobs in the active and blocked queues]` `...`

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mshow -a command to show available resources

mshow -a

Synopsis

mshow -a [-i] [-o] [-T] [-w *where*] [-x] [--xml]

Overview

The mshow -a command allows for querying of available system resources.

Arguments

[-i]	INTERSECTION
[-o]	NO AGGREGATE
[-T]	TIMELOCK
[-w]	WHERE
[-x]	EXCLUSIVE

Table 3-4: Argument Format

--flags	
Name	Flags
Format	--flags=[future | policy | tid | summary | verbose]
Description	**future** will return resources available immediately and available in the future. **policy (Deprecated. May be removed in a future release.)** will apply charging policies to determine the total cost of each reported solution (only enabled for XML responses). **summary** will assign all jointly allocated transactions as dependencies of the first transaction reported. **tid** will associate a transaction id with the reported results. **verbose** will return diagnostic information.
Example	`> mshow -a -w user=john --flags=tid --xml` *Show resources available to john in XML format with a transaction ID.*

--xml	
Name	XML
Format	--xml
Description	Report results in XML format.
Example	`> mshow -a -w user=john --flags=tid --xml` *Show resources available to* `john` *in XML format with a transaction ID.*

-i	
Name	INTERSECTION
Description	Specifies that an intersection should be performed during an mshow -a command with multiple requirements.

-o	
Name	NO AGGREGATE
Description	Specifies that the results of the command mshow -a with multiple requirements should not be aggregated together.

-T	
Name	TIMELOCK
Description	Specifies that the multiple requirements of an mshow -a command should be timelocked.
Example	`> mshow -a -w minprocs=1,os=linux,duration=1:00:00 \` `  -w minprocs=1,os=aix,duration=10:00 \` `  --flags=tid,future -x -T`

-w	
Name	WHERE
Format	Comma delimited list of <ATTR>=<VAL> pairs: *<ATTR>=<VAL>* [,*<ATTR>=<VAL>*]... If any of the *<ATTR>=<VAL>* pairs contains a sub-list that is also comma delimited, the entire `-w` string must be wrapped in single quotations with the sub-list expression wrapped in double quotations. See the example below. Attributes are listed below in table 2.
Description	Add a Where clause to the current command (currently supports up to six co-allocation clauses).
Example	`> mshow -a -w minprocs=2,duration=1:00:00 -w nodemem=512,duration=1:00:00` *Moab returns a list of all nodes with at least `2` processors and one hour duration or with a memory of `512` and a duration of one hour.* `> mshow -a -w nodefeature=\!vmware:gpfs --flags=future` *Moab returns a list of all nodes that do not contain the `vmware` feature but that do contain the `gpfs` feature.* `> mshow -a -w 'duration=INFINITY,"excludehostlist=n01,n12,n23"'` *Moab returns a list of all nodes with a duration of INFINITY, except for nodes named n01, n12, and n23.* *Note the use of single quotations containing the entire `-w` string and the use of double quotations containing the **excludehostlist** attribute.*

-x	
Name	EXCLUSIVE
Description	Specifies that the multiple requirements of an mshow -a command should be exclusive (i.e. each node may only be allocated to a single requirement)
Example	`> mshow -a -w minprocs=1,os=linux -w minprocs=1,os=aix --flags=tid -x`

Table 3-5: Request Attributes

Name	Description
account	The account credential of the requestor
acl	ACL to attach to the reservation This ACL must be enclosed in quotation marks. For example: `$ mshow -a ... -w acl=\"user=john\" ...`
arch	Select only nodes with the specified architecture
cal	Select resources subject to the constraints of the specified global calendar
class	The class credential of the requestor
coalloc	The co-allocation group of the specific Where request (can be any string but must match co-allocation group of at least one other Where request) The number of tasks requested in each Where request must be equal whether this taskcount is specified via `minprocs`, `mintasks`, or `gres`.
count	The number of profiles to apply to the resource request
displaymode	Possible value is `future`. (Example: `displaymode=future`). Constrains how results are presented; setting `future` evaluates which resources are available now and which resources will be available in the future that match the requested attributes.
duration	The duration for which the resources will be required in format `[[[DD:]HH:]MM:]SS`
excludehostlist	Do not select any nodes from the given list. The list must be comma delimited. `> mshow -a -w 'duration=INFINITY,"excludehostlist=n01,n12,n23"'` *Moab returns a list of all nodes with a duration of INFINITY, except for nodes named n01, n12, and n23.* *Note the use of single quotations to contain the entire `-w` string, and the use of double quotations containing the **excludehostlist** attribute.*
gres	Select only nodes which possess the specified generic resource
group	The group credential of the requestor

Name	Description
hostlist	Select only the specified resources. The list must be comma delimited. `> mshow -a -w 'duration=INFINITY,"hostlist=n01,n12,n23"'` *Moab returns a list of nodes from the selected hostlist that have a duration of INFINITY.* *Note the use of single quotations to contain the entire `-w` string, and the use of double quotations containing the **hostlist** attribute.*
job	Use the resource, duration, and credential information for the job specified as a resource request template
jobfeature	Select only resources which would allow access to jobs with the specified job features
jobflags	Select only resources which would allow access to jobs with the specified job flags. The `jobflags` attribute accepts a colon delimited list of multiple flags.
label	Removed label 1/2013 for DOC-16 Associate the specified label with all results matching this request
minnodes	Return only results with at least the number of nodes specified. If used with TID's, return only solutions with exactly minnodes nodes available
minprocs	Return only results with at least the number of processors specified. If used with TID's, return only solutions with exactly minprocs processors available
mintasks	**FORMAT**: *<TASKCOUNT>*[@*<RESTYPE>*:*<COUNT>*[+*<RESTYPE>*:*<COUNT>*]...] where *<RESTYPE>* is one of `procs`, `mem`, `disk`, or `swap`. Return only results with at least the number of tasks specified. If used with TID's, return only solutions with exactly mintasks available
nodedisk	Select only nodes with at least nodedisk MB of local disk configured
nodefeature	Select only nodes with all specified features present and nodes without all `\!` specified features using format `[\!]<feature>[:[\!]<feature>]...` You must set the future flag when specifying node features.
nodemem	Select only nodes with at least `nodemem` MB of memory configured
offset	Select only resources which can be co-allocated with the specified time offset where offset is specified in the format `[[[DD:]HH:]MM:]SS`

Name	Description
os	Select only nodes with have, or can be provisioned to have, the specified operating system
partition	The partition in which the resources must be located
policylevel	Enable policy enforcement at the specified policy constraint level
qos	The qos credential of the requestor
rsvprofile	Use the specified profile if committing a resulting transaction id directly to a reservation
starttime	Constrain the timeframe for the returned results by specifying one or more ranges using the format *<STIME>*[-*<ENDTIME>*][;*<STIME>*[-*<ENDTIME>*]] where each time is specified in the format in absolute, relative, or epoch time format (`[HH[:MM[:SS]]][_MO[/DD[/YY]]]` or +`[[[DD:]HH:]MM:]SS` or *<EPOCHTIME>*). The starttime specified is not the exact time at which the returned range must start, but is rather the earliest possible time the range may start.
taskmem	Require `taskmem` MB of memory per task located
tpn	Require exactly `tpn` tasks per node on all discovered resources
user	The user credential of the requestor
var	Use associated variables in generating per transaction charging quotes
variables	Takes a string of the format `variables='var[=attr]'[;'var[=attr]'` and passes the variables onto the reservation when used in conjunction with `--flags=tid` and `mrsvctl -c -R` *<tid>*.
vmusage	Possible value is `vmcreate`. Moab will find resources for the job assuming it is a vmcreate job, and if `os` is also specified, Moab will look for a hypervisor capable of running a VM with the requested OS.

Usage Notes

The mshow -a command allows for querying of available system resources. When combined with the `--flags=tid` option these available resources can then be placed into a packaged reservation (using mrsvctl -c -R). This allows system administrators to grab and reserve available resources for whatever reason, without conflicting with jobs or reservations that may be holding certain resources.

There are a few restrictions on which *<ATTR>* from the -w command can be placed in the same req: `minprocs`, `minnodes`, and `gres` are all mutually exclusive, only one may be used per `-w` request.

The allocation of available nodes will follow the global NODEALLOCATIONPOLICY.

When the '-o' flag is not used, multi-request results will be aggregated. This aggregation will negate the use of offsets and request-specific starttimes.

The config parameter RESOURCEQUERYDEPTH controls the maximum number of options that will be returned in response to a resource query.

Examples

Example 3-43: Basic Compute Node Query and Reservation

```
> mshow -a -w duration=10:00:00,minprocs=1,os=AIX53,jobfeature=shared --
flags=tid,future

Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1      10:00:00      00:00:00  13:28:09_04/27  TID=4  ReqID=0
ALL               1      1      10:00:00      10:00:00  17:14:48_04/28  TID=5  ReqID=0
ALL               1      1      10:00:00      20:00:00  21:01:27_04/29  TID=6  ReqID=0
> mrsvctl -c -R 4
Note:  reservation system.2 created
```

Example 3-44: Mixed Processor and License Query

Select one node with 4 processors and 1 matlab license where the matlab license is only available for the last hour of the reservation. Also, select 16 additional processors which are available during the same timeframe but which can be located anywhere in the cluster. Group the resulting transactions together using transaction dependencies so only the first transaction needs to be committed to reserve all associated resources.

```
> mshow -a -i -o -x -w mintasks=1@PROCS:4,duration=10:00:00,coalloc=a \
                    -w gres=matlab,offset=9:00:00,duration=1:00:00,coalloc=a \
                    -w minprocs=16,duration=10:00:00 --flags=tid,future,summary
Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1      10:00:00      00:00:00  13:28:09_04/27  TID=4  ReqID=0
ALL               1      1      10:00:00      10:00:00  17:14:48_04/28  TID=5  ReqID=0
ALL               1      1      10:00:00      20:00:00  21:01:27_04/29  TID=6  ReqID=0
> mrsvctl -c -R 4

Note:  reservation system.2 created
Note:  reservation system.3 created
Note:  reservation system.4 created
```

Example 3-45: Request for Generic Resources

Query for a generic resource on a specific host (no processors, only a generic resource).

```
> mshow -a -i -x -o -w gres=dvd,duration=10:00,hostlist=node03 --flags=tid,future
Partition     Tasks  Nodes   StartOffset      Duration       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1      00:00:00      00:10:00  11:33:25_07/27  TID=16
ReqID=0
ALL               1      1      00:10:00      00:10:00  11:43:25_07/27  TID=17
ReqID=0
ALL               1      1      00:20:00      00:10:00  11:53:25_07/27  TID=18
ReqID=0
> mrsvctl -c -R 16
Note:  reservation system.6 created
> mdiag -r system.6
Diagnosing Reservations
RsvID                      Type Par   StartTime     EndTime     Duration Node Task
Proc
-----                      ---- ---   ---------     -------     -------- ---- ---- ---
-
system.6                   User loc   -00:01:02    00:08:35     00:09:37    1    1
0
    Flags: ISCLOSED
    ACL:   RSV==system.6=
    CL:    RSV==system.6
    Accounting Creds:  User:test
    Task Resources: dvd: 1
    Attributes (HostExp='^node03$')
    Rsv-Group: system.6
```

Example 3-46: Allocation of Shared Resources

This example walks through a relatively complicated example in which a set of resources can be reserved to be allocated for shared requests. In the example below, the first mshow query looks for resources within an existing shared reservation. In the example, this first query fails because there is now existing reservation. The second query looks for resources within an existing shared reservation. In the example, this first query fails because there is now existing reservation. The second mshow request asks for resources outside of a shared reservation and finds the desired resources. These resources are then reserved as a shared pool. The third mshow request again asks for resources inside of a shared reservation and this time finds the desired resources.

```
> mshow -a -w duration=10:00:00,minprocs=1,os=AIX53,jobflags=ADVRES,jobfeature=shared
--flags=tid

Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
> mshow -a -w duration=100:00:00,minprocs=1,os=AIX53,jobfeature=shared --flags=tid
Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1     100:00:00      00:00:00  13:20:23_04/27  TID=1  ReqID=0
> mrsvctl -c -R 1
Note:  reservation system.1 created
> mshow -a -w duration=10:00:00,minprocs=1,os=AIX53,jobflags=ADVRES,jobfeature=shared
--flags=tid
Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1      10:00:00      00:00:00  13:20:36_04/27  TID=2  ReqID=0
> mrsvctl -c -R 2
Note:  reservation system.2 created
```

Example 3-47: Full Resource Query in XML Format

The following command will report information on all available resources which meet at least the minimum specified processor and walltime constraints and which are available to the specified user. The results will be reported in XML to allow for easy system processing.

```
> mshow -a -w class=grid,minprocs=8,duration=20:00 --format=xml --flags=future,verbose

<Data>
  <Object>cluster</Object>
  <job User="john" time="1162407604"></job>
  <par Name="template">
    <range duration="Duration" nodecount="Nodes" proccount="Procs"
starttime="StartTime"></range>
    </par>
  <par Name="ALL" feasibleNodeCount="131" feasibleTaskCount="163">
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
041:1,opt-042:1,x86-001:1,P690-001:1,P690-021:1,P690-022:1"
        index="0" nodecount="10" proccount="8" reqid="0"
starttime="1162407604"></range>
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
039:1,opt-041:1,opt-042:1,x86-001:1,P690-001:1,P690-021:1,P690-022:1"
        index="0" nodecount="11" proccount="8"reqid="0"
starttime="1162411204"></range>
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
039:1,opt-041:1,opt-042:1,x86-001:1,x86-002:1,x86-004:1,
        x86-006:1,x86-013:1,x86-014:1,x86-015:1,x86-016:1,x86-037:1,P690-001:1,P690-
021:1,P690-022:1"
        index="0" nodecount="19" proccount="8" reqid="0"
starttime="1162425519"></range>
    </par>
  <par Name="SharedMem">
    <range duration="1200" hostlist="P690-001:1,P690-002:1,P690-003:1,P690-004:1,P690-
005:1,P690-006:1,P690-007:1,P690-008:1,P690-009:1,
        P690-010:1,P690-011:1,P690-012:1,P690-013:1,P690-014:1,P690-015:1,P690-
016:1,P690-017:1,P690-018:1,P690-019:1,P690-020:1,P690-021:1,
        P690-022:1,P690-023:1,P690-024:1,P690-025:1,P690-026:1,P690-027:1,P690-
028:1,P690-029:1,P690-030:1,P690-031:1,P690-032:1"
        index="0" nodecount="32" proccount="8" reqid="0"
starttime="1163122507"></range>
    </par>
  <par Name="64Bit">
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
039:1,opt-041:1,opt-042:1"
        index="0" nodecount="7" proccount="8" reqid="0"
starttime="1162411204"></range>
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
039:1,opt-041:1,opt-042:1,opt-043:1,opt-044:1,opt-045:1,
        opt-046:1,opt-047:1,opt-048:1,opt-049:1,opt-050:1"
        index="0" nodecount="15" proccount="8" reqid="0"
starttime="1162428996"></range>
    <range duration="1200" hostlist="opt-001:1,opt-006:1,opt-007:2,opt-008:2,opt-
009:2,opt-010:2,opt-011:2,opt-012:2,opt-013:2,opt-014:2,
        opt-015:2,opt-016:2,opt-017:2,opt-018:2,opt-019:2,opt-020:2,opt-021:2,opt-
022:2,opt-023:2,opt-024:2,opt-025:1,opt-027:2,opt-039:1,
        opt-041:1,opt-042:1,opt-043:1,opt-044:1,opt-045:1,opt-046:1,opt-047:1,opt-
048:1,opt-049:1,opt-050:1"
        index="0" nodecount="33" proccount="8" reqid="0"
starttime="1162876617"></range>
    </par>
  <par Name="32Bit">
    <range duration="1200" hostlist="x86-001:1,x86-002:1,x86-004:1,x86-006:1,x86-
013:1,x86-014:1,x86-015:1,x86-016:1,x86-037:1"
        index="0" nodecount="9" proccount="8" reqid="0"
starttime="1162425519"></range>
    <range duration="1200" hostlist="x86-001:1,x86-002:1,x86-004:1,x86-006:1,x86-
013:1,x86-014:1,x86-015:1,x86-016:1,x86-037:1,x86-042:1,x86-043:1"
        index="0" nodecount="11" proccount="8" reqid="0"
```

```
starttime="1162956803"></range>
    <range duration="1200" hostlist="x86-001:1,x86-002:1,x86-004:1,x86-006:1,x86-
013:1,x86-014:1,x86-015:1,x86-016:1,x86-027:1,x86-028:1,
        x86-029:1,x86-030:1,x86-037:1,x86-041:1,x86-042:1,x86-043:1,x86-046:1,x86-
047:1,x86-048:1,x86-049:1"
        index="0" nodecount="20" proccount="8" reqid="0"
starttime="1163053393"></range>
    </par>
  </Data>
```


This command reports the original query, and the timeframe, resource size, and hostlist associated with each possible time slot.

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mshow in a hosting environment

mshow -a

Basic Current and Future Requests

The mshow command can report information on many aspects of the scheduling environment. To request information on available resources, the -a flag should be used. By default, the mshow command resource availability query only reports resources that are immediately available. To request information on specific resources, the type of resources required can be specified using the -w flag as in the following example:

```
> mshow -a -w taskmem=1500,duration=600
...
```

To view current and future resource availability, the future flag should be set as in the following example:

```
> mshow -a -w taskmem=1500,duration=600 --flags=future
...
```

Co-allocation Resources Queries

In many cases, a particular request will need simultaneous access to resources of different types. The mshow command supports a co-allocation request specified by using multiple -w arguments. For example, to request 16 nodes with feature fastcpu and 2 nodes with feature fastio, the following request might be used:

```
> mshow -a -w minprocs=16,duration=1:00:00,nodefeature=fastcpu -w
minprocs=2,nodefeature=fastio,duration=1:00:00 --flags=future
Partition     Procs  Nodes   StartOffset      Duration       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL              16      8      00:00:00       1:00:00  13:00:18_08/25  ReqID=0
ALL               2      1      00:00:00       1:00:00  13:00:18_08/25  ReqID=1
```

The mshow -a documentation contains a list of the different resources that may be queried as well as examples on using mshow.

Using Transaction IDs

By default, the mshow command reports simply when and where the requested resources are available. However, when the `tid` flag is specified, the mshow command returns both resource availability information and a handle to these resources called a Transaction ID as in the following example:

```
> mshow -a -w minprocs=16,nodefeature=fastcpu,duration=2:00:00 --flags=future,tid
Partition     Procs  Nodes   StartOffset      Duration       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL              16     16      00:00:00       2:00:00  13:00:18_08/25  TID=26 ReqID=0
```

In the preceding example, the returned transaction id (TID) may then be used to reserve the available resources using the mrsvctl -c -R command:

```
> mrsvctl -c -R 26
reservation system.1 successfully created
```

Any TID can be printed out using the mschedctl -l trans command:

```
Code example (replace with your own content)
```

> mschedctl -l trans 26 TID[26] A1='node01' A2='600' A3='1093465728' A4='ADVRES' A5='fastio'

Where `A1` is the hostlist, `A2` is the duration, `A3` is the starttime, `A4` are any flags, and `A5` are any features.

Using Reservation Profiles

Reservation profiles (RSVPROFILE) stand as templates against which reservations can be created. They can contain a hostlist, startime, endtime, duration, access-control list, flags, triggers, variables, and most other attributes of an Administrative Reservation. The following example illustrates how to create a reservation with the exact same trigger-set.

```
-----
# moab.cfg
-----
RSVPROFILE[test1] TRIGGER=Sets=$Var1.$Var2.$Var3.!Net,EType=start,AType=exec,
        Action=/tmp/host/triggers/Net.sh,
        Timeout=1:00:00
RSVPROFILE[test1]                                  TRIGGER=Requires=$Var1.$Var2.$Var3,
        Sets=$Var4.$Var5,EType=start,
        AType=exec,Action=/tmp/host/triggers/
        FS.sh+$Var1:$Var2:$Var3,Timeout=20:00
RSVPROFILE[test1]
TRIGGER=Requires=$Var1.$Var2.$Var3.$Var4.$Var5,
        Sets=!NOOSinit.OSinit,Etype=start,
        AType=exec,
        Action=/tmp/host/triggers/
        OS.sh+$Var1:$Var2:$Var3:$Var4:$Var5
RSVPROFILE[test1]
TRIGGER=Requires=NOOSini,AType=cancel,EType=start
RSVPROFILE[test1]
TRIGGER=EType=start,Requires=OSinit,AType=exec,
        Action=/tmp/host/triggers/success.sh
...
-----
```

To create a reservation with this profile the mrsvctl -c -P command is used:

```
> mrsvctl -c -P test1

reservation system.1 successfully created
```

Using Reservation Groups

Reservation groups are a way for Moab to tie reservations together. When a reservation is created using multiple Transaction IDs, these transactions and their resulting reservations are tied together into one group.

```
> mrsvctl -c -R 34,35,36
reservation system.99 successfully created
reservation system.100 successfully created
reservation system.101 successfully created
```

In the preceding example, these three reservations would be tied together into a single group. The mdiag -r command can be used to see which group a reservation belongs to. The mrsvctl -q diag -g command can also be used to print out a specific group of reservations. The mrsvctl -c -g command can also be used to release a group of reservations.

Related topics

- mshow

msub

Synopsis

msub [-a *datetime*][-A *account*][-c interval][-C directive_prefix][-d *path*] [-e *path*][-E][-F][-h][-I][-j *join*][-k keep][-K][-l *resourcelist*][-m *mailoptions*] [-M user_list][-N name][-o *path*][-p *priority*][-q *destination*][-r] [-S *pathlist*][-t *jobarrays*][-u *userlist*][-v *variablelist*][-V] [-W *additionalattributes*][-x][-z][--stagein][--stageout][--stageinfile][--stageoutfile][--stageinsize][--stageoutsize][--workflowjobids][*script*]

Overview

msub allows users to submit jobs directly to Moab. When a job is submitted directly to a resource manager (such as TORQUE), it is constrained to run on only those nodes that the resource manager is directly monitoring. In many instances, a site may be controlling multiple resource managers. When a job is submitted to Moab rather than to a specific resource manager, it is not constrained as to what nodes it is executed on. msub can accept command line arguments (with the same syntax as qsub), job scripts (in either PBS or LoadLeveler syntax), or the SSS Job XML specification.

> Moab must run as a root user in order for msub submissions to work. Workload submitted via msub when Moab is running as a non-root user fail immediately.

Submitted jobs can then be viewed and controlled via the mjobctl command.

> Flags specified in the following table are not necessarily supported by all resource managers.

Access

When Moab is configured to run as root, any user may submit jobs via msub.

Flags

-a	
Name	Eligible Date
Format	[[[[CC]YY]MM]DD]hhmm[.SS]
Description	Declares the time after which the job is eligible for execution.
Example	`> msub -a 12041300 cmd.pbs` *Moab will not schedule the job until 1:00 pm on December 4, of the current year.*

-A	
Name	Account
Format	*<ACCOUNT NAME>*
Description	Defines the account associated with the job.
Example	`> msub -A research cmd.pbs` *Moab will associate this job with account research.*

-c	
Name	Checkpoint Interval
Format	[n|s|c|c=*<minutes>*]
Description	Checkpoint of the will occur at the specified interval. **n** — No Checkpoint is to be performed. **s** — Checkpointing is to be performed only when the server executing the job is shut down. **c** — Checkpoint is to be performed at the default minimum time for the server executing the job. **c=*<minutes>*** — Checkpoint is to be performed at an interval of minutes.
Example	`> msub -c c=12 cmd.pbs` *The job will be checkpointed every* `12` *minutes.*

-C	
Name	Directive Prefix
Format	'*<PREFIX NAME>*'
Default	First known prefix (#PBS, #@, #BSUB, #!, #MOAB, #MSUB)

-C	
Description	Specifies which directive prefix should be used from a job script. • It is best to submit with single quotes. '#PBS' • An empty prefix will cause Moab to not search for any prefix. -C '' • Command line arguments have precedence over script arguments. • Custom prefixes can be used with the -C flag. -C '#MYPREFIX' • Custom directive prefixes must use PBS syntax. • If the -C flag is not given, Moab will take the first default prefix found. Once a directive is found, others are ignored.
Example	`> msub -C '#MYPREFIX' cmd.pbs` `#MYPREFIX -l walltime=5:00:00 (in cmd.pbs)` *Moab will use the #MYPREFIX directive specified in cmd.pbs, setting the wallclock limit to five hours.*

-d	
Name	Execution Directory
Format	*<path>*
Default	Depends on the RM being used. If using TORQUE, the default is $HOME. If using SLURM, the default is the submission directory.
Description	Specifies which directory the job should execute in.
Example	`> msub -d /home/test/job12 cmd.pbs` *The job will begin execution in the* `/home/test/job12` *directory.*

-e	
Name	Error Path
Format	[*<hostname>*:]*<path>*
Default	`$SUBMISSIONDIR/$JOBNAME.e$JOBID`

-e	
Description	Defines the path to be used for the standard error stream of the batch job.
Example	`> msub -e test12/stderr.txt` *The STDERR stream of the job will be placed in the relative (to execution) directory specified.*

-E	
Name	Environment Variables
Description	Moab adds the following variables, if populated, to the job's environment: • MOAB_ACCOUNT — Account name. • MOAB_BATCH — Set if a batch job (non-interactive). • MOAB_CLASS — Class name. • MOAB_DEPEND — Job dependency string. • MOAB_GROUP — Group name. • MOAB_JOBARRAYINDEX —For a job in an array, the index of the job. • MOAB_JOBARRAYRANGE — For a system with job arrays, the range of all job arrays. • MOAB_JOBID — Job ID. If submitted from the grid, grid jobid. • MOAB_JOBNAME — Job name. • MOAB_MACHINE — Name of the machine (i.e. Destination RM) that the job is running on. • MOAB_NODECOUNT — Number of nodes allocated to job. • MOAB_NODELIST — Comma-separated list of nodes (listed singly with no ppn info). • MOAB_PARTITION — Partition name the job is running in. If grid job, cluster scheduler's name. • MOAB_PROCCOUNT — Number of processors allocated to job. • MOAB_QOS — QOS name. • MOAB_TASKMAP — Node list with procs per node listed. <nodename>.<procs> • MOAB_USER — User name. In SLURM environments, not all variables will be populated since the variables are added at submission (such as NODELIST). With TORQUE/PBS, the variables are added just before the job is started. This feature only works with SLURM and TORQUE/PBS.
Example:	`> msub -E mySim.cmd` *The job `mySim` will be submitted with extra environment variables.*

-F	
Name	Script Flags
Format	"\"*<STRING>*\""
Description	Specifies the flags TORQUE will pass to the job script at execution time. The `-F` flag is only compatible with TORQUE resource managers.
Example	`> msub -F "\"arg1 arg2\"" -l nodes=1,walltime=60 files/job.sh` *TORQUE will pass parameters `arg1` and `arg2` to the `job.sh` script when the job executes.*

-h	
Name	Hold
Description	Specifies that a user hold be applied to the job at submission time.
Example	`> msub -h cmd.ll` *The job will be submitted with a user hold on it.*

-I	
Name	Interactive
Description	Declares the job is to be run interactively. qsub must exist on the same host as msub if the interactive job is destined for a TORQUE cluster, because the interactive msub request will be converted to a qsub -I request.
Example	`> msub -I job117.sh` *The job will be submitted in interactive mode.*

-j	
Name	Join
Format	[eo|oe|n]
Default	n (not merged)
Description	If `eo` is specified, the error and output streams are merged into the *error* stream. If `oe` is specified, the error and output streams will be merged into the *output* stream. If using either the -e or the -o option and the `-j eo|oe` option, the `-j` option takes precedence and all standard error and output messages go to the chosen output file.
Example	`> msub -j oe cmd.sh` *STDOUT and STDERR will be merged into one file.*

-k	
Name	Keep
Format	[e|o|eo|oe|n]
Default	n (not retained)
Description	Defines which (if either) of output and error streams will be retained on the execution host (overrides path for stream).
Example	`> msub -k oe myjob.sh` *STDOUT and STDERR for the job will be retained on the execution host.*

-K	
Name	Continue Running
Format	N/A

-K	
Description	Tells the client to continue running until the submitted job is completed. The client will query the status of the job every 5 seconds. The time interval between queries can be specified or disabled via MSUBQUERYINTERVAL. Use the `-K` option sparingly (if at all) as it slows down the Moab scheduler with frequent queries. Running ten jobs with the `-K` option creates an additional fifty queries per minute for the scheduler.
Example	`> msub -K newjob.sh` `3` `Job 3 completed*` **Only shows up after job completion.*

-l	
Name	Resource List
Format	*<STRING>* -l [BANDWIDTH|DDISK|DEADLINE|DEPEND|DMEM|EXCLUDENODES|FEATURE...|] Additional options can be referenced on the resource manager extensions page.
Description	Defines the resources that are required by the job and establishes a limit to the amount of resource that can be consumed. Resources native to the resource manager, scheduler resource manager extensions, or job flags may be specified. Note that resource lists are dependent on the resource manager in use. For information on specifying multiple types of resources for allocation, see Multi-Req Support.

-l	
Example	```> msub -l nodes=32:ppn=2,pmem=1800mb,walltime=3600,VAR=testvar:myvalue cmd.sh``` ```> msub -l nodes=32:ppn=2,pmem=1800mb,walltime=3600,VAR=testvar: myvalue cmd.sh``` *The job requires `32` nodes with `2` processors each, `1800` MB per task, a walltime of `3600` seconds, and a variable named `testvar` with a value of `myvalue`.* If JOBNODEMATCHPOLICY is not set, Moab does not reserve the requested number of processors on the requested number of nodes. It reserves the total number of requested processors (nodes x ppn) on any number of nodes. Rather than setting `nodes=<value>:ppn=<value>`, set `procs=<value>`, replacing *<value>* with the total number of processors the job requires. Note that **JOBNODEMATCHPOLICY** is not set by default. ```> msub -l nodes=32:ppn=2 -l advres=!<resvid>``` *This entry would tell Moab to only consider resources outside of the specified <reservation id>.*

-m	
Name	Mail Options
Format	*<STRING>* (either `n` or one or more of the characters `a`, `b`, and `e`)
Description	Defines the set of conditions (abort,begin,end) when the server will send a mail message about the job to the user.
Example	```> msub -m be cmd.sh``` *Mail notifications will be sent when the job begins and ends.*

-M	
Name	Mail List
Format	*<user>*[@*<host>*][,*<user>*[@*<host>*],...]
Default	$JOBOWNER

-M	
Description	Specifies the list of users to whom mail is sent by the execution server. Overrides the **EMAILADDRESS** specified on the USERCFG credential.
Example	`> msub -M jon@node01,bill@node01,jill@node02 cmd.sh` *Mail will be sent to the specified users if the job is aborted.*

-N	
Name	Name
Format	*<STRING>*
Default	STDIN or name of job script
Description	Specifies the user-specified job name attribute.
Example	`> msub -N chemjob3 cmd.sh` *Job will be associated with the name `chemjob3`.*

-o	
Name	Output Path
Format	[*<hostname>*:]*<path>* - %J and %I are acceptable variables. %J is the master array name and %I is the array member index in the array.
Default	`$SUBMISSIONDIR/$JOBNAME.o$JOBID`

-o	
Description:	Defines the path to be used for the standard output stream of the batch job. More variables are allowed when they are used in the job script instead of msub -o. In the job script, specify a `#PBS -o` line and input your desired variables. The allowable variables are: • OID • OTYPE • USER • OWNER • JOBID • JOBNAME Submitting a job script that has the line `#PBS -o $(USER)_$(JOBID)_$(JOBNAME).txt` results in a file called *<username>_<jobID>_<jobName>*.txt. Do not use msub -o when submitting a job script that has a `#PBS -o` line defined.
Example	`> msub -o test12/stdout.txt` *The STDOUT stream of the job will be placed in the relative (to execution) directory specified.* `> msub -t 1-2 -o /home/jsmith/simulations/%J-%I.out ~/sim5.sh` *A job array is submitted and the name of the output files includes the master array index and the array member index.*

-p	
Name	Priority
Format	*<INTEGER>* (between -1024 and 0)
Default	0
Description	Defines the priority of the job. To enable priority range from -1024 to +1023, see ENABLEPOSUSERPRIORITY.
Example	`> msub -p 25 cmd.sh` *The job will have a user priority of 25.*

-q	
Name	Destination Queue (Class)
Format	[<*queue*>][@<*server*>]
Default	[<*DEFAULT*>]
Description	Defines the destination of the job.
Example	`> msub -q priority cmd.sh` *The job will be submitted to the priority queue.*

-r	
Name	Rerunable
Format	[y|n]
Default	n
Description:	Declares whether the job is rerunable.
Example	`> msub -r n cmd.sh` *The job cannot be rerun.*

-S	
Name	Shell
Format	<*path*>[@<*host*>][,<*path*>[@<*host*>],...]
Default	$SHELL
Description	Declares the shell that interprets the job script.

-S	
Example	`> msub -S /bin/bash` *The job script will be interpreted by the* `/bin/bash` *shell.*

-t	
Name	Job Arrays
Format	*<name>*[*<indexlist>*]%*<limit>*
Description	Starts a job array with the jobs in the index list. The limit variable specifies how many jobs may run at a time. For more information, see Submitting Job Arrays. Moab enforces an internal limit of 100,000 sub-jobs that a single array job submission can specify.
Example	`> msub -t myarray[1-1000]%4`

-u	
Name	User List
Format	*<user>*[@*<host>*][,*<user>*[@*<host>*],...]
Default	UID of msub command
Description	Defines the user name under which the job is to run on the execution system.
Example	`> msub -u bill@node01 cmd.sh` *On* `node01` *the job will run under* `Bill`*'s UID, if permitted.*

-v	
Name	Variable List

-v	
Format	*<string>*[,*<string>*,...]
Description	Expands the list the environment variables that are exported to the job (taken from the msub command environment).
Example	`> msub -v DEBUG cmd.sh` *The `DEBUG` environment variable will be defined for the job.*

-V	
Name	All Variables
Description	Declares that all environment variables in the msub environment are exported to the batch job
Example	`> msub -V cmd.sh` *All environment variables will be exported to the job.*

-W	
Name	Additional Attributes
Format	*<string>*
Description	Allows for specification of additional job attributes (See Resource Manager Extension)
Example	`> msub -W x=GRES:matlab:1 cmd.sh` *The job requires one resource of `matlab`.* This flag can be used to set a filter for what namespaces will be passed from a job to a trigger using a comma-delimited list. This limits the trigger's action to objects contained in certain workflows. For more information, see **Requesting name space variables** on page 757. `> msub -W x="trigns=vc1,vc2"` *The job passes namespaces `vc1` and `vc2` to triggers.*

-x	
Format	*<script>* or *<command>*
Description	When running an interactive job, the `-x` flag makes it so that the corresponding script won't be parsed for PBS directives, but is instead a command that is launched once the interactive job has started. The job terminates at the completion of this command. This option works only when using TORQUE. The `-x` option for msub differs from qsub in that qsub does not require the script name to come directly after the flag. The msub command requires a script or command immediately after the `-x` declaration.
Example	`> msub -I -x ./script.pl` `> msub -I -x /tmp/command`

-z	
Name	Silent Mode
Description	The job's identifier will not be printed to stdout upon submission.
Example	`> msub -z cmd.sh` *No job identifier will be printout the stdout upon successful submission.*

Staging data

Data staging, or the ability to copy data required for a job from one location to another or to copy resulting data to a new location (See **About data staging** on page 880 for more information), must be specified at job submission. To stage data in, you would use the msub `--stagein` and/or `--stageinfile` option, optionally with `--stageinsize`. You would use similar options the same way for staging out: `--stageout`, `--stageoutfile`, and `--stageoutsize`. `--stagein` and `--stageout`, which you can use multiple times in the same msub command, allow you to specify a single file or directory to stage in or out. `--stageinfile` and `--stageoutfile` allow you to specify a text file that lists the files to stage in or out. The `--stageinsize` and [`--stageoutsize`] options allow you to estimate the total size of the files and directories that you want to stage in or out, which can help Moab make an intelligent guess about how long it will take to stage the data in or out, thus ensuring that the job can start as soon as possible after the staging has occurred.

Staging a file or directory

The `--stagein` and `--stageout` options use the same format.

`--`*`<stagein|stageout>`*`<=| >`*`<source>`*`%`*`<destination>`*

Where *<source>* and *<destination>* take on the following format:

`[<user>@]<host>:/<path>[/<fileName>]`

Specifying a user and file name are optional. If you do not specify a file name, Moab will assume a directory.

```
> msub ... --stagein=student@biology:/stats/file001%admin@moab:/tmp/staging
<jobScript>
```

This msub commands tells Moab that the job requires `file001` from `student`'s `stats` directory on the `biology` server to be staged to `admin`'s `staging` directory on the `moab` server prior to the job's starting.

You can specify the option multiple times for the same msub command; however, staging large number of files is easier with `--stageinfile` or `--stageoutfile`.

You can also use `#MSUB` or `#PBS` within a job script to specify data staging options. For example:

```
#MSUB --stageinsize=1gb
#MSUB --stagein=...
```

See **Sample user job script** on page 901 for more information. Note that the data staging options are not compatible with qsub.

Staging multiple files or directories

The `--stageinfile` and `--stageoutfile` options use the same format. You must include the path to a text file that lists each file to stage in or out on its own line. Each file specification follows the same format as a `--stagein` or `--stageout` specification as described above. The format of the command options looks like this:

`--<stageinfile|stageoutfile><=| ><path>/<fileName>`

The file contains multiple lines with the following format:

`[<user>@]<host>:/<path>[/<fileName>]%[<user>@]<host>:/<path>[/<fileName>]`

`...`

Moab ignores blank lines in the file. You can comment out lines by preceding them with a pound sign (#). The following examples demonstrate what the `--stageinfile` option looks like on the command line and what the file it specifies might look like.

```
> msub ... --stageinfile=/tmp/myStagingFile <jobScript>
```

`/tmp/myStagingFile`:

```
student@biology:/stats/file001%moab:/tmp/staging
student@biology:/stats/file002%moab:/tmp/staging
student@biology:/stats/file003%moab:/tmp/staging
#student@biology:/stats/file004%moab:/tmp/staging
student@biology:/stats/file005%moab:/tmp/staging

student@biology:/stats/file006%moab:/tmp/staging
student@biology:/stats/file007%moab:/tmp/staging
student@biology:/stats/file008%moab:/tmp/staging
student@biology:/stats/file009%moab:/tmp/staging
student@biology:/stats/file010%moab:/tmp/staging
```

Moab stages in each file listed in `myStagingFile` to the `/tmp/staging` directory. Each file resides on the `biology` host as the `student` user. Moab ignores the blank line and the line specifying `file004`.

Stage in or out file size

The optional `--stageinsize` and `--stageoutsize` options give you the opportunity to estimate the size of the file(s) or directory(-ies) being staged to aid Moab in choosing an appropriate start time. Both options use the same format:

`--<stageinsize|stageoutsize>=<integer>[unit]`

The integer indicates the size of the file(s) and directory(-ies) in megabytes unless you specify a different unit. Moab accepts the follow case-insensitive suffixes: `KB`, `MB`, `GB`, or `TB`.

```
> msub --stageinfile=/stats/file003 --stageinsize=100 <jobScript>
```

Moab copies the `/davidharris/research/recordlist` file, which is approximately 100 megabytes, from the biology node to the host where the job will run prior to job start.

```
> msub --stageinfile=/stats/file002 --stageinsize=1gb <jobScript>
```

Moab copies all files specified in the `/davidharris/research/recordlist` file, which add up to approximately 1 gigabyte, to the host where the job will run prior to job start.

Return all the job IDs in the workflow at submission time

By default, msub will print the job ID to stdout at the time of submission. If you want msub to print all of the jobs that are created as part of the workflow template, you can use the msub `--workflowjobids` option to show all the job IDs at submission time:

```
$ echo sleep 60 | msub -l walltime=15 --workflowjobids

MoabA.3.dsin MoabA.3 MoabA.3.dsout
```

Job Script

The msub command supports job scripts written in any one of the following languages:

Language	Notes
PBS/TORQUE Job Submission Language	---

Language	Notes
LoadLeveler Job Submission Language	Use the INSTANTSTAGE parameter as only a subset of the command file keywords are interpreted by Moab.
SSS XML Job Object Specification	---
LSF Job Submission Language	enabled in Moab 4.2.4 and higher

/etc/msubrc

Sites that wish to automatically add parameters to every job submission can populate the file `/etc/msubrc` with global parameters that every job submission will inherit.

For example, if a site wished every job to request a particular generic resource they could use the following `/etc/msubrc`:

```
-W x=GRES:matlab:2
```

Usage Notes

msub is designed to be as flexible as possible, allowing users accustomed to PBS, LSF, or LoadLeveler syntax, to continue submitting jobs as they normally would. It is not recommended that different styles be mixed together in the same msub command.

When only one resource manager is configured inside of Moab, all jobs are immediately staged to the only resource manager available. However, when multiple resource managers are configured Moab will determine which resource manager can run the job soonest. Once this has been determined, Moab will stage the job to the resource manager.

It is possible to have Moab take a best effort approach at submission time using the `forward` flag. When this flag is specified, Moab will do a quick check and make an intelligent guess as to which resource manager can run the job soonest and then immediately stage the job.

Moab can be configured to instantly stage a job to the underlying resource manager (like TORQUE/LOADLEVELER) through the parameter INSTANTSTAGE. When set inside `moab.cfg`, Moab will migrate the job instantly to an appropriate resource manager. Once migrated, Moab will destroy all knowledge of the job and refresh itself based on the information given to it from the underlying resource manager.

In most instances Moab can determine what syntax style the job belongs to (PBS or LoadLeveler); if Moab is unable to make a guess, it will default the style to whatever resource manager was configured at compile time. If LoadLeveler and PBS were both compiled then LoadLeveler takes precedence.

Moab can translate a subset of job attributes from one syntax to another. It is therefore possible to submit a PBS style job to a LoadLeveler resource manager, and vice versa, though not all job attributes will be translated.

Examples

Example 3-48:

```
> msub -l nodes=3:ppn=2,walltime=1:00:00,pmem=100kb script2.pbs.cmd
4364.orion
```

Example 3-49:

This example is the XML-formatted version of the above example. See Submitting Jobs via msub in XML for more information.

```
<job>
  <InitialWorkingDirectory>/home/user/test/perlAPI
  </InitialWorkingDirectory>
  <Executable>/home/user/test/perlAPI/script2.pbs.cmd
  </Executable>
  <SubmitLanguage>PBS</SubmitLanguage>
  <Requested>
    <Feature>ppn2</Feature>
    <Processors>3</Processors>
    <WallclockDuration>3600</WallclockDuration>
  </Requested>
</job>
```

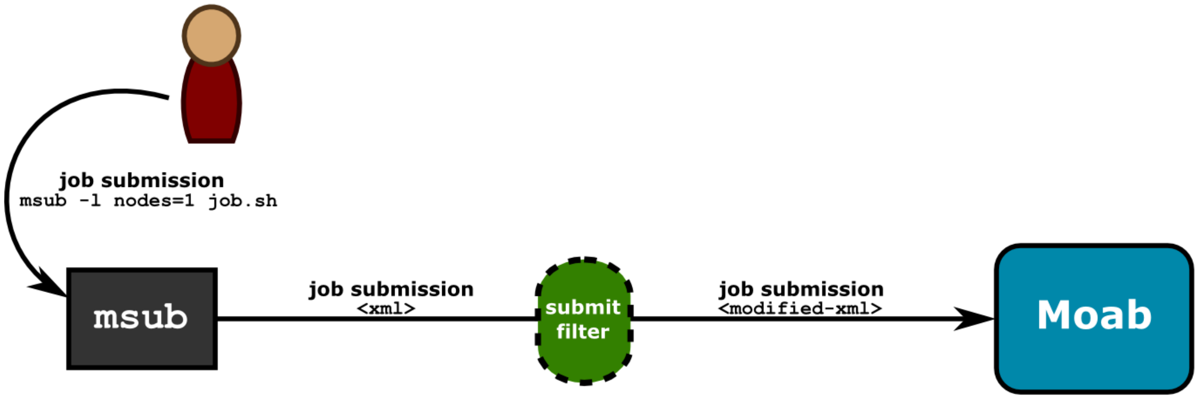
Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mjobctl command to view, modify, and cancel jobs
- checkjob command to view detailed information about the job
- mshow command to view all jobs in the queue
- DEFAULTSUBMITLANGUAGE parameter
- MSUBQUERYINTERVAL parameter
- SUBMITFILTER parameter
- Applying the msub Submit Filter for job script sample

Applying the msub submit filter

When you use msub to submit a job, msub processes the input, converts it to XML, and sends the job specification XML to the Moab scheduler. You can create a submission filter to modify the job XML based on the criteria you set before Moab receives and processes it.

Image 3-1: Job submission process



The filter gives you the ability to customize the submission process, which is helpful if jobs should have certain defaults assigned to them, if you want to keep detailed submission statistics, or if you want to change job requests based on custom needs.

The submit filter, is a simple executable or script that receives XML via its standard input and returns the modified XML in its standard output. It modifies the attributes of the job specification XML based on policies you specify. It can perform various other actions at your request, too; for instance, logging. Once the submit filter has modified the job XML based on your criteria, it writes the XML representing the actual job submission to stdout. The new XML could potentially match the original XML, depending on whether the job met the criteria for modification set in the job submit filter script. Job submissions you want to proceed will leave the filter with an exit code of 0 and continue to Moab for scheduling. If the job meets the filter's specified criteria for rejection, it exits with a non-zero value, aborting the job submission process. You can configure the filter script to write a descriptive rejection message to stderr.

Job submit filters follow these rejection rules: 1) msub will reject job XML with an exit code of anything other than zero, 2) the msub command displays filter's error output on the command line, 3) msub will reject the job if the filter outputs invalid job XML, and 4) msubwill reject the job if it violates any policies in your general Moab configuration; you cannot use a submit filter to bypass other policies.

To see the schema for job submission XML, please refer to Submitting Jobs via msub in XML.

Submit filter types

You can implement submit filters on either the client or server side of a job submission. The primary differences between the two submit filter types are the location from which the filter runs, the powers and privileges of the user running the filter, and whether a user can bypass the filter. Client-based submit filters run from the msub client as the user who submits the job and can be bypassed, and server-based submit filters run from the Moab server as the user as which the server is running and cannot be bypassed.

Client-based submit filter

Client-based filters run from the msub client as the user who is submitting the job. Because they do not have elevated privileges, the risk of client-based submit filters' being abused is low; however, it is possible for the client to specify its own configuration file and bypass the filter or substitute its own filter. Job submissions do not even reach the server if a client-based submit filter rejects it.

To configure msub to use the submit filter, give each submission host access to the submit filter script and add a SUBMITFILTER parameter to the Moab configuration file (`moab.cfg`) on each submission host. The following example demonstrates how you might modify the `moab.cfg` file:

```
SUBMITFILTER /home/submitfilter/filter.pl
```

If you experience problems with your submit filter and want to debug its interaction with msub, enter `msub --loglevel=9`. This will cause msub to print verbose log messages to the terminal.

Server-based submit filter

Server-based submit filters run from the Moab server as the user as which the server is running. Because it runs as a privileged user, you must evaluate the script closely for security implications. A client configuration cannot bypass the filter.

To configure Moab to automatically apply a filter to all job submissions, use the **SERVERSUBMITFILTER** on page 1019 parameter. **SERVERSUBMITFILTER** specifies the path to a global job submit filter script, which Moab will run on the head node and apply to every job submitted.

```
SERVERSUBMITFILTER /opt/moab/scripts/jobFilter.pl
```

Moab runs `jobFilter.pl`, located in the `/opt/moab/scripts` directory, on the head node, applying the filter to all jobs submitted.

Sample submit filter script

The following example is a trivial implementation that will not affect whether a job is submitted. Use it as reference to verify that you are writing your filter properly.

```
#!/usr/bin/perl
use strict;

## Simple filter example that re-directs the output to a file.

my $file = "xmllog.out";

open FILE,">>$file" or die "Couldn't open $file: $!";
while (<>)
{
print FILE;
print;
}
close FILE;
```

Submitting Jobs via msub in XML

The following describes the XML format used with the msub command to submit a job to a Moab server. This information can be used to implement a filter and modify the XML normally generated by the msub command. The XML format described in what follows is based on a variant of the Scalable Systems Software Job Object Specification.

Overall XML Format

The overall format of an XML request to submit a job can be shown through the following example:

```
<job>
**job attribute children**
</job>
```

An example of a simple job element with all the required children for a job submission is as follows:

```
<job>
    <Owner>user</Owner>
    <UserId>user</UserId>
    <GroupId>group</GroupId>
    <InitialWorkingDirectory>/home/user/directory</InitialWorkingDirectory>
    <UMask>18</UMask>
    <Executable>/full/path/to/script/or/first/line/of/stdin</Executable>
    <SubmitLanguage>Resource Manager Type</SubmitLanguage>
    <SubmitString>\START\23!/usr/bin/ruby\0contents\20of\20script</SubmitString>
  </job>
```

The section that follows entitled Job Element Format describes the possible attributes and their meanings in detail. In actuality, all that is needed to run a job in Moab is something similar to the following:

```
<job>
  <SubmitString>\START\23!/bin/sh\0asleep\201000</SubmitString>
</job>
```

This piece of XML requests Moab to submit a job using the contents of the SubmitString tag as a script, which is in this case a simple sh script to sleep for 1000 seconds. The msub command will create default values for all other needed attributes.

Job Element Format

The job element of the submission request contains a list of children and string values inside the children that represent the attribute/value pairs for the job. The earlier section, Overall XML Format, gives an example of this format. This section explains these attributes in detail.

Arguments — The arguments to be passed to the program are normally specified as arguments after the first argument specifying the script to be executed.

EligibleTime — The minimum time after which the job is eligible. This is the equivalent of the `-a` option in msub. Format: `[[[[CC]YY]MM]DD]hhmm[.SS]`

Environment — The semi-colon list of environment variables that are exported to the job (taken from the msub command environment). The `-V` msub flag, for example, adds all the environment variables present at the time msub is invoked. Environment variables are delimited by the `~rs;` characters. Following is an example of the results of the `msub -v arg1=1,arg2=2` command:

```
<Environment>arg1=1~rs;arg2=2~rs;</Environment>
```

ErrorFile — Defines the path to be used for the standard error stream of the batch job. This is equivalent to the `-e` flag in msub.

Executable — This is normally either the name of the script to be executed, or the first line of the script if it is passed to msub through standard input.

Extension — The resource manager extension string. This can be specified via the command line in a number of ways, including the -W x= directive. Some other requests, such as some extensions used in the `-l` flag, are also converted to an extension string. The element has the following format:

```
<Extension>x=extension</Extension>
```

See Using the Extension Element to Submit Triggers for additional information on the extension element.

GroupId — The string name of the group of the user submitting the job. This will correspond to the user's primary group on the operating system.

Hold — Specifies that a user hold be applied to the job at submission time. This is the equivalent to the msub flag `-h`. It will have the form:

```
<Hold>User</Hold>
```

InitialWorkingDirectory — Specifies in which directory the job should begin executing. This is equivalent to the `-d` flag in the msub command.

```
<InitialWorkingDirectory>/home/user/directory</InitialWorkingDirectory>
```

Interactive — Specifies that the job is to be interactive. This is the equivalent of the `-I` flag in msub.

```
<Interactive>TRUE</Interactive>
```

JobName — Specifies the user-specified job name attribute. This is equivalent to the `-N` flag in msub.

NotificationList — Specifies the job states after which an email should be sent and also specifies the users to be emailed. This is the equivalent of the `-m` and `-M` options in msub.

```
<NotificationList URI=user1:user2>JobFail,JobStart,JobEnd</NotificationList>
```

In this example, the command `msub -m abe -M user1:user2` ran indicating that emails should be sent when a job fails, starts, or ends, and that they should be sent to `user1` and `user2`.

OutputFile — Defines the path to be used for the standard output stream of the batch job. This is the equivalent of the `-o` flag in msub.

Priority — A user-requested priority value. This is the equivalent to the msub `-p` flag.

ProjectId — Defines the account associated with the job. This is equivalent to the `-A` msub flag.

QueueName — The requested class of the job. This is the equivalent of the msub `-q` flag.

Requested — Specifies resources and attributes the job specifically requests and has the following form:

```
<Requested>
     <... requested attributes>
</Requested>
```

See the section dedicated to requestable attributes in this element.

RMFlags — Flags that will get passed directly to the resource manager on job submission. This is equivalent to any arguments listed after the `-l` msub flag.

```
<RMFlags>arg1 arg2 arg3</RMFlags>
```

ShellName — Declares the shell that interprets the job script. This is equivalent to the msub flag `-S`.

SubmitLanguage — Resource manager whose language the job is using. Use TORQUE to specify a TORQUE resource manager.

SubmitString — Contains the contents of the script to be run, retrieved either from an actual script or from standard input. This also includes all resource manager specific directives that may have been in the script already or added as a result of other command line arguments.

TaskGroup — Groups a set of requested resources together. It does so by encapsulating a Requested element. For example, the command `msub -l nodes=2+nodes=3:ppn=2` generates the following XML:

```
<TaskGroup>
    <Requested>
      <Processors>2</Processors>
      <TPN>2</TPN>
    </Requested>
  </TaskGroup>
  <TaskGroup>
    <Requested>
      <Processors>2</Processors>
    </Requested>
  </TaskGroup>
```

UserId — The string value of the user ID of the job owner. This will correspond to the user's name on the operating system.

Using the Extension Element to Submit Triggers

Use the Extension element to submit triggers. With the exception of certain characters, the syntax for trigger creation is the same for non-XML trigger submission. See **About object triggers** on page 724 for detailed information on triggers. The ampersand (`&`) and less than sign (`<`) characters must be replaced for the XML to be valid. The following example shows how the Extension element is used to submit multiple triggers (separated by a semi-colon). Note that ampersand characters are replaced with `&` in the example:

```
<Job>
  <UserId>user1</UserId>
  <GroupId>user1</GroupId>
  <Arguments>60</Arguments>
  <Executable>/bin/sleep</Executable>

<Extension>x=trig:AType=exec&amp;Action="env"&amp;EType=start;trig:AType=exec&amp;Acti
on="trig2.sh"&amp;EType=end</Extension>
  <Processors>3</Processors>
  <Disk>500</Disk>
  <Memory>1024</Memory>
  <Swap>600</Swap>
  <WallclockDuration>300</WallclockDuration>
  <Environment>PERL5LIB=/perl5:</Environment>
</Job>
```

Elements Found in Requested Element

The following describes the tags that can be found in the Requested sub-element of the job element in a job submission request.

Nodes — A list of nodes that the job requests to be run on. This is the equivalent of the `-l hosts=<host-list>` msub directive.

```
<Requested>
  <Nodes>
    <Node>n1:n2</Node>
  </Nodes>
</Requested>
```

In this example, the users requested the hosts `n1` and `n2` with the command `msub -l host=n1:n2`.

Processors — The number of processors requested by the job. The following example was generated with the command `msub -l nodes=5`:

```
<Requested>
    <Processors>5</Processors>
</Requested>
```

TPN — Tasks per node. This is generated using the ppn resource manager extensions. For example, from `msub -l nodes=3:ppn=2`, the following results:

```
  <Requested>
    <Processors>6</Processors>
    <TPN>2</TPN>
  </Requested>
```

WallclockDuration — The requested wallclock duration of the job. This attribute is specified in the Requested element.

```
<Requested>
  <WallclockDuration>3600</WallclockDuration>
</Requested>
```

Related topics

- Applying the msub Submit Filter
- SUBMITFILTER parameter

mvcctl (Moab Virtual Container Control)

Synopsis

- mvcctl -a *<OType>*:*<OName>*[,*<OType>*:*<OName>*] *<name>*
- mvcctl -c [*<description>*]
- mvcctl -d *<name>*
- mvcctl -m *<ATTR>*=VAL[,*<ATTR>*=*<VAL>*] *<name>*

- mvcctl -q [<*name*>|ALL] [--xml][--blocking][--flags=fullxml]
- mvcctl -r <*OType*>:<*OName*>[,<*OType*>:<*OName*>] <*name*>
- mvcctl -x <*action*><*name*>

Overview

A virtual container (VC) is a logical grouping of objects with a shared variable space and applied policies. Containers can hold virtual machines, jobs, reservations, and nodes. Containers can also be nested inside other containers.

A VC can be owned by a user, group, or account. Users can only view VCs to which they have access. Level 1 administrators (Admin1) can view and modify all VCs. The owner can also be changed. When modifying the owner, you must also specify the owner type:

```
mvcctl -m OWNER=acct:bob myvc
```

Adding objects to VCs at submission: You associate jobs, VMs, and reservations with a specified VC upon submission. For example,

- mrsvctl -c ... -H <*VC*>
- msub ... -W x="vc=<*VC*>"
- mvmctl -c ...,vc=<*VC*>

> The user who submits objects must have access to the VC or the command is rejected.

FullXML flag

The FullXML flag will cause the mvcctl -q command to show VCs in a hierarchical manner. If doing a non-XML (plaintext) query, sub-VCs will be listed inside their parent VCs. Each VC will be indented more than its parent.

```
VC[vc2] (vc2)
     Owner: user:jason
     VCs:
     VC[vc1] (vc1)
         Owner: user:jason
         Jobs:  Moab.1
         Rsvs:  system.1
         VCs:
         VC[vc3] (vc3)
             Owner: user:jason
     VC[vc4] (vc4)
         Owner: user:jason
```

If doing an XML query, the XML for all sub-objects (VCs, but also reservations, jobs, etc.) will also be included in the VC.

```
<Data>
   <vc DESCRIPTION="vc2" NAME="vc2" OWNER="user:jason">
     <vc DESCRIPTION="vc1" NAME="vc1" OWNER="user:jason">
       <job CmdFile="sleep 7200" Flags="GLOBALQUEUE,NORMSTART"
      Group="jason" JobID="Moab.1" PAL="[base]" RM="internal"
      ReqAWDuration="2:00:00" User="jason">
         <req Index="0"></req>
       </job>
       <rsv ACL="RSV=%=system.1=;" AUser="jason"
       AllocNodeList="n0,n1,n2,n3,n4,n5,n6,n7,n8,n9" HostExp="ALL"
       HostExpIsSpecified="TRUE" Name="system.1" Partition="base"
       ReqNodeList="n0:1,n1:1,n2:1,n3:1,n4:1,n5:1,n6:1,n7:1,n8:1,n9:1"
       Resources="PROCS=[ALL]" StatCIPS="5964.00" SubType="Other"
       Type="User" ctime="1299953557" duration="3600"
       endtime="1299957157"
       flags="ISCLOSED,ISGLOBAL,ISACTIVE,REQFULL"
       starttime="1299953557">
         <ACL aff="neutral" cmp="%=" name="system.1" type="RSV">
         </ACL>
         <CL aff="neutral" cmp="%=" name="system.1" type="RSV"></CL>
         <History>
           <event state="PROCS=40" time="1299953557"></event>
         </History>
       </rsv>
       <vc DESCRIPTION="vc3" NAME="vc3" OWNER="user:jason"></vc>
     </vc>
     <vc DESCRIPTION="vc4" NAME="vc4" OWNER="user:jason"></vc>
   </vc>
 </Data>
```

Note that the XML from the blocking and non-blocking commands may differ.

Virtual Container Flags

The following table indicates available virtual container (VC) flags and associated descriptions. Note that the `Deleting`, `HasStarted`, and `Workflow` flags cannot be set by a user but are helpful indicators of status.

VC Flags	
DestroyObjects	When the VC is destroyed, any reservations, jobs, and VMs in the VC are also destroyed. This is recursive, so any objects in sub-VCs are also destroyed. Nodes are not removed.
DestroyWhenEmpty	When the VC is empty, it is destroyed.
Deleting	Set by the scheduler when the VC has been instructed to be removed. Internal flag. Administrators cannot set or clear this flag.

VC Flags	
HasStarted	This flag is set on a VC workflow where at least one job has started. Internal flag. Administrators cannot set or clear this flag.
HoldJobs	This flag will place a hold on any job that is submitted to the VC while this flag is set. It is not applied for already existing jobs that are added into the VC. If a job with a workflow is submitted to the VC, all jobs within the workflow are placed on hold.
NoReleaseWhenScheduled	Prevents Moab from lifting the UserHold on the workflow when it is scheduled. This enables an approval method in which an administrator must release the hold manually before the service is allowed to start as scheduled.
Workflow	Designates this VC as a VC that is for workflows. This flag is set when generated by a job template workflow. Workflow jobs can only be attached to one workflow VC. Internal flag. Administrators cannot set or clear this flag.

Format

-a	
Format	mvcctl -a*<OType>*:*<OName>*[,*<OType>*:*<OName>*] *<name>* Where *<OType>* is one of `JOB`, `RSV`, `NODE`, `VC`, or `VM`.
Description	Add the given object(s).
Example	`mvcctl -a JOB:Moab.45 vc13` `>>job 'Moab.45' added to VC 'vc13'`

-c	
Format	mvcctl -c [*<description>*]
Description	Create a virtual container (VC). The VC name is auto-generated. It is recommended that you supply a description; otherwise the description is the same as the auto-generated name.
Example	`mvcctl -c "Linux testing machine"` `>>VC 'vc13' created`

-d	
Format	mvcctl -d*<lab01>*
Description	Destroy the VC.
Example	`mvcctl -d vc13` `>>VC 'vc13' destroyed`

-m	
Format	mvcctl -m*<ATTR>*=VAL[,*<ATTR>*=*<VAL>*] *<name>*
Description	Modify the VC. Attributes are flags, owner, reqstarttime, reqnodeset, variables, and owner; note that only the owner can modify owner. Use reqstarttime when implementing guaranteed start time to specify when jobs should start. The reqnodeset attribute indicates the node set that jobs should run in that are submitted to a virtual container.
Example	`mvcctl -m variables+=HV=node8 vc13` `>>VC 'vc13' successfully modified` `mvcctl -m flags+=DESTROYWHENEMPTY vc1` `>>VC 'vc1' successfully modified`

-q	
Format	mvcctl -q [*<name>*|ALL] [--xml][--blocking][--flags=fullxml]
Description	Query VCs
Example	`mvcctl -q ALL` `VC[vc13] (Linux testing machine)` `Create Time: 1311027343    Creator: jdoe` `Owner: user:jdoe` `ACL:   USER=%=jdoe+;` `Jobs:  Moab.45` `Vars:  HV=node88` `Flags: DESTROYWHENEMPTY`

-r	
Format	mvcctl -r*<OType>*:*<OName>*[,*<OType>*:*<OName>*] *<name>* Where *<OType>* is one of `JOB`, `RSV`, `NODE`, `VC`, or `VM`.
Description	Remove the given object(s) from the VC.
Example	`mvcctl -r JOB:Moab.45 vc13` `>>job 'Moab.45' removed from VC 'vc13'`

-x	
Format	mvcctl -x*<action><name>*
Description	Executes the given action on the virtual container (VC).
Example	`mvcctl -x schedulevc vc1`

mvmctl

Synopsis

mvmctl -d [--flags=force] *<vmid>*

mvmctl -f *<migrationPolicy>* [--flags=eval [--xml]]

mvmctl -m [*<options>*] *<vmid>*

mvmctl -M dsthost=*<newhost><vmid>*

mvmctl -q *<vmid>* [--blocking] [--xml]

mvmctl -w state=drained

Overview

mvmctl controls the modification, querying, migration, and destruction of virtual machines (VMs).

Format

-d	
Name	Destroy
Format	mvmctl -d [--flags=force] *<vmid>*
Description	Destroys the specified VM. When you add the force flag, Moab forces the deletion of the VM if and only if it does not have a VM-tracking job.
Example	`> mvmctl -d oldVM` `> mvmctl -d --flags=force oldVM` *Because `oldVM` does not have a VM-tracking job associated with it and you set the force flag, Moab forces the deletion of `oldVM`*

-f	
Name	Force Migrate
Format	mvmctl -f consolidation|overcommit [--flags=eval [--xml]]
Description	Forces the migration policy on the system. The `eval` flag causes Moab to run through migration routines and report the results without actually migrating the VMs.
Example	`> mvmctl -f consolidation --flags=eval` Moab returns a report like the following: `1: VM 'vm1' from 'h0' to 'h3'` `2: VM 'vm2' from 'h0' to 'h5'`

-m	
Name	Modify
Format	[*<options>*] *<vmid>* The *<options>* variable is a comma-separated list of *<attr>*=*<value>* pairs.

-m	
Description	Modifies the VM.
Example	`> mvmctl -m gevent=hitemp:'mymessage' myNewVM` *Gevents can be set using `gevent`.* `> mvmctl -m gmetric=bob:5.6 myNewVM` *Gmetrics can be set using `gmetric`.* `> mvmctl -m os=compute myNewVM` *Reprovisioning is done by changing `os`.* `> mvmctl -m powerstate=off myNewVM` *Power management is done by modifying `powerstate`.* `> mvmctl -m variable=user:bob+purpose:myVM myNewVM` *The modify variable uses the same syntax as Create.* `> mvmctl -m flags=cannotmigrate myNewVM` *Allow a VM to migrate by setting the `canmigrate` flag.* `> mvmctl -m flags=canmigrate myNewVM` *Allows a VM to migrate by setting the canmigrate flag.*
Notes	• The variable option is a set-only operation. Previous variables will be overwritten.

-M	
Name	Migrate
Format	dsthost=*<newhost><vmid>*

-M	
Description	Migrate the given VM to the destination host. When you set the *vmid* to ANY, Moab migrates the VM to any available eligible hypervisor. For this to work, the following conditions must be met: • The VM reports a **CPULOAD**, and it is greater than 0. • The VM's **AMEMORY** is less than its **CMEMORY**. This indicates that some memory is currently in use and tells Moab that the RM is reporting memory correctly. • The VM's state is not "Unknown." • All hypervisors report a **CPULOAD**, and it is greater than 0. • All hypervisors report an **AMEMORY**, and it is less than its **CMEMORY**. • All hypervisors report a hypervisor type.
Example	`> mvmctl -M dsthost=node05 myNewVM` *`myNewVM` migrates to `node05`.* `> mvmctl -M dsthost=ANY vm42` *Moab migrates `vm42` to a node based on policy destination limitations (such as the NoVMMigrations flag).*

-q	
Name	Query
Format	`<vmid>` [--blocking] [--xml]
Description	Queries the specified VM; that is, it returns detailed information about the given VM. May be used with or without the `--xml` flag. ALL may also be used to display information about all VMs. This option gathers information from the Moab cache which prevents it from waiting for the scheduler, but the `--blocking` option can be used to bypass the cache and allow waiting for the scheduler.
Example	`> mvmctl -q myNewVM` `> mvmctl -q ALL --blocking` `> mvmctl -q ALL --xml`

-w	
Name	Constraint
Format	state=drained
Description	Overrides the HIDEDRAINED DISPLAYFLAGS attribute allowing display of VMs in a DRAINED state.
Example	`> mvmctl -q -w state=drained`

showbf

Synopsis

showbf [-A] [-a *account*] [-c *class*] [-d *duration*] [-D] [-f *features*] [-g *group*] [-L] [-m [==|>|>=|<|<=] *memory*] [-n *nodecount*] [-p *partition*] [-q *qos*] [-u *user*] [-v] [--blocking]

Overview

Shows what resources are available for immediate use.

> The results Moab returns do not include resources that may be freed due to preemption.

This command can be used by any user to find out how many processors are available for immediate use on the system. It is anticipated that users will use this information to submit jobs that meet these criteria and thus obtain quick job turnaround times. This command incorporates down time, reservations, and node state information in determining the available backfill window.

> If specific information is not specified, showbf will return information for the user and group running but with global access for other credentials. For example, if `-q qos` is not specified, Moab will return resource availability information for a job as if it were entitled to access all QOS based resources (i.e., resources covered by reservations with a QOS based ACL), if `-c class` is not specified, the command will return information for resources accessible by any class.

> The showbf command incorporates node configuration, node utilization, node state, and node reservation information into the results it reports. This command does not incorporate constraints imposed by credential based fairness policies on the results it reports.

Access

By default, this command can be used by any user or administrator.

Parameters

Parameter	Description
ACCOUNT	Account name.
CLASS	Class/queue required.
DURATION	Time duration specified as the number of seconds or in [DD:]HH:MM:SS notation.
FEATURELIST	Colon separated list of node features required.
GROUP	Specify particular group.
MEMCMP	Memory comparison used with the `-m` flag. Valid signs are >, >=, ==, <=, and <.
MEMORY	Specifies the amount of required real memory configured on the node, (in MB), used with the `-m` flag.
NODECOUNT	Specify number of nodes for inquiry with `-n` flag.
PARTITION	Specify partition to check with `-p` flag.
QOS	Specify QOS to check with `-q` flag.
USER	Specify particular user to check with `-u` flag.

Flags

Flag	Description
-A	Show resource availability information for all users, groups, and accounts. By default, showbf uses the default user, group, and account ID of the user issuing the command.
-a	Show resource availability information only for specified account.
--blocking	Do not use cache information in the output. The `--blocking` flag retrieves results exclusively from the scheduler.
-d	Show resource availability information for specified duration.

Flag	Description
-D	Display current and future resource availability notation.
-g	Show resource availability information only for specified group.
-h	Help for this command.
-L	Enforce Hard limits when showing available resources.
-m	Allows user to specify the memory requirements for the backfill nodes of interest. It is important to note that if the optional MEMCMP and MEMORY parameters are used, they must be enclosed in single ticks (') to avoid interpretation by the shell. For example, enter `showbf -m '==256'` to request nodes with 256 MB memory.
-n	Show resource availability information for a specified number of nodes. That is, this flag can be used to force showbf to display only blocks of resources with at least this many nodes available.
-p	Show resource availability information for the specified partition.
-q	Show information for the specified QOS.
-r	Show resource availability for the specified processor count.
-u	Show resource availability information only for specified user.

Examples

Example 3-50:

In this example, a job requiring up to 2 processors could be submitted for immediate execution in partition `ClusterA` for any duration. Additionally, a job requiring 1 processor could be submitted for immediate execution in partition `ClusterB`. Note that by default, each task is tracked and reported as a request for a single processor.

```
> showbf
Partition     Tasks  Nodes   StartOffset      Duration       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               3      3      00:00:00      INFINITY  11:32:38_08/19
ReqID=0
ClusterA          1      1      00:00:00      INFINITY  11:32:38_08/19
ReqID=0
ClusterB          2      2      00:00:00      INFINITY  11:32:38_08/19
ReqID=0
```

StartOffset is the amount of time remaining before resources will be available.

Example 3-51:

In this example, the output verifies that a backfill window exists for jobs requiring a 3 hour runtime and at least 16 processors. Specifying job duration is of value when time based access is assigned to reservations (i.e., using the **SRCFG** **TIMELIMIT** ACL)

```
> showbf -r 16 -d 3:00:00
Partition     Tasks  Nodes   Duration    StartOffset    StartDate
---------     -----  -----   --------    -----------    ---------
ALL           20     20      INFINITY    00:00:00       09:22:25_07/19
```

Example 3-52:

In this example, a resource availability window is requested for processors located only on nodes with at least 512 MB of memory.

```
> showbf -m ' =512'
Partition     Tasks  Nodes   Duration    StartOffset    StartDate
---------     -----  -----   --------    -----------    ---------
ALL           20     20      INFINITY    00:00:00       09:23:23_07/19
ClusterA      10     10      INFINITY    00:00:00       09:23:23_07/19
ClusterB      10     10      INFINITY    00:00:00       09:23:23_07/19
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- showq
- mdiag -t

showq

Synopsis

showq [-b] [-g] [-l] [-c|-i|-r] [-n] [-o] [-p *partition*] [-R *rsvid*] [-u] [-v] [-w <CONSTRAINT>] [--blocking] [--noblock]

Overview

Displays information about active, eligible, blocked, and/or recently completed jobs. Since the resource manager is not actually scheduling jobs, the job ordering it displays is not valid. The showq command displays the actual job ordering under the Moab Workload Manager. When used without flags, this command displays all jobs in active, idle, and non-queued states.

Access

By default, this command can be run by any user. However, the `-c`, `-i`, and `-r` flags can only be used by level 1, 2, or 3 Moab administrators.

Flags

Flag	Description
-b	Display blocked jobs only
-c	Display details about recently completed jobs (see example, JOBCPURGETIME).
-g	Display grid job and system IDs for all jobs.
-i	Display extended details about idle jobs. (see example)
-l	Display local/remote view. For use in a Grid environment, displays job usage of both local and remote compute resources.
-n	Displays normal showq output, but lists job names under JOBID
-o	Displays jobs in the active queue in the order specified (uses format `showq -o <specifiedOrder>`). Valid options include REMAINING, REVERSEREMAINING, JOB, USER, STATE, and STARTTIME. The default is REMAINING.
-p	Display only jobs assigned to the specified partition.
-r	Display extended details about active (running) jobs. (see example)
-R	Display only jobs which overlap the specified reservation.
-u	Display all running jobs for a particular user.
-v	Display local and full resource manager job IDs as well as partitions. If specified with the `-i` option, will display job reservation time. The `-v` option displays all array subjobs. All showq commands without the `-v` option show just the master jobs in an array.
-w	Display only jobs associated with the specified constraint. Valid constraints include user, group, acct, class, and qos (see showq -w example.).
--blocking	Do not use cache information in the output. The `--blocking` flag retrieves results exclusively from the scheduler.
--noblock	Use cache information for a faster response.

Details

Beyond job information, the showqcommand will also report if the scheduler is stopped or paused or if a system reservation is in place. Further, the showq command will also report public system messages.

Examples

- **Default Report** on page 327
    - **Detailed Active/Running Job Report** on page 330
    - **Eligible Jobs** on page 329
    - **Detailed Completed Job Report** on page 333
- **Filtered Job Report** on page 334

Example 3-53: Default Report

The output of this command is divided into three parts, Active Jobs, Eligible Jobs, and Blocked Jobs.

```
> showq

active jobs------------------------
JOBIDUSERNAMESTATEPROCSREMAINING            STARTTIME

12941             sartois    Running    25     2:44:11  Thu Sep  1 15:02:50
12954              tgates    Running     4     2:57:33  Thu Sep  1 15:02:52
12944               eval1    Running    16     6:37:31  Thu Sep  1 15:02:50
12946              tgates    Running     2   1:05:57:31  Thu Sep  1 15:02:50

4 active jobs             47 of 48 processors active (97.92%)
                          32 of 32 nodes active      (100.00%)

eligible jobs----------------------
JOBID              USERNAME      STATE  PROCS     WCLIMIT            QUEUETIME

12956              cfosdyke       Idle     32     6:40:00  Thu Sep  1 15:02:50
12969              cfosdyke       Idle      4     6:40:00  Thu Sep  1 15:03:23
12939                 eval1       Idle     16     3:00:00  Thu Sep  1 15:02:50
12940               mwillis       Idle      2     3:00:00  Thu Sep  1 15:02:50
12947               mwillis       Idle      2     3:00:00  Thu Sep  1 15:02:50
12949                 eval1       Idle      2     3:00:00  Thu Sep  1 15:02:50
12953                tgates       Idle     10     4:26:40  Thu Sep  1 15:02:50
12955                 eval1       Idle      2     4:26:40  Thu Sep  1 15:02:50
12957                tgates       Idle     16     3:00:00  Thu Sep  1 15:02:50
12963                 eval1       Idle     16  1:06:00:00  Thu Sep  1 15:02:52
12964                tgates       Idle     16  1:00:00:00  Thu Sep  1 15:02:52
12937               allendr       Idle      9  1:00:00:00  Thu Sep  1 15:02:50
12962                aacker       Idle      6    00:26:40  Thu Sep  1 15:02:50
12968               tamaker       Idle      1     4:26:40  Thu Sep  1 15:02:52

14 eligible jobs

blocked jobs-----------------------
JOBID              USERNAME      STATE  PROCS     WCLIMIT            QUEUETIME


0 blocked jobs

Total jobs:  18
```

The fields are as follows:

Column	Description
JOBID	Job identifier.
USERNAME	User owning job.
STATE	Job State. Current batch state of the job.
PROCS	Number of processors being used by the job.

Column	Description
REMAINING/WCLIMIT	For active jobs, the time the job has until it has reached its wallclock limit or for idle/b-locked jobs, the amount of time requested by the job. Time specified in [DD:]HH:MM:SS notation.
STARTTIME	Time job started running.

Active Jobs

Active jobs are those that are Running or Starting and consuming resources. Displayed are the job id*, the job's owner, and the job state. Also displayed are the number of processors allocated to the job, the amount of time remaining until the job completes (given in HH:MM:SS notation), and the time the job started. All active jobs are sorted in "Earliest Completion Time First" order.

> *Job IDs may be marked with a single character to specify the following conditions:
>
> | Character | Description |
> |---|---|
> | **_ (underbar)** | job violates usage limit |
> | *** (asterisk)** | job is backfilled AND is preemptible |
> | **+ (plus)** | job is backfilled AND is NOT preemptible |
> | **- (hyphen)** | job is NOT backfilled AND is preemptible |

> Detailed active job information can be obtained using the `-r` flag.

Eligible Jobs

Eligible Jobs are those that are queued and eligible to be scheduled. They are all in the Idle job state and do not violate any fairness policies or have any job holds in place. The jobs in the Idle section display the same information as the Active Jobs section except that the wallclock CPULIMIT is specified rather than job time REMAINING, and job QUEUETIME is displayed rather than job STARTTIME. The jobs in this section are ordered by job priority. Jobs in this queue are considered eligible for both scheduling and backfilling.

> Detailed eligible job information can be obtained using the `-i` flag.

Blocked Jobs

Blocked jobs are those that are ineligible to be run or queued. Jobs listed here could be in a number of states for the following reasons:

State	Description
Idle	Job violates a fairness policy. Use `diagnose -q` for more information.
UserHold	A user hold is in place.
SystemHold	An administrative or system hold is in place.
BatchHold	A scheduler batch hold is in place (used when the job cannot be run because the requested resources are not available in the system or because the resource manager has repeatedly failed in attempts to start the job).
Deferred	A scheduler defer hold is in place (a temporary hold used when a job has been unable to start after a specified number of attempts. This hold is automatically removed after a short period of time).
NotQueued	Job is in the resource manager state NQ (indicating the job's controlling scheduling daemon in unavailable).

A summary of the job queue's status is provided at the end of the output.

Example 3-54: Detailed Active/Running Job Report

```
> showq -r

active jobs------------------------
JOBID               S  PAR  EFFIC  XFACTOR  Q      USER    GROUP        MHOST PROCS
REMAINING            STARTTIME

12941               R    3 100.00      1.0  -   sartois   Arches       G5-014 25
2:43:31  Thu Sep  1 15:02:50
12954               R    3 100.00      1.0 Hi    tgates   Arches       G5-016  4
2:56:54  Thu Sep  1 15:02:52
12944               R    2 100.00      1.0 De     eval1  RedRock     P690-016 16
6:36:51  Thu Sep  1 15:02:50
12946               R    3 100.00      1.0  -    tgates   Arches       G5-001  2
1:05:56:51  Thu Sep  1 15:02:50

4 active jobs             47 of 48 processors active (97.92%)
                          32 of 32 nodes active      (100.00%)

Total jobs:  4
```

The fields are as follows:

Column	Description
JOBID	Name of active job.

Column	Description
S	Job State. Either R for Running or S for Starting.
PAR	Partition in which job is running.
EFFIC	CPU efficiency of job.
XFACTOR	Current expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit
Q	Quality Of Service specified for job.
USERNAME	User owning job.
GROUP	Primary group of job owner.
MHOST	Master Host running primary task of job.
PROCS	Number of processors being used by the job.
REMAINING	Time the job has until it has reached its wallclock limit. Time specified in HH:MM:SS notation.
STARTTIME	Time job started running.

After displaying the running jobs, a summary is provided indicating the number of jobs, the number of allocated processors, and the system utilization.

Column	Description
JobName	Name of active job.
S	Job State. Either R for Running or S for Starting.
CCode	Completion Code. The return/completion code given when a job completes. (Only applicable to completed jobs.)
Par	Partition in which job is running.
Effic	CPU efficiency of job.
XFactor	Current expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit

Column	Description
Q	Quality Of Service specified for job.
User	User owning job.
Group	Primary group of job owner.
Nodes	Number of processors being used by the job.
Remaining	Time the job has until it has reached its wallclock limit. Time specified in HH:MM:SS notation.
StartTime	Time job started running.

```
> showq -i

eligible jobs----------------------
JOBID                 PRIORITY  XFACTOR  Q      USER    GROUP  PROCS     WCLIMIT
CLASS      SYSTEMQUEUETIME

12956*                      20      1.0  -  cfosdyke  RedRock     32     6:40:00
batch  Thu Sep  1 15:02:50
12969*                      19      1.0  -  cfosdyke  RedRock      4     6:40:00
batch  Thu Sep  1 15:03:23
12939                       16      1.0  -     eval1  RedRock     16     3:00:00
batch  Thu Sep  1 15:02:50
12940                       16      1.0  -   mwillis   Arches      2     3:00:00
batch  Thu Sep  1 15:02:50
12947                       16      1.0  -   mwillis   Arches      2     3:00:00
batch  Thu Sep  1 15:02:50
12949                       16      1.0  -     eval1  RedRock      2     3:00:00
batch  Thu Sep  1 15:02:50
12953                       16      1.0  -    tgates   Arches     10     4:26:40
batch  Thu Sep  1 15:02:50
12955                       16      1.0  -     eval1  RedRock      2     4:26:40
batch  Thu Sep  1 15:02:50
12957                       16      1.0  -    tgates   Arches     16     3:00:00
batch  Thu Sep  1 15:02:50
12963                       16      1.0  -     eval1  RedRock     16  1:06:00:00
batch  Thu Sep  1 15:02:52
12964                       16      1.0  -    tgates   Arches     16  1:00:00:00
batch  Thu Sep  1 15:02:52
12937                        1      1.0  -   allendr  RedRock      9  1:00:00:00
batch  Thu Sep  1 15:02:50
12962                        1      1.2  -    aacker  RedRock      6    00:26:40
batch  Thu Sep  1 15:02:50
12968                        1      1.0  -   tamaker  RedRock      1     4:26:40
batch  Thu Sep  1 15:02:52

14 eligible jobs

Total jobs:  14
```

The fields are as follows:

Column	Description
JOBID	Name of job.
PRIORITY	Calculated job priority.
XFACTOR	Current expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit
Q	Quality Of Service specified for job.
USER	User owning job.
GROUP	Primary group of job owner.
PROCS	Minimum number of processors required to run job.
WCLIMIT	Wallclock limit specified for job. Time specified in HH:MM:SS notation.
CLASS	Class requested by job.
SYSTEMQUEUETIME	Time job was admitted into the system queue.

An asterisk at the end of a job (job `12956*` in this example) indicates that the job has a job reservation created for it. The details of this reservation can be displayed using the checkjob command.

Example 3-55: Detailed Completed Job Report

```
> showq -c
completed jobs------------------------
JOBID               SCCODE  PAR  EFFIC  XFACTOR  Q  USERNAME    GROUP        MHOST
PROC    WALLTIME              STARTTIME
13098                C     0  bas  93.17      1.0  -   sartois    Arches       G5-014
25    2:43:31  Thu Sep  1 15:02:50
13102                C     0  bas  99.55      2.2 Hi    tgates    Arches       G5-016
 4    2:56:54  Thu Sep  1 15:02:52
13103                C     2  tes  99.30      2.9 De     eval1  RedRock     P690-016
16    6:36:51  Thu Sep  1 15:02:50
13115                C     0  tes  97.04      1.0  -    tgates    Arches       G5-001
 2 1:05:56:51  Thu Sep  1 15:02:50
3 completed jobs
```

The fields are as follows:

Column	Description
JOBID	job id for completed job.
S	Job State. Either C for Completed or V for Vacated.
CCODE	Completion code reported by the job.
PAR	Partition in which job ran.
EFFIC	CPU efficiency of job.
XFACTOR	Expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit
Q	Quality of Service specified for job.
USERNAME	User owning job.
GROUP	Primary group of job owner.
MHOST	Master Host which ran the primary task of job.
PROCS	Number of processors being used by the job.
WALLTIME	Wallclock time used by the job. Time specified in [DD:]HH:MM:SS notation.
STARTTIME	Time job started running.

After displaying the active jobs, a summary is provided indicating the number of jobs, the number of allocated processors, and the system utilization.

> If the DISPLAYFLAGS parameter is set to *ACCOUNTCENTRIC*, job group information will be replaced with job account information.

Example 3-56: Filtered Job Report

Show only jobs associated with user `john` and class `benchmark`.

```
> showq -w class=benchmark -w user=john
...
```

Job Array

Job arrays show the name of the job array and then in parenthesis, the number of sub-jobs in the job array that are in the specified state.

```
> showq

active jobs------------------------
JOBID              USERNAME      STATE PROCS   REMAINING            STARTTIME

Moab.1(14)          aesplin    Running    14    00:59:41  Fri May 27 14:58:57

14 active jobs            14 of 14 processors in use by local jobs (100.00%)
2 of 2 nodes active      (100.00%)

eligible jobs----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME

Moab.1(4)           aesplin       Idle     4     1:00:00  Fri May 27 14:58:52

4 eligible jobs

blocked jobs-----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME

Moab.1(2)           aesplin    Blocked     2     1:00:00  Fri May 27 14:58:52

2 blocked jobs

Total jobs:  20
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- showbf - command to display resource availability.
- mdiag -j - command to display detailed job diagnostics.
- checkjob - command to check the status of a particular job.
- JOBCPURGETIME - parameter to adjust the duration of time Moab preserves information about completed jobs
- DISPLAYFLAGS - parameter to control what job information is displayed

showhist.moab.pl

Synopsis

showhist.moab.pl [-a *accountname*]
[-c *classname*] [-e *enddate*]
[-g *groupname*] [-j *jobid*] [-n *days*]
[-q *qosname*] [-s *startdate*]
[-u *username*]

Overview

The `showhist.moab.pl` script displays historical job information. Its purpose is similar to the checkjob command's, but `showhist.moab.pl` displays information about jobs that have already completed.

Access

By default, this script's use is limited to administrators on the head node; however, end users can also be given power to run the script. To grant access to the script to end users, move `showhist.moab.pl` from the `tools` directory to the `bin` directory.

Arguments

-a (Account)	
Format	*<ACCOUNTNAME>*
Description	Displays job records matching the specified account.
Example	`> showhist.moab.pl -a myAccount` *Information about jobs related to the account `myAccount` is displayed.*

-c (Class)	
Format	*<CLASSNAME>*
Description	Displays job records matching the specified class (queue).
Example	`> showhist.moab.pl -c newClass` *Information about jobs related to the class `newClass` is displayed.*

-e (End Date)	
Format	YYYY-MM-DD
Description	Displays the records of jobs recorded before or on the specified date.

-e (End Date)	
Example	`> showhist.moab.pl -e 2001-01-03` *Information about all jobs recorded on or before January 3, 2001 is displayed.* `> showhist.moab.pl -s 2011-01-01 -e 2011-01-31` *Information is displayed about all jobs recorded in January 2011.*

-g (Group)	
Format	*<GROUPNAME>*
Description	Displays job records matching the specified group.
Example	`> showhist.moab.pl -g admins` *Information about jobs related to the group `admins` is displayed.*

-j (Job ID)	
Format	*<JOBID>*
Description	Displays job records matching the specified job id.
Example	`> showhist.moab.pl -j moab01` *Information about job `moab01` is displayed.*

-n (Number of Days)	
Format	*<INTEGER>*

-n (Number of Days)	
Description	Restricts the number of past jobs to search by a specified number of days relative to today.
Example	`> showhist.moab.pl -n 90 -j moab924` *Displays job information for job `moab924`. The search is restricted to the last 90 days.*

-q (QoS)	
Format	*<QOSNAME>*
Description	Displays job records matching the specified quality of service.
Example	`> showhist.moab.pl -q myQos` *Information about jobs related to the QoS `myQos` is displayed.*

-s (Start Date)	
Format	YYYY-MM-DD
Description	Displays the records of jobs that recorded on the specified date and later.
Example	`> showhist.moab.pl -s 1776-07-04` *Information about all jobs recorded on July 4, 1776 and later is displayed.* `> showhist.moab.pl -s 2001-07-05 -e 2002-07-05` *Information is displayed about all jobs recorded between July 5, 2001 and July 5, 2002.*

-u (User)	
Format	*<USERNAME>*
Description	Displays job records matching the specified user.
Example	`> showhist.moab.pl -u bob` *Information about user* `bob`*'s jobs is displayed.*

Sample Output

```
> showhist.moab.pl
Job Id            : Moab.4
User Name         : user1
Group Name        : company
Queue Name        : NONE
Processor Count   : 4
Wallclock Duration: 00:00:00
Submit Time       : Mon Nov 21 10:48:32 2011
Start Time        : Mon Nov 21 10:49:37 2011
End Time          : Mon Nov 21 10:49:37 2011
Exit Code         : 0
Allocated Nodelist: 10.10.10.3

Job Id            : Moab.1
Executable        : 4
User Name         : user1
Group Name        : company
Account Name      : 1321897709
Queue Name        : NONE
Quality Of Service: 0M
Processor Count   : -0
Wallclock Duration: 00:01:05
Submit Time       : Mon Nov 21 10:48:29 2011
Start Time        : Mon Nov 21 10:48:32 2011
End Time          : Mon Nov 21 10:49:37 2011
Exit Code         : 0
Allocated Nodelist: 512M
```

Information is displayed for all completed jobs.

When a job's Start Time and End Time are the same, the job is infinite and still running.

Related topics

- checkjob - explains how to query for a status report for a specified job.
- mdiag -j command - display additional detailed information regarding jobs
- showq command - showq high-level job summaries

showres

Synopsis

showres [-f] [-n [-g]] [-o] [-r] [*reservationid*]

Overview

This command displays all reservations currently in place within Moab. The default behavior is to display reservations on a reservation-by-reservation basis.

Access

By default, this command can be run by any Moab administrator.

Flag	Description
-f	Show free (unreserved) resources rather than reserved resources. The `-f` flag cannot be used in conjunction with the any other flag
-g	When used with the `-n` flag, shows grep-able output with nodename on every line
-n	Display information regarding all nodes reserved by *<RSVID>*
-o	Display all reservations which overlap *<RSVID>* (in time and space) Not supported with `-n` flag
-r	Display reservation timeframes in relative time mode
-v	Show verbose output. If used with the `-n` flag, the command will display all reservations found on nodes contained in *<RSVID>*. Otherwise, it will show long reservation start dates including the reservation year.

Parameter	Description
RSVID	ID of reservation of interest — optional

Examples

Example 3-57:

```
> showres

ReservationID       Type S       Start         End     Duration    N/P    StartTime

12941                Job R   -00:05:01     2:41:39     2:46:40   13/25    Thu Sep  1
15:02:50
12944                Job R   -00:05:01     6:34:59     6:40:00   16/16    Thu Sep  1
15:02:50
12946                Job R   -00:05:01  1:05:54:59  1:06:00:00    1/2     Thu Sep  1
15:02:50
12954                Job R   -00:04:59     2:55:01     3:00:00    2/4     Thu Sep  1
15:02:52
12956                Job I  1:05:54:59  1:12:34:59     6:40:00   16/32    Fri Sep  2
21:02:50
12969                Job I     6:34:59    13:14:59     6:40:00    4/4     Thu Sep  1
21:42:50

6 reservations located
```

The above example shows all reservations on the system.

The fields are as follows:

Column	Description
Type	Reservation Type. This will be one of the following: Job or User.
ReservationID	This is the name of the reservation. Job reservation names are identical to the job name. User, Group, or Account reservations are the user, group, or account name followed by a number. System reservations are given the name SYSTEM followed by a number.
S	State. This field is valid only for job reservations. It indicates whether the job is (S)tarting, (R)unning, or (I)dle.
Start	Relative start time of the reservation. Time is displayed in HH:MM:SS notation and is relative to the present time.
End	Relative end time of the reservation. Time is displayed in HH:MM:SS notation and is relative to the present time. Reservations that will not complete in 1,000 hours are marked with the keyword INFINITY.
Duration	Duration of the reservation in HH:MM:SS notation. Reservations lasting more than 1,000 hours are marked with the keyword INFINITY.
Nodes	Number of nodes involved in reservation.

Column	Description
StartTime	Time Reservation became active.

Example 3-58:

```
> showres -n
reservations on Thu Sep  1 16:49:59

NodeName         Type      ReservationID   JobState Task       Start    Duration
StartTime

G5-001            Job             12946    Running    2    -1:47:09  1:06:00:00  Thu
Sep  1 15:02:50
G5-001            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-002            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-002            Job             12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-003            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-003            Job             12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-004            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-004            Job             12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-005            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-005            Job             12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-006            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-006            Job             12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-007            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-007            Job             12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-008            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-008            Job             12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-009            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-009            Job             12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-010            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-010            Job             12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-011            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-011            Job             12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-012            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-012            Job             12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-013            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-013            Job             12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-014            Job             12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-014            Job             12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
```

```
G5-015              Job                 12956        Idle     2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-015              Job                 12949     Running     2   -00:08:57     3:00:00  Thu
Sep  1 16:41:02
G5-016              Job                 12956        Idle     2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-016              Job                 12947     Running     2   -00:08:57     3:00:00  Thu
Sep  1 16:41:02
P690-001            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-002            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-003            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-004            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-005            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-006            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-007            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-008            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-009            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-010            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-011            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-012            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-013            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-013            Job                 12969        Idle     1     4:52:51     6:40:00  Thu
Sep  1 21:42:50
P690-014            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-014            Job                 12969        Idle     1     4:52:51     6:40:00  Thu
Sep  1 21:42:50
P690-015            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-015            Job                 12969        Idle     1     4:52:51     6:40:00  Thu
Sep  1 21:42:50
P690-016            Job                 12944     Running     1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-016            Job                 12969        Idle     1     4:52:51     6:40:00  Thu
Sep  1 21:42:50

52 nodes reserved
```

This example shows reservations for nodes.

The fields are as follows:

Column	Description
NodeName	Node on which reservation is placed.

Column	Description
Type	Reservation Type. This will be one of the following: Job or User.
ReservationID	This is the name of the reservation. Job reservation names are identical to the job name. User, Group, or Account reservations are the user, group, or account name followed by a number. System reservations are given the name SYSTEM followed by a number.
JobState	This field is valid only for job reservations. It indicates the state of the job associated with the reservation.
Start	Relative start time of the reservation. Time is displayed in HH:MM:SS notation and is relative to the present time.
Duration	Duration of the reservation in HH:MM:SS notation. Reservations lasting more than 1000 hours are marked with the keyword INFINITY.
StartTime	Time Reservation became active.

Example 3-59:

```
> showres 12956

ReservationID       Type S       Start          End     Duration    N/P    StartTime

12956                Job I  1:04:09:32  1:10:49:32     6:40:00   16/32   Fri Sep  2
21:02:50

1 reservation located
```

In this example, information for a specific reservation (job) is displayed.

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mrsvctl -c - create new reservations.
- mrsvctl -r - release existing reservations.
- mdiag -r - diagnose/view the state of existing reservations.
- Reservation Overview - description of reservations and their use.

showstart

Synopsis

showstart {*jobid*|*proccount*[@*duration*]|*s3jobspec*} [-e {all|hist|prio|rsv}] [-f] [-g [*peer*]] [-l qos=<QOS>] [--blocking] [--format=xml]

Overview

This command displays the estimated start time of a job based a number of analysis types. This analysis may include information based on historical usage, earliest available reservable resources, and priority based backlog analysis. Each type of analysis will provide somewhat different estimates based on current cluster environmental conditions. By default, only reservation based analysis is performed.

> The start time estimate Moab returns does not account for resources that will become available due to preemption.

Historical analysis utilizes historical queue times for jobs which match a similar processor count and job duration profile. This information is updated on a sliding window which is configurable within `moab.cfg`

Reservation based start time estimation incorporates information regarding current administrative, user, and job reservations to determine the earliest time the specified job could allocate the needed resources and start running. In essence, this estimate will indicate the earliest time the job would start assuming this job was the highest priority job in the queue.

Priority based job start analysis determines when the queried job would fit in the queue and determines the estimated amount of time required to complete the jobs which are currently running or scheduled to run before this job can start.

In all cases, if the job is running, this command will return the time the job started. If the job already has a reservation, this command will return the start time of the reservation.

Access

By default, this command can be run by any user.

Parameters

Parameter	Description
--blocking	Do not use cache information in the output. The `--blocking` flag retrieves results exclusively from the scheduler.
DURATION	Duration of pseudo-job to be checked in format [[[DD:]HH:]MM:]SS (default duration is 1 second)
-e	Estimate method. By default, Moab will use the reservation based estimation method.
-f	Use feedback. If specified, Moab will apply historical accuracy information to Improve the quality of the estimate. See ENABLESTARTESTIMATESTATS for more information.

Parameter	Description
-g	Grid mode. Obtain showstart information from remote resource managers. If -g is not used and Moab determines that job is already migrated, Moab obtains showstart information from the remote Moab where the job was migrated to. All resource managers can be queried by using the keyword "all" which returns all information in a table. `$ showstart -g all head.1` `Estimated Start Times` `[ Remote RM ] [ Reservation ] [ Priority ] [ Historical ]` `[ c1 ] [ 00:15:35 ] [ ] [ ]` `[ c2 ] [ 3:15:38 ] [ ] [ ]`
-l qos-s=<QOS>	Specifies what QOS the job must start under, using the same syntax as the **msub** command. Currently, no other resource manager extensions are supported. This flag only applies to hypothetical jobs by using the `proccount[@duration]` syntax.
JOBID	Job to be checked
PROCCOUNT	Number of processors in pseudo-job to be checked
S3JOBSPEC	XML describing the job according to the Dept. of Energy Scalable Systems Software/S3 job specification.

Examples

Example 3-60:

```
> showstart orion.13762
job orion.13762 requires 2 procs for 0:33:20
Estimated Rsv based start in                  1:04:55 on Fri Jul 15 12:53:40
Estimated Rsv based completion in             2:44:55 on Fri Jul 15 14:33:40
Estimated Priority based start in             5:14:55 on Fri Jul 15 17:03:40
Estimated Priority based completion in        6:54:55 on Fri Jul 15 18:43:40
Estimated Historical based start in          00:00:00 on Fri Jul 15 11:48:45
Estimated Historical based completion in      1:40:00 on Fri Jul 15 13:28:45
Best Partition: fast
```

Example 3-61:

```
> showstart 12@3600
job 12@3600 requires 12 procs for 1:00:00
Earliest start in         00:01:39 on Wed Aug 31 16:30:45
Earliest completion in     1:01:39 on Wed Aug 31 17:30:45
Best Partition: 32Bit
```

> You cannot specify job flags when running showstart, and since a job by default can only run on one partition, showstart fails when querying for a job requiring more nodes than the largest partition available.

Additional Information

For reservation based estimates, the information provided by this command is more highly accurate if the job is highest priority, if the job has a reservation, or if the majority of the jobs which are of higher priority have reservations. Consequently, sites wishing to make decisions based on this information may want to consider using the RESERVATIONDEPTH parameter to increase the number of priority based reservations. This can be set so that most or even all idle jobs receive priority reservations and make the results of this command generally useful. The only caution of this approach is that increasing the **RESERVATIONDEPTH** parameter more tightly constrains the decisions of the scheduler and may resulting in slightly lower system utilization (typically less than 8% reduction).

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- checkjob
- showres
- showstats -f eststarttime
- showstats -f avgqtime
- Job Start Estimates

showstate

Synopsis

showstate

Overview

This command provides a summary of the state of the system. It displays a list of all active jobs and a text-based map of the status of all nodes and the jobs they are servicing. Basic diagnostic tests are also performed and any problems found are reported.

Access

By default, this command can be run by any Moab Administrator.

Examples

Example 3-62:

```
> showstate
cluster state summary for Wed Nov 23 12:00:21
    JobID              S      User    Group  Procs   Remaining            StartTime
    ------------------ - --------- -------- ----- -----------  -------------------
(A)      fr17n11.942.0 R     johns    staff    16    13:21:15      Nov 22 12:00:21
(B)      fr17n12.942.0 S     johns    staff    32    13:07:11      Nov 22 12:00:21
(C)      fr17n13.942.0 R     johns    staff     8    11:22:25      Nov 22 12:00:21
(D)      fr17n14.942.0 S     johns    staff     8    10:43:43      Nov 22 12:01:21
(E)      fr17n15.942.0 S     johns    staff     8     9:19:25      Nov 22 12:01:21
(F)      fr17n16.942.0 R     johns    staff     8     9:01:16      Nov 22 12:01:21
(G)      fr17n17.942.0 R     johns    staff     1     7:28:25      Nov 22 12:03:22
(H)      fr17n18.942.0 R     johns    staff     1     3:05:17      Nov 22 12:04:22
(I)      fr17n19.942.0 S     johns    staff    24     0:54:38      Nov 22 12:00:22
Usage Summary:  9 Active Jobs  106 Active Nodes
              [0][0][0][0][0][0][0][0][0][1][1][1][1][1][1][1]
              [1][2][3][4][5][6][7][8][9][0][1][2][3][4][5][6]
Frame      2: XXXXXXXXXXXXXXXXXXXXXXXX[ ][A][C][ ][A][C][C][A]
Frame      3: [ ][ ][ ][ ][ ][ ][A][ ][I][ ][I][ ][ ][ ][ ][ ]
Frame      4: [ ][I][ ][ ][ ][A][ ][I][ ][ ][ ][E][ ][I][ ][E]
Frame      5: [F][ ][E][ ][ ][ ][F][F][F][I][ ][ ][E][ ][E][E]
Frame      6: [ ][I][I][E][I][ ][I][I][ ][I][F][I][I][I][I][F]
Frame      7: [ ]XXX[ ]XXX[ ]XXX[ ]XXX[b]XXX[ ]XXX[ ]XXX[#]XXX
Frame      9: [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][E][ ]
Frame     11: [ ][ ][ ][ ][ ][ ][I][F][@][ ][A][I][ ][F][ ][A]
Frame     12: [A][ ][ ][A][ ][ ][C][A][ ][C][A][A][ ][ ][ ][ ]
Frame     13: [D]XXX[I]XXX[ ]XXX[ ]XXX[ ]XXX[ ]XXX[I]XXX[I]XXX
Frame     14: [D]XXX[I]XXX[I]XXX[D]XXX[ ]XXX[H]XXX[I]XXX[ ]XXX
Frame     15: [b]XXX[b]XXX[b]XXX[b]XXX[D]XXX[b]XXX[b]XXX[b]XXX
Frame     16: [b]XXX[ ]XXX[b]XXX[ ]XXX[b]XXX[b]XXX[ ]XXX[b]XXX
Frame     17: [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
Frame     21: [ ]XXX[b]XXX[b]XXX[ ]XXX[b]XXX[b]XXX[b]XXX[b]XXX
Frame     22: [b]XXX[b]XXX[b]XXX[ ]XXX[b]XXX[ ]XXX[b]XXX[b]XXX
Frame     27: [b]XXX[b]XXX[ ]XXX[b]XXX[b]XXX[b]XXX[b]XXX[b]XXX
Frame     28: [G]XXX[ ]XXX[D]XXX[ ]XXX[D]XXX[D]XXX[D]XXX[ ]XXX
Frame     29: [A][C][A][A][C][ ][A][C]XXXXXXXXXXXXXXXXXXXXXXXX
Key:  XXX:Unknown [*]:Down w/Job [#]:Down [']:Idle w/Job [ ]:Idle [@]:Busy w/No Job
[!]:Drained
Key:  [a]:(Any lower case letter indicates an idle node that is assigned to a job)

Check Memory on Node fr3n07
Check Memory on Node fr4n06
Check Memory on Node fr4n09
```

In this example, nine active jobs are running on the system. Each job listed in the top of the output is associated with a letter. For example, job `fr17n11.942.0` *is associated with the letter A. This letter can now be used to determine where the job is currently running. By looking at the system map, it can be found that job* `fr17n11.942.0` *(job A) is running on nodes* `fr2n10`, `fr2n13`, `fr2n16`, `fr3n07` *...*

The key at the bottom of the system map can be used to determine unusual node states. For example, `fr7n15` *is currently in the state down.*

After the key, a series of warning messages may be displayed indicating possible system problems. In this case, warning message indicate that there are memory problems on three nodes, `fr3n07`, `fr4n06`, *and* `fr4n09`*. Also, warning messages indicate that job* `fr15n09.1097.0` *is having difficulty starting. Node* `fr11n08` *is in state BUSY but has no job assigned to it (it possibly has a runaway job running on it).*

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- Specifying Node Rack/Slot Location

showstats

Synopsis

showstats

showstats -a [*accountid*] [-v] [-t <TIMESPEC>]

showstats -c [*classid*] [-v] [-t <TIMESPEC>]

showstats -f <*statistictype*>

showstats -g [*groupid*] [-v] [-t <TIMESPEC>]

showstats -j [*jobtemplate*] [-t <TIMESPEC>]

showstats -n [*nodeid*] [-t <TIMESPEC>]

showstats -q [*qosid*] [-v] [-t <TIMESPEC>]

showstats -s

showstats -T [*leafid* | *tree-level*]

showstats -u [*userid*] [-v] [-t <TIMESPEC>]

Overview

This command shows various accounting and resource usage statistics for the system. Historical statistics cover the timeframe from the most recent execution of the mschedctl -f command.

Access

By default, this command can be run by any Moab level 1, 2, or 3 Administrator.

Parameters

Flag	Description
-a[<ACCOUNTID>]	Display account statistics. See **Account statistics** on page 353 for an example.
-c[<CLASSID>]	Display class statistics
-f <statistictype>	Display full matrix statistics (see showstats -f for full details)

Flag	Description
-g[<GROUPID>]	Display group statistics. See **Group statistics** on page 355 for an example.
-j [<JOBTEMPLATE>]	Display template statistics
-n[<NODEID>]	Display node statistics (ENABLEPROFILING must be set). See **Node statistics** on page 357 for an example.
-q [<QOSID>]	Display QoS statistics
-s	display general scheduler statistics
-t	Display statistical information from the specified timeframe: `<START_TIME>[,<END_TIME>]` `(ABSTIME: [HH[:MM[:SS]]][_MO[/DD[/YY]]] ie 14:30_06/20)` `(RELTIME: -[[[DD:]HH:]MM:]SS)` See **Statistics from an absolute time frame** on page 363 and **Statistics from a relative time frame** on page 363 for examples. Profiling must be enabled for the credential type you want statistics for. See Credential Statistics for information on how to enable profiling. Also, **-t** is not a stand-alone option. It must be used in conjunction with the **-a**, **-c**, **-g**, **-n**, **-q**, or **-u** flag.
-T	Display fairshare tree statistics. See **Fairshare tree statistics** on page 362 for an example.
-u[<USERID>]	Display user statistics. See **User statistics** on page 359 for an example.
-v	Display verbose information. See **Verbose statistics** on page 357 for an example.

Examples

Example 3-63: Account statistics

```
> showstats -a
Account Statistics Initialized Tue Aug 26 14:32:39
              |----- Running ------|--------------------------------- Completed ------
----------------------------|
  Account      Jobs Procs ProcHours Jobs     %   PHReq     %    PHDed     %   FSTgt  AvgXF
 MaxXF  AvgQH  Effic  WCAcc
   137651        16    92   1394.52  229  39.15 18486  45.26 7003.5  41.54 40.00   0.77
  8.15   5.21  90.70  34.69
   462212        11    63    855.27   43   7.35  6028  14.76 3448.4  20.45  6.25   0.71
  5.40   3.14  98.64  40.83
   462213         6    72    728.12   90  15.38  5974  14.63 3170.7  18.81  6.25   0.37
  4.88   0.52  82.01  24.14
   005810         3    24    220.72   77  13.16  2537   6.21 1526.6   9.06 -----   1.53
 14.81   0.42  98.73  28.40
   175436         0     0      0.00   12   2.05  6013  14.72  958.6   5.69  2.50   1.78
  8.61   5.60  83.64  17.04
   000102         0     0      0.00    1   0.17    64   0.16    5.1   0.03 -----  10.85
 10.85  10.77  27.90   7.40
   000023         0     0      0.00    1   0.17    12   0.03    0.2   0.00 -----   0.04
  0.04   0.19  21.21   1.20
```

This example shows a statistical listing of all active accounts. The top line (Account Statistics Initialized...) of the output indicates the beginning of the timeframe covered by the displayed statistics.

The statistical output is divided into two categories, Running and Completed. Running statistics include information about jobs that are currently running. Completed statistics are compiled using historical information from both running and completed jobs.

The fields are as follows:

Column	Description
Account	Account Number
Jobs	Number of running jobs
Procs	Number of processors allocated to running jobs
ProcHours	Number of proc-hours required to complete running jobs
Jobs*	Number of jobs completed
%	Percentage of total jobs that were completed by account
PHReq*	Total proc-hours requested by completed jobs
%	Percentage of total proc-hours requested by completed jobs that were requested by account

Column	Description
PHDed	Total proc-hours dedicated to active and completed jobs. The proc-hours dedicated to a job are calculated by multiplying the number of allocated procs by the length of time the procs were allocated, regardless of the job's CPU usage.
%	Percentage of total proc-hours dedicated that were dedicated by account
FSTgt	Fairshare target. An account's fairshare target is specified in the `fs.cfg` file. This value should be compared to the account's node-hour dedicated percentage to determine if the target is being met.
AvgXF*	Average expansion factor for jobs completed. A job's XFactor (expansion factor) is calculated by the following formula: (QueuedTime + RunTime) / WallClockLimit.
MaxXF*	Highest expansion factor received by jobs completed
AvgQH*	Average queue time (in hours) of jobs
Effic	Average job efficiency. Job efficiency is calculated by dividing the actual node-hours of CPU time used by the job by the node-hours allocated to the job.
WCAcc*	Average wallclock accuracy for jobs completed. Wallclock accuracy is calculated by dividing a job's actual run time by its specified wallclock limit. A job's wallclock accuracy is capped at 100% so even if a job exceeds its requested walltime it will report an accuracy of 100%.

* These fields are empty until an account has completed at least one job.

Example 3-64: Group statistics

```
> showstats -g
Group Statistics Initialized Tue Aug 26 14:32:39
              |----- Running ------|--------------------------------- Completed ------
----------------------------|
GroupName  GID Jobs Procs ProcHours Jobs    %   PHReq    %    PHDed    %   FSTgt
AvgXF  MaxXF  AvgQH  Effic  WCAcc
     univ  214   16    92   1394.52  229  39.15 18486  45.26 7003.5  41.54 40.00
0.77   8.15   5.21  90.70  34.69
      daf  204   11    63    855.27   43   7.35  6028  14.76 3448.4  20.45  6.25
0.71   5.40   3.14  98.64  40.83
    dnavy  207    6    72    728.12   90  15.38  5974  14.63 3170.7  18.81  6.25
0.37   4.88   0.52  82.01  24.14
     govt  232    3    24    220.72   77  13.16  2537   6.21 1526.6   9.06 -----
1.53  14.81   0.42  98.73  28.40
      asp  227    0     0      0.00   12   2.05  6013  14.72  958.6   5.69  2.50
1.78   8.61   5.60  83.64  17.04
    derim  229    0     0      0.00   74  12.65   669   1.64  352.5   2.09 -----
0.50   1.93   0.51  96.03  32.60
   dchall  274    0     0      0.00    3   0.51   447   1.10  169.2   1.00 25.00
0.52   0.88   2.49  95.82  33.67
      nih  239    0     0      0.00   17   2.91   170   0.42  148.1   0.88 -----
0.95   1.83   0.14  97.59  84.31
    darmy  205    0     0      0.00   31   5.30   366   0.90   53.9   0.32  6.25
0.14   0.59   0.07  81.33  12.73
  systems   80    0     0      0.00    6   1.03    67   0.16   22.4   0.13 -----
4.07   8.49   1.23  28.68  37.34
      pdc  252    0     0      0.00    1   0.17    64   0.16    5.1   0.03 -----
10.85  10.85  10.77  27.90   7.40
    staff    1    0     0      0.00    1   0.17    12   0.03    0.2   0.00 -----
0.04   0.04   0.19  21.21   1.20
```

This example shows a statistical listing of all active groups. The top line (Group Statistics Initialized...) of the output indicates the beginning of the timeframe covered by the displayed statistics.

The statistical output is divided into two categories, Running and Completed. Running statistics include information about jobs that are currently running. Completed statistics are compiled using historical information from both running and completed jobs.

The fields are as follows:

Column	Description
GroupName	Name of group.
GID	Group ID of group.
Jobs	Number of running jobs.
Procs	Number of procs allocated to running jobs.
ProcHours	Number of proc hours required to complete running jobs.
Jobs*	Number of jobs completed.

Column	Description
%	Percentage of total jobs that were completed by group.
PHReq*	Total proc-hours requested by completed jobs.
%	Percentage of total proc-hours requested by completed jobs that were requested by group.
PHDed	Total proc-hours dedicated to active and completed jobs. The proc-hours dedicated to a job are calculated by multiplying the number of allocated procs by the length of time the procs were allocated, regardless of the job's CPU usage.
%	Percentage of total proc-hours dedicated that were dedicated by group.
FSTgt	Fairshare target. A group's fairshare target is specified in the `fs.cfg` file. This value should be compared to the group's node-hour dedicated percentage to determine if the target is being met.
AvgXF*	Average expansion factor for jobs completed. A job's XFactor (expansion factor) is calculated by the following formula: (QueuedTime + RunTime) / WallClockLimit.
MaxXF*	Highest expansion factor received by jobs completed.
AvgQH*	Average queue time (in hours) of jobs.
Effic	Average job efficiency. Job efficiency is calculated by dividing the actual node-hours of CPU time used by the job by the node-hours allocated to the job.
WCAcc*	Average wallclock accuracy for jobs completed. Wallclock accuracy is calculated by dividing a job's actual run time by its specified wallclock limit. A job's wallclock accuracy is capped at 100% so even if a job exceeds its requested walltime it will report an accuracy of 100%.

* These fields are empty until a group has completed at least one job.

Example 3-65: Node statistics

```
> showstats -n
node stats from Mon Jul 10 00:00:00 to Mon Jul 10 16:30:00
node       CfgMem MinMem MaxMem AvgMem | CfgProcs MinLoad MaxLoad AvgLoad
node01      58368      0  21122   5841        32    0.00   32.76   27.62
node02     122880      0  19466    220        30    0.00   33.98   29.54
node03      18432      0   9533   2135        24    0.00   25.10   18.64
node04      60440      0  17531   4468        32    0.00   30.55   24.61
node05      13312      0   2597   1189         8    0.00    9.85    8.45
node06      13312      0   3800   1112         8    0.00    8.66    5.27
node07      13312      0   2179   1210         8    0.00    9.62    8.27
node08      13312      0   3243   1995         8    0.00   11.71    8.02
node09      13312      0   2287   1943         8    0.00   10.26    7.58
node10      13312      0   2183   1505         8    0.00   13.12    9.28
node11      13312      0   3269   2448         8    0.00    8.93    6.71
node12      13312      0  10114   6900         8    0.00   13.13    8.44
node13      13312      0   2616   2501         8    0.00    9.24    8.21
node14      13312      0   3888    869         8    0.00    8.10    3.85
node15      13312      0   3788    308         8    0.00    8.40    4.67
node16      13312      0   4386   2191         7    0.00   18.37    8.36
node17      13312      0   3158   1870         8    0.00    8.95    5.91
node18      13312      0   5022   2397         8    0.00   19.25    8.19
node19      13312      0   2437   1371         8    0.00    8.98    7.09
node20      13312      0   4474   2486         8    0.00    8.51    7.11
node21      13312      0   4111   2056         8    0.00    8.93    6.68
node22      13312      0   5136   2313         8    0.00    8.61    5.75
node23      13312      0   1850   1752         8    0.00    8.39    5.71
node24      13312      0   3850   2539         8    0.00    8.94    7.80
node25      13312      0   3789   3702         8    0.00   21.22   12.83
node26      13312      0   3809   1653         8    0.00    9.34    4.91
node27      13312      0   5637     70         4    0.00   17.97    2.46
node28      13312      0   3076   2864         8    0.00   22.91   10.33
```

Example 3-66: Verbose statistics

```
> showstats -v
current scheduler time: Sat Aug 18 18:23:02 2007
moab active for      00:00:01   started on Wed Dec 31 17:00:00
statistics for iteration     0  initialized on Sat Aug 11 23:55:25
Eligible/Idle Jobs:                  6/8      (75.000%)
Active Jobs:                        13
Successful/Completed Jobs:         167/167    (100.000%)
Preempt Jobs:                        0
Avg/Max QTime (Hours):            0.34/2.07
Avg/Max XFactor:                  1.165/3.26
Avg/Max Bypass:                   0.40/8.00
Dedicated/Total ProcHours:      4.46K/4.47K  (99.789%)
Preempt/Dedicated ProcHours:      0.00/4.46K  (0.000%)
Current Active/Total Procs:         32/32     (100.0%)
Current Active/Total Nodes:         16/16      (100.0%)
Avg WallClock Accuracy:           64.919%
Avg Job Proc Efficiency:          99.683%
Min System Utilization:           87.323% (on iteration 46)
Est/Avg Backlog:                02:14:06/03:02:567
```

This example shows a concise summary of the system scheduling state. Note that showstats *and* showstats -s *are equivalent.*

The first line of output indicates the number of scheduling iterations performed by the current scheduling process, followed by the time the scheduler started. The second line indicates the amount of time the Moab Scheduler has been scheduling in HH:MM:SS notation followed by the statistics initialization time.

The fields are as follows:

Column	Description
Active Jobs	Number of jobs currently active (Running or Starting).
Eligible Jobs	Number of jobs in the system queue (jobs that are considered when scheduling).
Idle Jobs	Number of jobs both in and out of the system queue that are in the LoadLeveler Idle state.
Completed Jobs	Number of jobs completed since statistics were initialized.
Successful Jobs	Jobs that completed successfully without abnormal termination.
XFactor	Average expansion factor of all completed jobs.
Max XFactor	Maximum expansion factor of completed jobs.
Max Bypass	Maximum bypass of completed jobs.
Available ProcHours	Total proc-hours available to the scheduler.
Dedicated ProcHours	Total proc-hours made available to jobs.
Effic	Scheduling efficiency (DedicatedProcHours / Available ProcHours).
Min Efficiency	Minimum scheduling efficiency obtained since scheduler was started.
Iteration	Iteration on which the minimum scheduling efficiency occurred.
Available Procs	Number of procs currently available.
Busy Procs	Number of procs currently busy.
Effic	Current system efficiency (BusyProcs/AvailableProcs).
WallClock Accuracy	Average wallclock accuracy of completed jobs (job-weighted average).
Job Efficiency	Average job efficiency (UtilizedTime / DedicatedTime).

Column	Description
Est Backlog	Estimated backlog of queued work in hours.
Avg Backlog	Average backlog of queued work in hours.

Example 3-67: User statistics

```
> showstats -u
User Statistics Initialized Tue Aug 26 14:32:39
              |----- Running ------|--------------------------------- Completed ------
----------------------------|
 UserName  UID Jobs Procs ProcHours Jobs     %   PHReq     %    PHDed     %   FSTgt
AvgXF  MaxXF  AvgQH  Effic  WCAcc
  moorejt 2617    1    16     58.80    2   0.34    221   0.54 1896.6  11.25 -----
1.02   1.04   0.14  99.52 100.00
    zhong 1767    3    24    220.72   20   3.42   2306   5.65 1511.3   8.96 -----
0.71   0.96   0.49  99.37  67.48
      lui 2467    0     0      0.00   16   2.74   1970   4.82 1505.1   8.93 -----
1.02   6.33   0.25  98.96  57.72
    evans 3092    0     0      0.00   62  10.60   4960  12.14 1464.3   8.69   5.0
0.62   1.64   5.04  87.64  30.62
   wengel 2430    2    64    824.90    1   0.17    767   1.88  630.3   3.74 -----
0.18   0.18   4.26  99.63   0.40
    mukho 2961    2    16     71.06    6   1.03    776   1.90  563.5   3.34 -----
0.31   0.82   0.20  93.15  30.28
  jimenez 1449    1    16    302.29    2   0.34    768   1.88  458.3   2.72 -----
0.80   0.98   2.31  97.99  70.30
     neff 3194    0     0      0.00   74  12.65    669   1.64  352.5   2.09  10.0
0.50   1.93   0.51  96.03  32.60
   cholik 1303    0     0      0.00    2   0.34    552   1.35  281.9   1.67 -----
1.72   3.07  25.35  99.69  66.70
 jshoemak 2508    1    24    572.22    1   0.17    576   1.41  229.1   1.36 -----
0.55   0.55   3.74  99.20  39.20
     kudo 2324    1     8    163.35    6   1.03   1152   2.82  211.1   1.25 -----
0.12   0.34   1.54  96.77   5.67
   xztang 1835    1     8     18.99 ---- ------ ----- ------  176.3   1.05  10.0 -----
- ------ ------  99.62 ------
   feller 1880    0     0      0.00   17   2.91    170   0.42  148.1   0.88 -----
0.95   1.83   0.14  97.59  84.31
    maxia 2936    0     0      0.00    1   0.17    191   0.47  129.1   0.77   7.5
0.88   0.88   4.49  99.84  69.10
 ktgnov71 2838    0     0      0.00    1   0.17    192   0.47   95.5   0.57 -----
0.53   0.53   0.34  90.07  51.20
```

This example shows a statistical listing of all active users. The top line (User Statistics Initialized...) of the output indicates the timeframe covered by the displayed statistics.

The statistical output is divided into two statistics categories, Running and Completed. Running statistics include information about jobs that are currently running. Completed statistics are compiled using historical information from both running and completed jobs.

The fields are as follows:

Column	Description
UserName	Name of user.
UID	User ID of user.
Jobs	Number of running jobs.
Procs	Number of procs allocated to running jobs.
ProcHours	Number of proc-hours required to complete running jobs.
Jobs*	Number of jobs completed.
%	Percentage of total jobs that were completed by user.
PHReq*	Total proc-hours requested by completed jobs.
%	Percentage of total proc-hours requested by completed jobs that were requested by user.
PHDed	Total proc-hours dedicated to active and completed jobs. The proc-hours dedicated to a job are calculated by multiplying the number of allocated procs by the length of time the procs were allocated, regardless of the job's CPU usage.
%	Percentage of total proc-hours dedicated that were dedicated by user.
FSTgt	Fairshare target. A user's fairshare target is specified in the `fs.cfg` file. This value should be compared to the user's node-hour dedicated percentage to determine if the target is being met.
AvgXF*	Average expansion factor for jobs completed. A job's XFactor (expansion factor) is calculated by the following formula: (QueuedTime + RunTime) / WallClockLimit.
MaxXF*	Highest expansion factor received by jobs completed.
AvgQH*	Average queue time (in hours) of jobs.
Effic	Average job efficiency. Job efficiency is calculated by dividing the actual node-hours of CPU time used by the job by the node-hours allocated to the job.

Column	Description
WCAcc*	Average wallclock accuracy for jobs completed. Wallclock accuracy is calculated by dividing a job's actual run time by its specified wallclock limit. A job's wallclock accuracy is capped at 100% so even if a job exceeds its requested walltime it will report an accuracy of 100%.

* These fields are empty until a user has completed at least one job.

Example 3-68: Fairshare tree statistics

```
> showstats -T
statistics initialized Mon Jul 10 15:29:41
              |-------- Active ---------|--------------------------------- Completed
----------------------------------|
user            Jobs Procs ProcHours  Mem Jobs    %    PHReq    %    PHDed    %   FSTgt
  AvgXF  MaxXF  AvgQH  Effic  WCAcc
root               0     0      0.00    0   56 100.00  2.47K 100.00  1.58K  48.87 -----
   1.22   0.00   0.24 100.00  58.84
 l1.1              0     0      0.00    0   25  44.64 845.77  34.31 730.25  22.54 -----
   1.97   0.00   0.20 100.00  65.50
  Administrati     0     0      0.00    0   10  17.86 433.57  17.59 197.17   6.09 -----
   3.67   0.00   0.25 100.00  62.74
  Engineering      0     0      0.00    0   15  26.79 412.20  16.72 533.08  16.45 -----
   0.83   0.00   0.17 100.00  67.35
 l1.2              0     0      0.00    0   31  55.36  1.62K  65.69 853.00  26.33 -----
   0.62   0.00   0.27 100.00  53.46
  Shared           0     0      0.00    0    3   5.36  97.17   3.94  44.92   1.39 -----
   0.58   0.00   0.56 100.00  31.73
  Test             0     0      0.00    0    3   5.36  14.44   0.59  14.58   0.45 -----
   0.43   0.00   0.17 100.00  30.57
  Research         0     0      0.00    0   25  44.64  1.51K  61.16 793.50  24.49 -----
   0.65   0.00   0.24 100.00  58.82

> showstats -T 2
statistics initialized Mon Jul 10 15:29:41
              |-------- Active ---------|--------------------------------- Completed
----------------------------------|
user            Jobs Procs ProcHours  Mem Jobs    %    PHReq    %    PHDed    %   FSTgt
  AvgXF  MaxXF  AvgQH  Effic  WCAcc
  Test             0     0      0.00    0   22   4.99 271.27   0.55 167.42   0.19 -----
   3.86   0.00   2.89 100.00  60.76
  Shared           0     0      0.00    0   59  13.38 12.30K  24.75  4.46K   5.16 -----
   6.24   0.00  10.73 100.00  49.87
  Research         0     0      0.00    0  140  31.75  9.54K  19.19  5.40K   6.25 -----
   2.84   0.00   5.52 100.00  57.86
  Administrati     0     0      0.00    0   84  19.05  7.94K  15.96  4.24K   4.91 -----
   4.77   0.00   0.34 100.00  62.31
  Engineering      0     0      0.00    0  136  30.84 19.67K  39.56 28.77K  33.27 -----
   3.01   0.00   3.66 100.00  63.70

> showstats -T l1.1
statistics initialized Mon Jul 10 15:29:41
              |-------- Active ---------|--------------------------------- Completed
----------------------------------|
user            Jobs Procs ProcHours  Mem Jobs    %    PHReq    %    PHDed    %   FSTgt
  AvgXF  MaxXF  AvgQH  Effic  WCAcc
 l1.1              0     0      0.00    0  220  49.89 27.60K  55.52 33.01K  38.17 -----
   3.68   0.00   2.39 100.00  63.17
  Administrati     0     0      0.00    0   84  19.05  7.94K  15.96  4.24K   4.91 -----
   4.77   0.00   0.34 100.00  62.31
  Engineering      0     0      0.00    0  136  30.84 19.67K  39.56 28.77K  33.27 -----
   3.01   0.00   3.66 100.00  63.70
```

Example 3-69: Statistics from an absolute time frame

```
> showstats -c batch -v -t 00:00:01_01/01/13,23:59:59_12/31/13
statistics initialized Wed Jan 1 00:00:00

-------- Active ---------       ------------------------------------ Completed ------------
-----------------------

class Jobs Procs ProcHours Mem Jobs      %    PHReq      %    PHDed      %     FSTgt  AvgXF
MaxXF  AvgQH  Effic  WCAcc
batch    0     0      0.00   0   23  100.00    15  100.00     1  100.00  -----  0.40
5.01   0.00  88.94  39.87
```

Moab returns information about the class `batch` from January 1, 2013 to December 31, 2013. For more information about specifying absolute dates, see "Absolute Time Format" in **TIMESPEC** on page 366.

Example 3-70: Statistics from a relative time frame

```
> showstats -u bob -v -t -30:00:00:00
statistics initialized Mon Nov 11 15:30:00

-------- Active ---------       ------------------------------------ Completed ------------
-----------------------

user Jobs Procs ProcHours Mem Jobs      %    PHReq      %    PHDed      %     FSTgt  AvgXF
MaxXF  AvgQH  Effic  WCAcc
bob      0     0      0.00   0   23  100.00    15  100.00     1  100.00  -----  0.40
5.01   0.00  88.94  39.87
```

Moab returns information about user `bob` from the past 30 days. For more information about specifying relative dates, see "Relative Time Format" in **TIMESPEC** on page 366.

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mschedctl -f command - re-initialize statistics
- showstats -f command - display full matrix statistics

showstats -f

Synopsis

showstats -f *<statistictype>*

Overview

Shows table of various scheduler statistics.

This command displays a table of the selected Moab Scheduler statistics, such as expansion factor, bypass count, jobs, proc-hours, wallclock accuracy, and backfill information.

> Statistics are aggregated over time. This means statistical information is not available for time frames and the `-t` option is not supported with showstats -f.

Access

This command can be run by any Moab Scheduler Administrator.

Parameters

Parameter	Description
AVGBYPASS	Average bypass count. Includes summary of job-weighted expansion bypass and total samples.
AVGQTIME	Average queue time. Includes summary of job-weighted queue time and total samples.
AVGXFACTOR	Average expansion factor. Includes summary of job-weighted expansion factor, processor-weighted expansion factor, processor-hour-weighted expansion factor, and total number of samples.
BFCOUNT	Number of jobs backfilled. Includes summary of job-weighted backfill job percent and total samples.
BFPHRUN	Number of proc-hours backfilled. Includes summary of job-weighted backfill proc-hour percentage and total samples.
ESTSTARTTIME	Job start time estimate for jobs meeting specified processor/duration criteria. This estimate is based on the reservation start time analysis algorithm.
JOBCOUNT	Number of jobs. Includes summary of total jobs and total samples.
MAXBYPASS	Maximum bypass count. Includes summary of overall maximum bypass and total samples.
MAXXFACTOR	Maximum expansion factor. Includes summary of overall maximum expansion factor and total samples.
PHREQUEST	proc-hours requested. Includes summary of total proc-hours requested and total samples.
PHRUN	proc-hours run. Includes summary of total proc-hours run and total samples.
QOSDELIVERED	Quality of service delivered. Includes summary of job-weighted quality of service success rate and total samples.
WCACCURACY	Wallclock accuracy. Includes summary of overall wall clock accuracy and total samples.

Examples

Example 3-71:

```
> showstats -f AVGXFACTOR
Average XFactor Grid
[ NODES ][ 00:02:00 ][ 00:04:00 ][ 00:08:00 ][ 00:16:00 ][ 00:32:00 ][ 01:04:00 ][
02:08:00 ][ 04:16:00 ][ 08:32:00 ][ 17:04:00 ][ 34:08:00 ][   TOTAL  ]
[    1  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[    2  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[    4  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][
1.00   1][ -------- ][  1.12   2][ -------- ][ -------- ][  1.10   3]
[    8  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][
1.00   2][  1.24   2][ -------- ][ -------- ][ -------- ][  1.15   4]
[   16  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][  1.01   2][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][  1.01   2]
[   32  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[   64  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[  128  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[  256  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[ T TOT ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][  1.01   2][
1.00   3][  1.24   2][  1.12   2][ -------- ][ -------- ]
Job Weighted X Factor:     1.0888
Node Weighted X Factor:    1.1147
NS Weighted X Factor:      1.1900
Total Samples:                  9
```

The showstats -f command returns a table with data for the specified STATISTICTYPE parameter. The left-most column shows the maximum number of processors required by the jobs shown in the other columns. The column headers indicate the maximum wallclock time (in HH:MM:SS notation) requested by the jobs shown in the columns. The data returned in the table varies by the STATISTICTYPE requested. For table entries with one number, it is of the data requested. For table entries with two numbers, the left number is the data requested and the right number is the number of jobs used to calculate the average. Table entries that contain only dashes (-------) indicate no job has completed that matches the profile associated for this inquiry. The bottom row shows the totals for each column. Following each table is a summary, which varies by the STATISTICTYPE requested.

The column and row break down can be adjusted using the STATPROC and STATTIME* parameters respectively.*

This particular example shows the average expansion factor grid. Each table entry indicates two pieces of information — the average expansion factor for all jobs that meet this slot's profile and the number of jobs that were used to calculate this average. For example, the XFactors of two jobs were averaged to obtain an average XFactor of 1.24 for jobs requiring over 2 hours 8 minutes, but not more than 4 hours 16 minutes and between 5 and 8 processors. Totals along the bottom provide overall XFactor averages weighted by job, processors, and processor-hours.

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mschedctl -f command
- showstats command
- STATPROCMIN parameter
- STATPROCSTEPCOUNT parameter
- STATPROCSTEPSIZE parameter
- STATTIMEMIN parameter

- STATTIMESTEPCOUNT parameter
- STATTIMESTEPSIZE parameter

TIMESPEC

Relative Time Format

The relative time format specifies a time by using the current time as a reference and specifying a time offset.

Format

+[[[DD:]HH:]MM:]SS

Examples

2 days, 3 hours and 57 seconds in the future:

```
+02:03:0:57
```

21 days (3 weeks) in the future:

```
+21:0:0:0
```

30 seconds in the future:

```
+30
```

Absolute Time Format

The absolute time format specifies a specific time in the future.

Format

[HH[:MM[:SS]]][_MO[/DD[/YY]]] i.e. 14:30_06/20)

Examples

1 PM, March 1 (this year)

```
13:00_03/01
```

Deprecated commands

canceljob

> This command is deprecated. Use mjobctl -c instead.

Synopsis

canceljob *jobid* [*jobid*]...

Overview

The canceljob command is used to selectively cancel the specified job(s) (active, idle, or non-queued) from the queue.

Access

This command can be run by any Moab Administrator and by the owner of the job (see ADMINCFG).

Flag	Name	Format	Default	Description	Example
-h	HELP		N/A	Display usage information	`> canceljob -h`
	JOB ID	<STRING>	---	a jobid, a job expression, or the keyword ALL	`> canceljob 13001 13003`

Examples

Example 3-72: Cancel job 6397

```
> canceljob 6397
```

changeparam

This command is deprecated. Use mschedctl -m instead.

Synopsis

changeparam*parametervalue*

Overview

The changeparam command is used to dynamically change the value of any parameter which can be specified in the `moab.cfg` file. The changes take effect at the beginning of the next scheduling iteration. They are not persistent, only lasting until Moab is shut down.

changeparam is a compact command of mschedctl -m.

Access

This command can be run by a level 1 Moab administrator.

diagnose

This command is deprecated. Use mdiag instead.

Synopsis

diagnose -a [*accountid*]

diagnose -b [-l *policylevel*] [-t *partition*]

diagnose -c [*classid*]

diagnose -C [*configfile*]

diagnose -f [-o user|group|account|qos|class]

diagnose -g [*groupid*]

diagnose -j [*jobid*]

diagnose -L

diagnose -m [*rackid*]

diagnose -n [-t *partition*] [*nodeid*]

diagnose -p [-t *partition*]

diagnose -q [*qosid*]

diagnose -r [*reservationid*]

diagnose -R [*resourcemanagername*]

diagnose -s [*standingreservationid*]

diagnose -S diagnose -u [*userid*]

diagnose -v

diagnose -x

Overview

The diagnose command is used to display information about various aspects of scheduling and the results of internal diagnostic tests.

releasehold

This command is deprecated. Use mjobctl -u instead.

Synopsis

releasehold [-a|-b] jobexp

Overview

Release hold on specified job(s).

This command allows you to release batch holds or all holds (system, user, and batch) on specified jobs. Any number of jobs may be released with this command.

Access

By default, this command can be run by any Moab Scheduler Administrator.

Parameters

JOBEXP	Job expression of job(s) to release.

Flags

-a	Release all types of holds (user, system, batch) for specified job(s).
-b	Release batch hold from specified job(s).
-h	Help for this command.

Examples

Example 3-73: releasehold -b

```
> releasehold -b 6443
batch hold released for job 6443
```

In this example, a batch hold was released from this one job.

Example 3-74: releasehold -a

```
> releasehold -a "81[1-6]"
holds modified for job 811
holds modified for job 812
holds modified for job 813
holds modified for job 814
holds modified for job 815
holds modified for job 816
```

In this example, all holds were released from the specified jobs.

Related topics

- sethold
- mjobctl

releaseres

This command is deprecated. Use mrsvctl -r instead.

Synopsis

releaseres [*arguments*] *reservationid* [*reservationid*...]

Overview

Release existing reservation.

This command allows Moab Scheduler Administrators to release any user, group, account, job, or system reservation. Users are allowed to release reservations on jobs they own. Note that releasing a reservation on an active job has no effect since the reservation will be automatically recreated.

Access

Users can use this command to release any reservation they own. Level 1 and level 2 Moab administrators may use this command to release any reservation.

Parameters

RESERVATION ID	Name of reservation to release.

Examples

Example 3-75: Release two existing reservations

```
> releaseres system.1 bob.2
released User reservation 'system.1'
released User reservation 'bob.2'
```

resetstats

This command is deprecated. Use mschedctl -f instead.

Synopsis

resetstats

Overview

This command resets all internally-stored Moab Scheduler statistics to the initial start-up state as of the time the command was executed.

Access

By default, this command can be run by level 1 scheduler administrators.

Examples

Example 3-76:

```
> resetstats Statistics Reset at time Wed Feb 25 23:24:55 2011
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes

runjob

This command is deprecated. Use mjobctl -x instead.

Synopsis

runjob [-c|-f|-n *nodelist*|-p *partition*|-s|-x] *jobid*

Overview

This command will attempt to immediately start the specified job.

runjob is a deprecated command, replaced by mjobctl.

Access

By default, this command can be run by any Moab administrator.

Parameters

JOBID	Name of the job to run.

Args	Description
-c	*Clear* job parameters from previous runs (used to clear PBS neednodes attribute after PBS job launch failure)
-f	Attempt to *force* the job to run, ignoring throttling policies
-n <NODELIST>	Attempt to start the job using the specified *nodelist* where nodenames are comma or colon delimited

Args	Description
-p <PARTITION>	Attempt to start the job in the specified *partition*
-s	Attempt to *suspend* the job
-x	Attempt to force the job to run, ignoring throttling policies, QoS constraints, and reservations

Examples

Example 3-77: Run job `cluster.231`

```
> runjob cluster.231
job cluster.231 successfully started
```

See Also

- mjobctl
- canceljob - cancel a job.
- checkjob - show detailed status of a job.
- showq - list queued jobs.

sethold

This command is deprecated. Use mjobctl -h instead.

Synopsis

sethold [-b] *jobid* [*jobid*...]

Overview

Set hold on specified job(s).

Permissions

This command can be run by any Moab Scheduler Administrator.

Parameters

JOB	Job number of job to hold.

Flags

-b	Set a batch hold. Typically, only the scheduler places batch holds. This flag allows an administrator to manually set a batch hold.
-h	Help for this command.

Examples

Example 3-78:

```
> sethold -b fr17n02.1072.0 fr15n03.1017.0
Batch Hold Placed on All Specified Jobs
```

In this example, a batch hold is placed on job `fr17n02.1072.0` and job `fr15n03.1017.0`.

setqos

This command is deprecated. Use mjobctl -m instead.

Synopsis

setqos*qosidjobid*

Overview

Set Quality Of Service for a specified job.

This command allows users to change the QOS of their own jobs.

Access

This command can be run by any user.

Parameters

JOBID	Job name.
QOSID	QOS name.

Examples

Example 3-79:

```
> setqos high_priority moab.3

Job QOS Adjusted
```

This example sets the Quality Of Service to a value of `high_priority` *for job* `moab.3`*.*

setres

This command is deprecated. Use mrsvctl -c instead.

Synopsis

setres [*arguments*] *resourceexpression*
[-a <ACCOUNT_LIST>]
[-b <SUBTYPE>]
[-c <CHARGE_SPEC>]
[-d <DURATION>]
[-e <ENDTIME>]
[-E] // EXCLUSIVE
[-f <FEATURE_LIST>]
[-g <GROUP_LIST>]
[-n <NAME>]
[-o <OWNER>]
[-p <PARTITION>]
[-q <QUEUE_LIST>] // (i.e. CLASS_LIST)
[-Q <QOSLIST>]
[-r <RESOURCE_DESCRIPTION>]
[-R <RESERVATION_PROFILE>]
[-s <STARTTIME>]
[-T <TRIGGER>]
[-u <USER_LIST>]
[-x <FLAGS>]

Overview

Reserve resources for use by jobs with particular credentials or attributes.

Access

This command can be run by level 1 and level 2 Moab administrators.

Parameters

Name	Format	Default	Description
ACCOUNT_LIST	*<STRING>*[:*<STRING>*]...	---	List of accounts that will be allowed access to the reserved resources
SUBTYPE	*<STRING>*	---	Specify the subtype for a reservation
CHARGE_SPEC	*<ACCOUNT>*[,*<GROUP>*[,*<USER>*]]	---	Specifies which credentials will be accountable for unused resources dedicated to the reservation
CLASS_LIST	*<STRING>*[:*<STRING>*]...	---	List of classes that will be allowed access to the reserved resource
DURATION	[[[DD:]HH:]MM:]SS	INFINITY	Duration of the reservation (not needed if ENDTIME is specified)
ENDTIME	[HH[:MM[:SS]]][_MO[/DD[/YY]]] or +[[[DD:]HH:]MM:]SS	INFINITY	Absolute or relative time reservation will end (not required if Duration specified)
EXCLUSIVE	N/A	N/A	Requests exclusive access to resources
FEATURE_LIST	*<STRING>*[:*<STRING>*]...	---	List of node features which must be possessed by the reserved resources
FLAGS	*<STRING>*[:*<STRING>*]...	---	List of reservation flags (See Managing Reservations for details)
GROUP_LIST	*<STRING>*[:*<STRING>*]...	---	List of groups that will be allowed access to the reserved resources
NAME	*<STRING>*	Name set to first name listed in ACL or `SYSTEM` if no ACL specified	Name for new reservation

Name	Format	Default	Description
OWNER	*<CREDTYPE>* :*<CREDID>* where *CREDTYPE* is one of user, group, acct, class, or qos	N/A	Specifies which credential is granted reservation ownership privileges
PARTITION	*<STRING>*	[ANY]	Partition in which resources must be located
QOS_LIST	*<STRING>* [:*<STRING>*]...	---	List of QOS's that will be allowed access to the reserved resource
RESERVATION_ PROFILE	Existing reservation profile ID	N/A	Requests that default reservation attributes be loaded from the specified reservation profile (see RSVPROFILE)
RESOURCE_ DESCRIPTION	Colon delimited list of zero or more of the following *<ATTR>*=*<VALUE>* pairs PROCS=*<INTEGER>* MEM=*<INTEGER>* DISK=*<INTEGER>* SWAP=*<INTEGER>* GRES=*<STRING>*	PROCS=-1	Specifies the resources to be reserved per task. (-1 indicates all resources on node)
RESOURCE_ EXPRESSION	ALL or TASKS{==|>=}*<TASKCOUNT>* or *<HOST_REGEX>*	Required Field. No Default	Specifies the tasks to reserve. ALL indicates all resources available should be reserved. If ALL or a host expression is specified, Moab will apply the reservation regardless of existing reservations and exclusive issues. If TASKS is used, Moab will only allocate accessible resources.
STARTTIME	[HH[:MM[:SS]]][_MO[/DD[/YY]]] or +[[[DD:]HH:]MM:]SS	NOW	Absolute or relative time reservation will start

Name	Format	Default	Description
TRIGGER	<STRING>	N/A	Comma delimited reservation trigger list following format described in the trigger format section of the reservation configuration overview.
USER_LIST	<STRING>[:<STRING>]...	---	List of users that will be allowed access to the reserved resources

Description

The setres command allows an arbitrary block of resources to be reserved for use by jobs which meet the specified access constraints. The timeframe covered by the reservation can be specified on either an absolute or relative basis. Only jobs with credentials listed in the reservation ACL (i.e., **USERLIST**, **GROUPLIST**,...) can utilize the reserved resources. However, these jobs still have the freedom to utilize resources outside of the reservation. The reservation will be assigned a name derived from the ACL specified. If no reservation ACL is specified, the reservation is created as a system reservation and no jobs will be allowed access to the resources during the specified timeframe (valuable for system maintenance, etc.). See the Reservation Overview for more information.

Reservations can be viewed using the showres command and can be released using the releaseres command.

Examples

Example 3-80:

```
> setres -u john:mary -s +24:00:00 -d 8:00:00 TASKS==2
reservation 'john.1' created on 2 nodes (2 tasks)
node001:1
node005:1
```

Reserve two nodes for use by users `john` and `mary` for a period of 8 hours starting in 24 hours.

Example 3-81:

```
> setres -s 8:00:00_06/20 -e 17:00:00_06/22 ALL
reservation 'system.1' created on 8 nodes (8 tasks)
node001:1
node002:1
node003:1
node004:1
node005:1
node006:1
node007:1
node008:1
```

Schedule a system wide reservation to allow system maintenance on Jun 20, 8:00 AM until Jun 22, 5:00 PM.

Example 3-82:

```
> setres -r PROCS=1:MEM=512 -g staff -l interactive 'node00[3-6]'
reservation 'staff.1' created on 4 nodes (4 tasks)
node003:1
node004:1
node005:1
node006:1
```

Reserve one processor and 512 MB of memory on nodes node003 through node node006 for members of the group staff and jobs in the interactive class.

setspri

This command is deprecated. Use mjobctl -p instead.

Synopsis

setspri [-r] *priorityjobid*

Overview

(This command is deprecated by the mjobctl command)

Set or remove absolute or relative system priorities for a specified job.

This command allows you to set or remove a system priority level for a specified job. Any job with a system priority level set is guaranteed a higher priority than jobs without a system priority. Jobs with higher system priority settings have priority over jobs with lower system priority settings.

Access

This command can be run by any Moab Scheduler Administrator.

Parameters

JOB	Name of job.
PRIORITY	System priority level. By default, this priority is an absolute priority overriding the policy generated priority value. Range is 0 to clear, 1 for lowest, 1000 for highest. The given value is added onto the system priority (see 32-bit and 64-bit values below), except for a given value of zero. If the '-r' flag is specified, the system priority is relative, adding or subtracting the specified value from the policy generated priority. If a relative priority is specified, any value in the range +/- 1,000,000,000 is acceptable.

Flags

-r	Set relative system priority on job.

Examples

Example 3-83:

```
> setspri 10 orion.4752
job system priority adjusted
```

In this example, a system priority of `10` is set for job `orion.4752`.

Example 3-84:

```
> setspri 0 clusterB.1102
job system priority adjusted
```

In this example, system priority is cleared for job `clusterB.1102`.

Example 3-85:

```
> setspri -r 100000 job.00001
job system priority adjusted
```

In this example, the job's priority will be increased by `100000` over the value determine by configured priority policy.

This command is deprecated. Use mjobctl instead.

showconfig

This command is deprecated. Use mschedctl -l instead.

Synopsis

showconfig [-v]

Overview

View the current configurable parameters of the Moab Scheduler.

The showconfig command shows the current scheduler version and the settings of all "in memory" parameters. These parameters are set via internal defaults, command line arguments, environment variable settings, parameters in the `moab.cfg` file, and via the mschedctl -m command. Because of the many sources of configuration settings, the output may differ from the contents of the `moab.cfg` file. The output is such that it can be saved and used as the contents of the `moab.cfg` file if desired.

Access

This command can be run by a level 1, 2, or 3 Moab administrator.

Flags

-h	Help for this command.
-v	This optional flag turns on verbose mode, which shows all possible Moab Scheduler parameters and their current settings. If this flag is not used, this command operates in context-sensitive terse mode, which shows only relevant parameter settings.

Examples

Example 3-86: showconfig

```
> showconfig
# moab scheduler version 4.2.4 (PID: 11080)
BACKFILLPOLICY                  FIRSTFIT
BACKFILLMETRIC                  NODES
ALLOCATIONPOLICY                MINRESOURCE
RESERVATIONPOLICY               CURRENTHIGHEST
...
```

The showconfig command without the `-v` flag does not show the settings of all parameters. It does show all major parameters and all parameters which are in effect and have been set to non-default values. However, it hides other rarely used parameters and those which currently have no effect or are set to default values. To show the settings of all parameters, use the `-v` (verbose) flag. This will provide an extended output. This output is often best used in conjunction with the grep command as the output can be voluminous.

Related topics

- Use the `mschedctl -m` command to change the various Moab Scheduler parameters.
- See the Parameters document for details about configurable parameters.

Prioritizing Jobs and Allocating Resources

Job Prioritization

In general, prioritization is the process of determining which of many options best fulfills overall goals. In the case of scheduling, a site will often have multiple, independent goals that may include maximizing system utilization, giving preference to users in specific projects, or making certain that no job sits in the queue for more than a given period of time. The approach used by Moab in representing a multi-faceted set of site goals is to assign weights to the various objectives so an overall value or priority can be associated with each potential scheduling decision. With the jobs prioritized, the scheduler can roughly fulfill site objectives by starting the jobs in priority order.

Related topics

- mdiag -p (Priority Diagnostics)

Priority Overview

Moab's prioritization mechanism allows component and subcomponent weights to be associated with many aspects of a job to enable fine-grained control over this aspect of scheduling. To allow this level of control, Moab uses a simple priority-weighting hierarchy where the contribution of each priority subcomponent is calculated as follows:

<COMPONENT WEIGHT> * <SUBCOMPONENT WEIGHT> * <PRIORITY SUBCOMPONENT VALUE>

Each priority component contains one or more subcomponents as described in the section titled **Job Priority Factors** on page 382. For example, the Resource component consists of Node, Processor, Memory, Swap, Disk, Walltime, and PE subcomponents. While there are numerous priority components and many more subcomponents, a site need only focus on and configure the subset of components related to their particular priority needs. In actual usage, few sites use more than a small fraction (usually 5 or fewer) of the available priority subcomponents. This results in fairly straightforward priority configurations and tuning. By mixing and matching priority weights, sites may generally obtain the desired job-start behavior. At any time, you can issue the mdiag -p command to determine the impact of the current priority-weight settings on idle jobs. Likewise, the command showstats -f can assist the administrator in evaluating priority effectiveness on historical system usage metrics such as queue time or expansion factor.

As mentioned above, a job's priority is the weighted sum of its activated subcomponents. By default, the value of all component and subcomponent weights is set to 1 and 0 respectively. The one exception is the "QUEUETIME" subcomponent weight that is set to 1. This results in a total job priority equal to the period of time the job has been queued, causing Moab to act as a simple FIFO. Once the summed component weight is determined, this value is then bounded resulting in a priority ranging between 0 and MAX_PRIO_VAL which is currently defined as 1000000000 (one billion). In no case will a job obtain a priority in excess of MAX_PRIO_VAL through its priority subcomponent values.

> Negative priority jobs may be allowed if desired; see ENABLENEGJOBPRIORITY and REJECTNEGPRIOJOBS for more information.

Using the mjobctl -p command, site administrators may adjust the base calculated job priority by either assigning a relative priority adjustment or an absolute system priority. A relative priority adjustment causes the base priority to be increased or decreased by a specified value. Setting an absolute system priority, SPRIO, causes the job to receive a priority equal to MAX_PRIO_VAL + SPRIO, and thus guaranteed to be of higher value than any naturally occurring job priority.

Related topics

- REJECTNEGPRIOJOBS parameter

Job Priority Factors

- Credential (CRED) Component
- Fairshare (FS) Component
- Resource (RES) Component
- Service (SERVICE) Component
- Target Service (TARG) Component
- Usage (USAGE) Component
- Job Attribute (ATTR) Component

Moab allows jobs to be prioritized based on a range of job related factors. These factors are broken down into a two-tier hierarchy of priority factors and subfactors, each of which can be independently assigned a weight. This approach provides the administrator with detailed yet straightforward control of the job selection process.

Each factor and subfactor can be configured with independent priority weight and priority cap values (described later). In addition, per credential and per QoS priority weight adjustments may be specified for a subset of the priority factors. For example, QoS credentials can adjust the queuetime subfactor weight and group credentials can adjust fairshare subfactor weight.

The following table highlights the factors and subfactors that make up a job's total priority.

Factor	SubFactor	Metric
CRED (job credentials)	USER	user-specific priority (See USERCFG)
	GROUP	group-specific priority (See GROUPCFG)
	ACCOUNT	account-specific priority (See ACCOUNTCFG)
	QOS	QoS-specific priority (See QOSCFG)
	CLASS	class/queue-specific priority (See CLASSCFG)

Factor	SubFactor	Metric
FS (fairshare usage)	FSUSER	user-based historical usage (See Fairshare Overview)
	FSGROUP	group-based historical usage (See Fairshare Overview)
	FSACCOUNT	account-based historical usage (See Fairshare Overview)
	FSQOS	QoS-based historical usage (See Fairshare Overview)
	FSCLASS	class/queue-based historical usage (See Fairshare Overview)
	FSGUSER	imported global user-based historical usage (See ID Manager and Fairshare Overview)
	FSGGROUP	imported global group-based historical usage (See ID Manager and Fairshare Overview)
	FSGACCOUNT	imported global account-based historical usage (See ID Manager and Fairshare Overview)
	FSJPU	current active jobs associated with job user
	FSPPU	current number of processors allocated to active jobs associated with job user
	FSPSPU	current number of processor-seconds allocated to active jobs associated with job user
	WCACCURACY	user's current historical job wallclock accuracy calculated as total processor-seconds dedicated / total processor-seconds requested Factor values are in the range of 0.0 to 1.0.

Factor	SubFactor	Metric
RES (requested job resources)	NODE	number of nodes requested
	PROC	number of processors requested
	MEM	total real memory requested (in MB)
	SWAP	total virtual memory requested (in MB)
	DISK	total local disk requested (in MB)
	PS	total processor-seconds requested
	PE	total processor-equivalent requested
	WALLTIME	total walltime requested (in seconds)
SERV (current service levels)	QUEUETIME	time job has been queued (in minutes)
	XFACTOR	minimum job expansion factor
	BYPASS	number of times job has been bypassed by backfill
	STARTCOUNT	number of times job has been restarted
	DEADLINE	proximity to job deadline
	SPVIOLATION	Boolean indicating whether the active job violates a soft usage limit
	USERPRIO	user-specified job priority
TARGET (target service levels)	TARGETQUEUETIME	time until queuetime target is reached (exponential)
	TARGETXFACTOR	distance to target expansion factor (exponential)

Factor	SubFactor	Metric
USAGE (consumed resources -- active jobs only)	CONSUMED	processor-seconds dedicated to date
	REMAINING	processor-seconds outstanding
	PERCENT	percent of required walltime consumed
	EXECUTIONTIME	seconds since job started
ATTR (job attribute-based prioritization)	ATTRATTR	Attribute priority if specified job attribute is set (attributes may be user-defined or one of **preemptor**, or **preemptee**). Default is 0.
	ATTRSTATE	Attribute priority if job is in specified state (see Job States). Default is 0.
	ATTRGRES	Attribute priority if a generic resource is requested. Default is 0.

*CAP parameters (FSCAP, for example) are available to limit the maximum absolute value of each priority component and subcomponent. If set to a positive value, a priority cap will bound priority component values in both the positive and negative directions.

All *CAP and *WEIGHT parameters are specified as positive or negative integers. Non-integer values are not supported.

Credential (CRED) Component

The credential component allows a site to prioritize jobs based on political issues such as the relative importance of certain groups or accounts. This allows direct political priorities to be applied to jobs.

The priority calculation for the credential component is as follows:

Priority += CREDWEIGHT * (
USERWEIGHT * Job.User.Priority +
GROUPWEIGHT * Job.Group.Priority +
ACCOUNTWEIGHT * Job.Account.Priority +
QOSWEIGHT * Job.Qos.Priority +
CLASSWEIGHT * Job.Class.Priority)

All user, group, account, QoS, and class weights are specified by setting the **PRIORITY** attribute of using the respective *CFG parameter (namely, USERCFG, GROUPCFG, ACCOUNTCFG, QOSCFG, and CLASSCFG).

For example, to set user and group priorities, you might use the following:

```
CREDWEIGHT      1
USERWEIGHT      1
GROUPWEIGHT     1
USERCFG[john]   PRIORITY=2000
USERCFG[paul]   PRIORITY=-1000
GROUPCFG[staff] PRIORITY=10000
```

> Class (or queue) priority may also be specified via the resource manager where supported (as in PBS queue priorities). However, if Moab class priority values are also specified, the resource manager priority values will be overwritten.

All priorities may be positive or negative.

Fairshare (FS) Component

Fairshare components allow a site to favor jobs based on short-term historical usage. The Fairshare Overview describes the configuration and use of fairshare in detail.

The fairshare factor is used to adjust a job's priority based on current and historical percentage system utilization of the job's user, group, account, class, or QoS. This allows sites to steer workload toward a particular usage mix across user, group, account, class, and QoS dimensions.

The fairshare priority factor calculation is as follows:

Priority += FSWEIGHT * MIN(FSCAP, (
 FSUSERWEIGHT * DeltaUserFSUsage +
 FSGROUPWEIGHT * DeltaGroupFSUsage +
 FSACCOUNTWEIGHT * DeltaAccountFSUsage +
 FSQOSWEIGHT * DeltaQOSFSUsage +
 FSCLASSWEIGHT * DeltaClassFSUsage +
 FSJPUWEIGHT * ActiveUserJobs +
 FSPPUWEIGHT * ActiceUserProcs +
 FSPSPUWEIGHT * ActiveUserPS +
 WCACCURACYWEIGHT * UserWCAccuracy))

All ***WEIGHT** parameters just listed are specified on a per partition basis in the `moab.cfg` file. The `Delta*Usage` components represent the difference in actual fairshare usage from the corresponding fairshare usage target. Actual fairshare usage is determined based on historical usage over the time frame specified in the fairshare configuration. The target usage can be a target, floor, or ceiling value as specified in the fairshare configuration file. See the Fairshare Overview for further information on configuring and tuning fairshare. Additional insight may be available in the fairshare usage example. The `ActiveUser*` components represent current usage by the job's user credential.

How violated ceilings and floors affect fairshare-based priority

Moab determines `FSUsageWeight` in the previous section. In order to account for violated ceilings and floors, Moab multiplies that number by the `FSUsagePriority` as demonstrated in the following formula:

`FSPriority = FSUsagePriority * FSUsageWeight`

When a ceiling or floor is violated, `FSUsagePriority = 0`, so `FSPriority = 0`. This means the job will gain no priority because of fairshare. If fairshare is the only component of priority, then violation takes the priority to 0. For more information, see **Priority-Based Fairshare** on page 443 and **Fairshare Targets** on page 440.

Resource (RES) Component

Weighting jobs by the amount of resources requested allows a site to favor particular types of jobs. Such prioritization may allow a site to better meet site mission objectives, improve fairness, or even improve overall system utilization.

Resource based prioritization is valuable when you want to favor jobs based on the resources requested. This is good in three main scenarios: (1) when you need to favor large resource jobs because it's part of your site's mission statement, (2) when you want to level the response time distribution across large and small jobs (small jobs are more easily backfilled and thus generally have better turnaround time), and (3) when you want to improve system utilization. While this may be surprising, system utilization actually increases as large resource jobs are pushed to the front of the queue. This keeps the smaller jobs in the back where they can be selected for backfill and thus increase overall system utilization. The situation is like the story about filling a cup with golf balls and sand. If you put the sand in first, it gets in the way and you are unable to put in as many golf balls. However, if you put in the golf balls first, the sand can easily be poured in around them completely filling the cup.

The calculation for determining the total resource priority factor is as follows:

Priority += RESWEIGHT* MIN(RESCAP, (
  NODEWEIGHT * TotalNodesRequested +
  PROCWEIGHT * TotalProcessorsRequested +
  MEMWEIGHT * TotalMemoryRequested +
  SWAPWEIGHT * TotalSwapRequested +
  DISKWEIGHT * TotalDiskRequested +
  WALLTIMEWEIGHT* TotalWalltimeRequested +
  PEWEIGHT * TotalPERequested))

The sum of all weighted resources components is then multiplied by the **RESWEIGHT** parameter and capped by the **RESCAP** parameter. Memory, Swap, and Disk are all measured in megabytes (MB). The final resource component, PE, represents Processor Equivalents. This component can be viewed as a processor-weighted maximum *percentage of total resources* factor.

For example, if a job requested 25% of the processors and 50% of the total memory on a 128-processor system, it would have a PE value of MAX(25,50) * 128, or 64. The concept of PEs is a highly effective metric in shared resource systems.

> Ideal values for requested job processor count and walltime can be specified using PRIORITYTARGETPROCCOUNT and PRIORITYTARGETDURATION.

Service (SERVICE) Component

The Service component specifies which service metrics are of greatest value to the site. Favoring one service subcomponent over another generally improves that service metric.

The priority calculation for the service priority factor is as follows:

Priority += SERVICEWEIGHT * (
  QUEUETIMEWEIGHT * <QUEUETIME> +

XFACTORWEIGHT * <XFACTOR> +
BYPASSWEIGHT * <BYPASSCOUNT> +
STARTCOUNTWEIGHT * <STARTCOUNT> +
DEADLINEWEIGHT * <DEADLINE> +
SPVIOLATIONWEIGHT * <SPBOOLEAN> +
USERPRIOWEIGHT * <USERPRIO>)

QueueTime (QUEUETIME) Subcomponent

In the priority calculation, a job's queue time is a duration measured in minutes. Using this subcomponent tends to prioritize jobs in a FIFO order. Favoring queue time improves queue time based fairness metrics and is probably the most widely used single job priority metric. In fact, under the initial default configuration, this is the only priority subcomponent enabled within Moab. It is important to note that within Moab, a job's queue time is not necessarily the amount of time since the job was submitted. The parameter JOBPRIOACCRUALPOLICY allows a site to select how a job will accrue queue time based on meeting various throttling policies. Regardless of the policy used to determine a job's queue time, this effective queue time is used in the calculation of the QUEUETIME, XFACTOR, TARGETQUEUETIME, and TARGETXFACTOR priority subcomponent values.

The need for a distinct effective queue time is necessitated by the fact that many sites have users who like to work the system, whatever system it happens to be. A common practice at some long existent sites is for some users to submit a large number of jobs and then place them on hold. These jobs remain with a hold in place for an extended period of time and when the user is ready to run a job, the needed executable and data files are linked into place and the hold released on one of these pre-submitted jobs. The extended hold time guarantees that this job is now the highest priority job and will be the next to run. The use of the **JOBPRIOACCRUALPOLICY** parameter can prevent this practice and prevent "queue stuffers" from doing similar things on a shorter time scale. These "queue stuffer" users submit hundreds of jobs at once to swamp the machine and consume use of the available compute resources. This parameter prevents the user from gaining any advantage from stuffing the queue by not allowing these jobs to accumulate any queue time based priority until they meet certain idle and active Moab fairness policies (such as max job per user and max idle job per user).

As a final note, you can adjust the QUEUETIMEWEIGHT parameter on a per QoS basis using the QOSCFG parameter and the **QTWEIGHT** attribute. For example, the line **QOSCFG**[*special*] **QTWEIGHT**=*5000* causes jobs using the QoS *special* to have their queue time subcomponent weight increased by 5000.

Expansion Factor (XFACTOR) Subcomponent

The expansion factor subcomponent has an effect similar to the queue time factor but favors shorter jobs based on their requested wallclock run time. In its traditional form, the expansion factor (XFactor) metric is calculated as follows:

XFACTOR = 1 + <QUEUETIME> / <EXECUTIONTIME>

However, a couple of aspects of this calculation make its use more difficult. First, the length of time the job will actually run—<EXECUTIONTIME>—is not actually known until the job completes. All that is known is how much time the job requests. Secondly, as described in the Queue Time Subcomponent section, Moab does not necessarily use the raw time since job submission to determine <QUEUETIME> to prevent various scheduler abuses. Consequently, Moab uses the following modified equation:

XFACTOR = 1 + <EFFQUEUETIME> / <WALLCLOCKLIMIT>

In the equation Moab uses, <EFFQUEUETIME> is the effective queue time subject to the JOBPRIOACCRUALPOLICY parameter and <WALLCLOCKLIMIT> is the user—or system—specified job wallclock limit.

Using this equation, it can be seen that short running jobs will have an XFactor that will grow much faster over time than the xfactor associated with long running jobs. The following table demonstrates this favoring of short running jobs:

Job Queue Time	1 hour	2 hours	4 hours	8 hours	16 hours
XFactor for 1 hour job	1 + (1 / 1) = 2.00	1 + (2 / 1) = 3.00	1 + (4 / 1) = 5.00	1 + (8 / 1) = 9.00	1 + (16 / 1) = 17.0
XFactor for 4 hour job	1 + (1 / 4) = 1.25	1 + (2 / 4) = 1.50	1 + (4 / 4) = 2.00	1 + (8 / 4) = 3.00	1 + (16 / 4) = 5.0

Since XFactor is calculated as a ratio of two values, it is possible for this subcomponent to be almost arbitrarily large, potentially swamping the value of other priority subcomponents. This can be addressed either by using the subcomponent cap XFACTORCAP, or by using the XFMINWCLIMIT parameter. If the latter is used, the calculation for the XFactor subcomponent value becomes:

XFACTOR = 1 + <EFFQUEUETIME> / MAX(<XFMINWCLIMIT>,<WALLCLOCKLIMIT>)

Using the **XFMINWCLIMIT** parameter allows a site to prevent very short jobs from causing the XFactor subcomponent to grow inordinately.

Some sites consider XFactor to be a more fair scheduling performance metric than queue time. At these sites, job XFactor is given far more weight than job queue time when calculating job priority and job XFactor distribution consequently tends to be fairly level across a wide range of job durations. (That is, a flat XFactor distribution of 1.0 would result in a one-minute job being queued on average one minute, while a 24-hour job would be queued an average of 24 hours.)

Like queue time, the effective XFactor subcomponent weight is the sum of two weights, the **XFACTORWEIGHT** parameter and the QoS-specific XFWEIGHT setting. For example, the line **QOSCFG** *[special]* **XFWEIGHT**=*5000* causes jobs using the QoS *special* to increase their expansion factor subcomponent weight by 5000.

Bypass (BYPASS) Subcomponent

The bypass factor is based on the bypass count of a job where the bypass count is increased by one every time the job is bypassed by a lower priority job via backfill. Backfill starvation has never been reported, but if encountered, use the BYPASS subcomponent.

StartCount (STARTCOUNT) Subcomponent

Apply the startcount factor to sites with trouble starting or completing due to policies or failures. The primary causes of an idle job having a startcount greater than zero are resource manager level job start failure, administrator based requeue, or requeue based preemption.

Deadline (DEADLINE) Subcomponent

The deadline factor allows sites to take into consideration the proximity of a job to its DEADLINE. As a jobs moves closer to its deadline its priority increases linearly. This is an alternative to the strict deadline discussed in QOS SERVICE.

Soft Policy Violation (SPVIOLATION) Subcomponent

The soft policy violation factor allows sites to favor jobs which do not violate their associated soft resource limit policies.

User Priority (USERPRIO) Subcomponent

The user priority subcomponent allows sites to consider end-user specified job priority in making the overall job priority calculation. Under Moab, end-user specified priorities may only be negative and are bounded in the range 0 to -1024. See Manual Priority Usage and Enabling End-user Priorities for more information.

> User priorities can be positive, ranging from -1024 to 1023, if `ENABLEPOSUSERPRIORITY TRUE` is specified in `moab.cfg`.

Target Service (TARG) Component

The target factor component of priority takes into account job scheduling performance targets. Currently, this is limited to target expansion factor and target queue time. Unlike the expansion factor and queue time factors described earlier which increase gradually over time, the target factor component is designed to grow exponentially as the target metric is approached. This behavior causes the scheduler to do essentially all in its power to make certain the scheduling targets are met.

The priority calculation for the target factor is as follows:

```
Priority += TARGETWEIGHT* (
  TARGETQUEUETIMEWEIGHT  * QueueTimeComponent +
  TARGETXFACTORWEIGHT    * XFactorComponent)
```

The queue time and expansion factor target are specified on a per QoS basis using the **XFTARGET** and **QTTARGET** attributes with the QOSCFG parameter. The QueueTime and XFactor component calculations are designed to produce small values until the target value begins to approach, at which point these components grow very rapidly. If the target is missed, this component remains high and continues to grow, but it does not grow exponentially.

Usage (USAGE) Component

The Usage component applies to active jobs only. The priority calculation for the usage priority factor is as follows:

```
Priority += USAGEWEIGHT * (
  USAGECONSUMEDWEIGHT     * ProcSecondsConsumed +
  USAGEHUNGERWEIGHT       * ProcNeededToBalanceDynamicJob +
  USAGEREMAININGWEIGHT    * ProcSecRemaining +
```

USAGEEXECUTIONTIMEWEIGHT * SecondsSinceStart +
USAGEPERCENTWEIGHT * WalltimePercent)

Job Attribute (ATTR) Component

The Attribute component allows the incorporation of job attributes into a job's priority. The most common usage for this capability is to do one of the following:

- adjust priority based on a job's state (favor suspended jobs)
- adjust priority based on a job's requested node features (favor jobs that request attribute **pvfs**)
- adjust priority based on internal job attributes (disfavor **backfill** or **preemptee** jobs)
- adjust priority based on a job's requested licenses, network consumption, or generic resource requirements

To use job attribute based prioritization, the JOBPRIOF parameter must be specified to set corresponding attribute priorities. To favor jobs based on node feature requirements, the parameter NODETOJOBATTRMAP must be set to map node feature requests to job attributes.

The priority calculation for the attribute priority factor is as follows:

Priority += ATTRWEIGHT * (
ATTRATTRWEIGHT * <ATTRPRIORITY> +
ATTRSTATEWEIGHT * <STATEPRIORITY> +
ATTRGRESWEIGHT * <GRESPRIORITY>
JOBIDWEIGHT * <JOBID> +
JOBNAMEWEIGHT * <JOBNAME_INTEGER>)

Example 3-87:

```
ATTRWEIGHT      100
ATTRATTRWEIGHT    1
ATTRSTATEWEIGHT   1
ATTRGRESWEIGHT    5
# favor suspended jobs
# disfavor preemptible jobs
# favor jobs requesting 'matlab'

JOBPRIOF STATE[Running]=100  STATE[Suspended]=1000  ATTR[PREEMPTEE]=-200  ATTR[gpfs]
=30  GRES[matlab]=400
# map node features to job features

NODETOJOBATTRMAP  gpfs,pvfs
...
```

Related topics

- Node Allocation Priority
- Per Credential Priority Weight Offsets
- Managing Consumable Generic Resources

Fairshare Job Priority Example

Consider the following information associated with calculating the fairshare factor for job X.

Job X
User A
Group B
Account C
QOS D
Class E

User A
Fairshare Target: 50.0
Current Fairshare Usage: 45.0

Group B
Fairshare Target: [NONE]
Current Fairshare Usage: 65.0

Account C
Fairshare Target: 25.0
Current Fairshare Usage: 35.0

QOS D
Fairshare Target: 10.0+
Current Fairshare Usage: 25.0

Class E
Fairshare Target: [NONE]
Current Fairshare Usage: 20.0

Priority Weights:
FSWEIGHT 100
FSUSERWEIGHT 10
FSGROUPWEIGHT 20
FSACCOUNTWEIGHT 30
FSQOSWEIGHT 40
FSCLASSWEIGHT 0

In this example, the Fairshare component calculation would be as follows:

```
Priority += 100 * (
    10 * 5 +
    20 * 0 +
    30 * (-10) +
    40 * 0 +
     0 * 0)
```

User A is 5% below his target so fairshare increases the total fairshare factor accordingly. Group B has no target so group fairshare usage is ignored. Account C is above its 10% above its fairshare usage target so this component decreases the job's total fairshare factor. QOS D is 15% over its target but the '+' in the target specification indicates that this is a 'floor' target, only influencing priority when fairshare usage drops below the target value. Thus, the QOS D fairshare usage delta does not influence the fairshare factor.

Fairshare is a great mechanism for influencing job turnaround time via priority to favor a particular distribution of jobs. However, it is important to realize that fairshare can only favor a particular distribution of jobs, it cannot force it. If user X has a fairshare target of 50% of the machine but does not submit enough jobs, no amount of priority favoring will get user X's usage up to 50%.

See the Fairshare Overview for more information.

Common Priority Usage

- Credential Priority Factors
- Service Level Priority Factors
- Priority Factor Caps
- User Selectable Prioritization

Site administrators vary widely in their preferred manner of prioritizing jobs. Moab's scheduling hierarchy allows sites to meet job control needs without requiring adjustments to dozens of parameters. Some choose to use numerous subcomponents, others a few, and still others are content with the default FIFO behavior. Any subcomponent that is not of interest may be safely ignored.

Credential Priority Factors

To help clarify the use of priority weights, a brief example may help. Suppose a site wished to maintain the FIFO behavior but also incorporate some credential based prioritization to favor a special user. Particularly, the site would like the user *john* to receive a higher initial priority than all other users. Configuring this behavior requires two steps. First, the user credential subcomponent must be enabled and second, *john* must have his relative priority specified. Take a look at the sample `moab.cfg` file:

```
USERWEIGHT       1
USERCFG[john]    PRIORITY=300
```

The "USER" priority subcomponent was enabled by setting the USERWEIGHT parameter. In fact, the parameters used to specify the weights of all components and subcomponents follow this same "*WEIGHT" naming convention (as in RESWEIGHT and TARGETQUEUETIMEWEIGHT.

The second part of the example involves specifying the actual user priority for the user *john*. This is accomplished using the USERCFG parameter. Why was the priority 300 selected and not some other value? Is this value arbitrary? As in any priority system, actual priority values are meaningless, only relative values are important. In this case, we are required to balance user priorities with the default queue time based priorities. Since queuetime priority is measured in minutes queued, the user priority of 300 places a job by user *john* on par with a job submitted 5 minutes earlier by another user.

Is this what the site wants? Maybe, maybe not. At the onset, most sites are uncertain what they want in prioritization. Often, an estimate initiates prioritization and adjustments occur over time. Cluster resources evolve, the workload evolves, and even site policies evolve, resulting in changing priority needs over time. Anecdotal evidence indicates that most sites establish a relatively stable priority policy within a few iterations and make only occasional adjustments to priority weights from that point.

Service Level Priority Factors

In another example, suppose a site administrator wants to do the following:

- favor jobs in the low, medium, and high QoSs so they will run in QoS order
- balance job expansion factor
- use job queue time to prevent jobs from starving

Under such conditions, the sample `moab.cfg` file might appear as follows:

```
QOSWEIGHT              1
XFACTORWEIGHT          1
QUEUETIMEWEIGHT       10
TARGETQUEUETIMEWEIGHT  1
QOSCFG[low]       PRIORITY=1000
QOSCFG[medium]    PRIORITY=10000
QOSCFG[high]      PRIORITY=100000
QOSCFG[DEFAULT]   QTTARGET=4:00:00
```

This example is a bit more complicated but is more typical of the needs of many sites. The desired QoS weightings are established by enabling the QoS subfactor using the QOSWEIGHT parameter while the various QoS priorities are specified using QOSCFG. XFACTORWEIGHT is then set as this subcomponent tends to establish a balanced distribution of expansion factors across all jobs. Next, the queuetime component is used to gradually raise the priority of all jobs based on the length of time they have been queued. Note that in this case, QUEUETIMEWEIGHT was explicitly set to 10, overriding its default value of 1. Finally, the TARGETQUEUETIMEWEIGHT parameter is used in conjunction with the USERCFG line to specify a queue time target of 4 hours.

Priority Factor Caps

Assume now that the site administrator is content with this priority mix but has a problem with users submitting large numbers of very short jobs. Very short jobs would tend to have rapidly growing XFactor values and would consequently quickly jump to the head of the queue. In this case, a factor cap would be appropriate. Such caps allow a site to limit the contribution of a job's priority factor to be within a defined range. This prevents certain priority factors from swamping others. Caps can be applied to either priority components or subcomponents and are specified using the **<COMPONENTNAME>CAP** parameter (such as **QUEUETIMECAP**, **RESCAP**, and **SERVCAP**). Note that both component and subcomponent caps apply to the pre-weighted value, as in the following equation:

```
Priority =
      C1WEIGHT * MIN(C1CAP,SUM(
        S11WEIGHT * MIN(S11CAP,S11S) +
        S12WEIGHT * MIN(S12CAP,S12S) +
        ...)) +
      C2WEIGHT * MIN(C2CAP,SUM(
        S21WEIGHT * MIN(S21CAP,S21S) +
        S22WEIGHT * MIN(S22CAP,S22S) +
        ...)) +
      ...
```

Example 3-88: Priority cap

```
QOSWEIGHT               1
QOSCAP                  10000
XFACTORWEIGHT           1
XFACTORCAP              1000
QUEUETIMEWEIGHT         10
QUEUETIMECAP            1000
```

User Selectable Prioritization

Moab allows users to specify a job priority to jobs they own or manage. This priority may be set at job submission time or it may be dynamically modified (using setspri or mjobctl) after submitting the job. For fairness reasons, users may only apply a negative priority to their job and thus slide it further back in the queue. This enables users to allow their more important jobs to run before their less important ones without gaining unfair advantage over other users.

User priorities can be positive if `ENABLEPOSUSERPRIORITY TRUE` is specified in `moab.cfg`.

In order to set **ENABLEPOSUSERPRIORITY**, you must change the USERPRIOWEIGHT from its default value of 0. For example:

```
USERPRIOWEIGHT   100
```

```
> setspri -r 100 332411
successfully modified job priority
```

Specifying a user priority at job submission time is resource manager specific. See the associated resource manager documentation for more information.

User Selectable Priority w/QoS

Using the QoS facility, organizations can set up an environment in which users can more freely select the desired priority of a given job. Organizations may enable access to a number of QoSs each with its own charging rate, priority, and target service levels. Users can then assign job importance by selecting the appropriate QoS. If desired, this can allow a user to jump ahead of other users in the queue if they are willing to pay the associated costs.

Related topics

- User Selectable Priority

Prioritization Strategies

Each component or subcomponent may be used to accomplish different objectives. **WALLTIME** can be used to favor (or disfavor) jobs based on their duration. Likewise, **ACCOUNT** can be used to favor jobs associated with a particular project while **QUEUETIME** can be used to favor those jobs waiting the longest.

- Queue Time
- Expansion Factor
- Resource
- Fairshare
- Credential
- Target Metrics

Each priority factor group may contain one or more subfactors. For example, the Resource factor consists of Node, Processor, Memory, Swap, Disk, and PE components. From the table in Job Priority Factors section, it is apparent that the prioritization problem is fairly complex since every site needs to prioritize a bit differently. When calculating a priority, the various priority factors are summed and then bounded between 0 and MAX_PRIO_VAL, which is currently defined as 100000000 (one billion).

The mdiag -p command assists with visualizing the priority distribution resulting from the current job priority configuration. Also, the showstats -f command helps indicate the impact of the current priority settings on scheduler service distributions.

Manual Job Priority Adjustment

Batch administrator's regularly find a need to adjust the calculated priority of a job to meet current needs. Current needs often are broken into two categories:

1. The need to run an administrator test job as soon as possible.
2. The need to pacify a disserviced user.

You can use the setspri command to handle these issues in one of two ways; this command allows the specification of either a relative priority adjustment or the specification of an absolute priority. Using absolute priority specification, administrators can set a job priority guaranteed to be higher than any calculated value. Where Moab-calculated job priorities are in the range of 0 to 1 billion, system administrator assigned absolute priorities start at 1 billion and go up. Issuing the `setspri` *<PRIO>* *<JOBID>* command, for example, assigns a priority of 1 billion + <PRIO> to the job. Thus, `setspri 5 job.1294` sets the priority of "job.1294" to 1000000005.

For more information, see Common Priority Usage - End-user Adjustment.

Node Allocation Policies

While job prioritization allows a site to determine which job to run, node allocation policies allow a site to specify how available resources should be allocated to each job. The algorithm used is specified by the parameter NODEALLOCATIONPOLICY. There are multiple node allocation policies to choose from allowing selection based on reservation constraints, node configuration, resource usage, prefed other factors. You can specify these policies with a system-wide default value, on a per-partition basis, or on a per-job basis. Please note that **LASTAVAILABLE** is the default policy.

Available algorithms are described in detail in the following sections and include FIRSTAVAILABLE, LASTAVAILABLE, PRIORITY, CPULOAD, MINRESOURCE, CONTIGUOUS, MAXBALANCE, PLUGIN.

Node Allocation Overview

Node allocation is the process of selecting the best resources to allocate to a job from a list of available resources. Making this decision intelligently is important in an environment that possesses one or more of the following attributes:

- heterogeneous resources (resources which vary from node to node in terms of quantity or quality)
- shared nodes (nodes may be utilized by more than one job)
- reservations or service guarantees
- non-flat network (a network in which a perceptible performance degradation may potentially exist depending on workload placement)

Heterogeneous Resources

Moab analyzes job processing requirements and assigns resources to maximize hardware utility.

For example, suppose two nodes are available in a system, A and B. Node A has 768 MB of RAM and node B has 512 MB. The next two jobs in the queue are X and Y. Job X requests 256 MB and job Y requests 640

MB. Job X is next in the queue and can fit on either node, but Moab recognizes that job Y (640 MB) can only fit on node A (768 MB). Instead of putting job X on node A and blocking job Y, Moab can put job X on node B and job Y on node A.

Shared Nodes

Symmetric Multiprocessing (SMP)

When sharing SMP-based compute resources amongst tasks from more than one job, resource contention and fragmentation issues arise. In SMP environments, the general goal is to deliver maximum system utilization for a combination of compute-intensive and memory-intensive jobs while preventing overcommitment of resources.

By default, most current systems do not do a good job of logically partitioning the resources (such as CPU, memory, and network bandwidth) available on a given node. Consequently contention often arises between tasks of independent jobs on the node. This can result in a slowdown for all jobs involved, which can have significant ramifications if large-way parallel jobs are involved. Virtualization, CPU sets, and other techniques are maturing quickly as methods to provide logical partitioning within shared resources.

On large-way SMP systems (> 32 processors/node), job packing can result in intra-node fragmentation. For example, take two nodes, A and B, each with 64 processors. Assume they are currently loaded with various jobs and A has 24 and B has 12 processors free. Two jobs are submitted; job X requests 10 processors and job Y requests 20 processors. Job X can start on either node but starting it on node A prevents job Y from running. An algorithm to handle intra-node fragmentation is straightforward for a single resource case, but the algorithm becomes more involved when jobs request a combination of processors, memory, and local disk. These workload factors should be considered when selecting a site's node allocation policy as well as identifying appropriate policies for handling resource utilization limit violations.

Interactive Nodes

In many cases, sites are interested in allowing multiple users to simultaneously use one or more nodes for interactive purposes. Workload is commonly not compute intensive consisting of intermittent tasks including coding, compiling, and testing. Because these jobs are highly variant in terms of resource usage over time, sites are able to pack a larger number of these jobs onto the same node. Consequently, a common practice is to restrict job scheduling based on utilized, rather than dedicated resources.

Interactive Node Example

The example configuration files that follow show one method by which node sharing can be accomplished within a TORQUE + Moab environment. This example is based on a hypothetical cluster composed of 4 nodes each with 4 cores. For the compute nodes, job tasks are limited to actual cores preventing overcommitment of resources. For the interactive nodes, up to 32 job tasks are allowed, but the node also stops allowing additional tasks if either memory is fully utilized or if the CPU load exceeds 4.0. Thus, Moab continues packing the interactive nodes with jobs until carrying capacity is reached.

Example 3-89: /opt/moab/etc/moab.cfg

```
# constrain interactive jobs to interactive nodes
# constrain interactive jobs to 900 proc-seconds
CLASSCFG[interactive]  HOSTLIST=interactive01,interactive02
CLASSCFG[interactive]  MAX.CPUTIME=900
RESOURCELIMITPOLICY    CPUTIME:ALWAYS:CANCEL
# base interactive node allocation on load and jobs
NODEALLOCATIONPOLICY  PRIORITY
NODECFG[interactive01] PRIORITYF='-20*LOAD - JOBCOUNT'
NODECFG[interactive02] PRIORITYF='-20*LOAD - JOBCOUNT'
```

Example 3-90: /var/spool/torque/server_priv/nodes

```
interactive01 np=32
interactive02 np=32
compute01     np=4
compute02     np=4
```

Example 3-91: /var/spool/torque/mom_priv/config on "interactive01"

```
# interactive01
$max_load 4.0
```

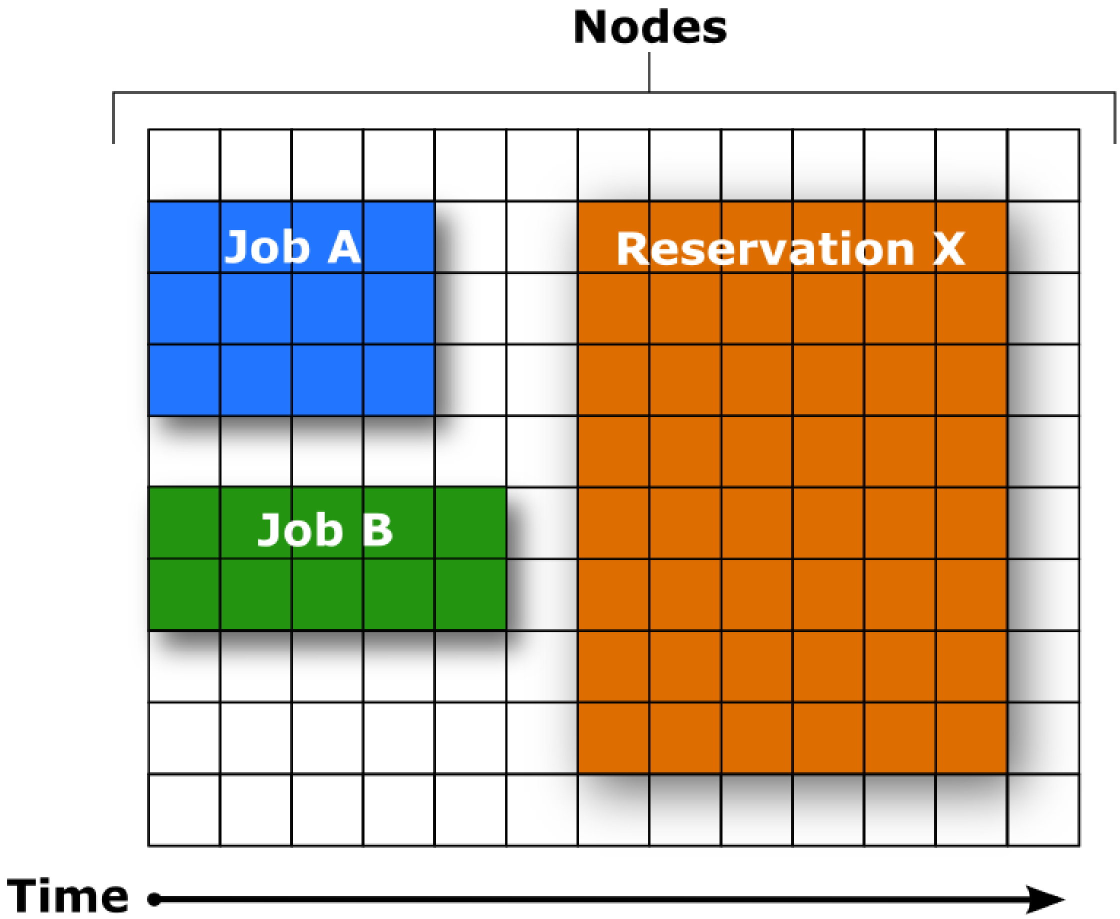
Example 3-92: /var/spool/torque/mom_priv/config on "interactive02"

```
# interactive02
$max_load 4.0
```

Reservations or Service Guarantees

A reservation-based system adds the time dimension into the node allocation decision. With reservations, node resources must be viewed in a type of two dimension node-time space. Allocating nodes to jobs fragments this node-time space and makes it more difficult to schedule jobs in the remaining, more constrained node-time slots. Allocation decisions should be made in such a way as to minimize this fragmentation and maximize the scheduler's ability to continue to start jobs in existing slots. The following figure shows that job A and job B are running. A reservation, X, is created some time in the future. Assume that job A is 2 hours long and job B is 3 hours long. Again, two new single-processor jobs are submitted, C and D; job C requires 3 hours of compute time while job D requires 5 hours. Either job will just fit in the free space located above job A or in the free space located below job B. If job C is placed above job A, job D, requiring 5 hours of time will be prevented from running by the presence of reservation X. However, if job C is placed below job B, job D can still start immediately above job A.

Image 3-2: Job A, Job B, and Reservation X scheduled on nodes



The preceding example demonstrates the importance of time based reservation information in making node allocation decisions, both at the time of starting jobs and at the time of creating reservations. The impact of time based issues grows significantly with the number of reservations in place on a given system. The LASTAVAILABLE algorithm works on this premise, locating resources that have the smallest space between the end of a job under consideration and the start of a future reservation.

Non-flat Network

On systems where network connections do not resemble a flat all-to-all topology, task placement may impact performance of communication intensive parallel jobs. If latencies and network bandwidth between any two nodes vary significantly, the node allocation algorithm should attempt to pack tasks of a given job as close to each other as possible to minimize impact of bandwidth and latency differences.

Node selection factors

While the node allocation policy determines which nodes a job will use, other factors narrow the options before the policy makes the final decision. The following process demonstrates how Moab executes its node allocation process and how other policies affect the decision:

1. Moab eliminates nodes that do not meet the hard resource requirements set by the job.
2. Moab gathers affinity information, first from workload proximity rules and then from reservation affinity rules (See **Affinity** on page 486 for more information.). Reservation affinity rules trump workload proximity rules.
3. Moab allocates nodes using the allocation policy.
    - If more than enough nodes with Required affinity exist, only they are passed down for the final sort by the node allocation policy.
    - If the number of nodes with Required affinity matches the number of nodes requested exactly, then the node allocation policy is skipped entirely and all of those nodes are assigned to the job.
    - If too few nodes have Required affinity, all of them are assigned to the job, then the node allocation policy is applied to the remaining eligible nodes (after Required, Moab will use Positive, then Neutral, then Negative.).

Resource-Based Algorithms

Moab contains a number of allocation algorithms that address some of the needs described earlier. You can also create allocation algorithms and interface them with the Moab scheduling system. Each of these policies has a name and descriptive alias. They can be configured using either one, but Moab will only report their names.

> If **ENABLEHIGHTHROUGHPUT** on page 929 is *TRUE*, you must set **NODEALLOCATIONPOLICY** on page 978 to *FIRSTAVAILABLE*.

The current suite of algorithms is described in what follows:

Allocation algorithm name	Alias	Description
CPULOAD	ProcessorLoad	Nodes are selected that have the maximum amount of available, unused CPU power (<#of CPU's> - <CPU load>). CPULOAD is a good algorithm for timesharing node systems and applies to jobs starting immediately. For the purpose of future reservations, the MINRESOURCE algorithm is used.
FIRSTAVAILABLE	InReportedOrder	Simple first come, first served algorithm where nodes are allocated in the order they are presented by the resource manager. This is a very simple, and very fast algorithm.

Allocation algorithm name	Alias	Description
LASTAVAILA-BLE	InReserveReportedOrder	Nodes are allocated in descending order that they are presented by the resource manager, or the reverse of FIRSTAVAILABLE.

Allocation algorithm name	Alias	Description
PRIORITY	CustomPriority	Allows a site to specify the priority of various static and dynamic aspects of compute nodes and allocate them with preference for higher priority nodes. It is highly flexible allowing node attribute and usage information to be combined with reservation affinity. Using node allocation priority, you can specify the following priority components: • *ADISK* - Local disk currently available to batch jobs in MB. • *AMEM* - Real memory currently available to batch jobs in MB. • *APROCS* - Processors currently available to batch jobs on node (configured procs - dedicated procs). • *ARCH[<ARCH>]* - Processor architecture. • *ASWAP* - Virtual memory currently available to batch jobs in MB. • *CDISK* - Total local disk allocated for use by batch jobs in MB. • *CMEM* - Total real memory on node in MB. • *COST* - Based on node **CHARGERATE**. • *CPROCS* - Total processors on node. • *CSWAP* - Total virtual memory configured on node in MB. • *FEATURE[<FNAME>]* - Boolean; specified feature is present on node. • *FREETIME* - FREETIME is calculated as the time during which there is no reservation on the machine. It uses either the job wallclock limit (if there is a job), or 2 months. The more free time a node has within either the job wallclock limit or 2 months, the higher this value will be. • *GMETRIC[<GMNAME>]* - Current value of specified generic metric on node. • *JOBCOUNT* - Number of jobs currently running on node. • *JOBFREETIME* - The number of seconds that the node is idle between now and when the job is scheduled to start. • *LOAD* - Current 1 minute load average. • *MTBF* - Mean time between failures (in seconds). • *NODEINDEX* - Node's nodeindex as specified by the resource manager. • *OS* - True if job compute requirements match node operating system. • *PARAPROCS* - Processors currently available to batch jobs within partition (configured procs - dedicated procs). • *POWER* - TRUE if node is ON. • *PREF* - Boolean; node meets job specific resource preferences. • *PRIORITY* - Administrator specified node priority. • *RANDOM* - Per iteration random value between 0 and 1. (Allows introduction of random allocation factor.) Regardless of coefficient, the contribution of this

Allocation algorithm name	Alias	Description
		Example 5: Pack tasks onto nodes with the most processors available and the lowest CPU temperature. `RMCFG[torque] TYPE=pbs` `RMCFG[temp]   TYPE=NATIVE` `CLUSTERQUERYURL=exec://$TOOLSDIR/hwmon.pl` `NODEALLOCATIONPOLICY PRIORITY` `NODECFG[DEFAULT] PRIORITYF='100*APROCS - GMETRIC[temp]'` `...`
MINRESOURCE	MinimumConfiguredResources	Prioritizes nodes according to the configured memory resources on each node. Those nodes with the fewest configured memory resources, that still meet the job's resource constraints, are selected.
CONTIGUOUS	Contiguous	Allocates nodes in contiguous (linear) blocks as required by the Compaq RMS system.
MAXBALANCE	ProcessorSpeedBalance	Attempts to allocate the most balanced set of nodes possible to a job. In most cases, but not all, the metric for balance of the nodes is node procspeed. Thus, if possible, nodes with identical procspeeds are allocated to the job. If identical procspeed nodes cannot be found, the algorithm allocates the set of nodes with the minimum node procspeed span or range.

User-Defined Algorithms

User-defined algorithms allow administrators to define their own algorithms based on factors such as their system's network topology. When node allocation is based on topology, jobs finish faster, administrators see better cluster productivity and users pay less for resources.

PLUGIN

This algorithm allows administrators to define their own node allocation policy and create a plug-in that allocates nodes based on factors such as a cluster's network topology. This has the following advantages:

- plug-ins keep the source code of the cluster's interconnect network for node allocation separate from Moab's source code (customers can implement plug-ins independent of Moab's release schedule)
- plug-ins can be independently created and tailored to specific hardware and network topology
- plug-ins can be modified without assistance from Adaptive Computing, Inc.

Specifying *Per Job* Resource Preferences

While the resource based node allocation algorithms can make a good guess at what compute resources would best satisfy a job, sites often possess a subset of jobs that benefit from more explicit resource allocation specification. For example one job may perform best on a particular subset of nodes due to direct access to a tape drive, another may be very memory intensive. Resource preferences are distinct from node requirements. While the former describes what a job needs to run at all, the latter describes what the job needs to run well. In general, a scheduler must satisfy a job's node requirement specification and then satisfy the job's resource preferences as well as possible.

Specifying Resource Preferences

A number of resource managers natively support the concept of resource preferences (such as Loadleveler). When using these systems, the language specific preferences keywords may be used. For systems that do not support resource preferences natively, Moab provides a resource manager extension keyword, "PREF," which you can use to specify desired resources. This extension allows specification of node features, memory, swap, and disk space conditions that define whether the node is considered preferred.

> Moab 5.2 (and earlier) only supports feature-based preferences.

Selecting Preferred Resources

Enforcing resource preferences is not completely straightforward. A site may have a number of potentially conflicting requirements that the scheduler is asked to simultaneously satisfy. For example, a scheduler may be asked to maximize the proximity of the allocated nodes at the same time it is supposed to satisfy resource preferences and minimize node overcommitment. To allow site specific weighting of these varying requirements, Moab allows resource preferences to be enabled through the **PRIORITY** node allocation algorithm. For example, to use resource preferences together with node load, the following configuration might be used:

```
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT]     PRIORITYF='5 * PREF - LOAD'
...
```

To request specific resource preferences, a user could then submit a job indicating those preferences. In the case of a PBS job, the following can be used:

```
> qsub -l nodes=4,walltime=1:00:00,pref=feature:fast
```

Related topics

- Generic Metrics
- Per Job Node Allocation Policy Specification via Resource Manager Extensions

Node Access Policies

Moab allocates resources to jobs on the basis of a job task—an atomic collection of resources that must be co-located on a single compute node. A given job may request 20 tasks where each task is defined as one processor and 128 MB of RAM. Compute nodes with multiple processors often possess enough resources to support more than one task simultaneously. When it is possible for more than one task to run on a node, node access policies determine which tasks may share the compute node's resources.

Moab supports a distinct number of node access policies that are listed in the following table:

Policy	Description
SHARED	Tasks from any combination of jobs may use available resources.
SHAREDONLY	Only jobs requesting shared node access may use available resources.
SINGLEACCOUNT	Tasks from any jobs owned by the same account may use available resources.
SINGLEGROUP	Tasks from any jobs owned by the same group may use available resources.
SINGLEJOB	Only tasks from a single job may use the node's resources.
SINGLETASK	Only a single task from a single job may run on the node.
SINGLEUSER	Tasks from any jobs owned by the same user may use available resources.
UNIQUEUSER	Any number of tasks from a single job may allocate resources from a node but only if the user has no other jobs running on that node. **UNIQUEUSER** limits the number of jobs a single user can run on a node, allowing other users to run jobs with the remaining resources. This policy is useful in environments where job epilog/prologs scripts are used to clean up processes based on userid.

Configuring Node Access Policies

The global node access polices may be specified via the parameter NODEACCESSPOLICY. This global default may be overridden on a per node basis with the ACCESS attribute of the NODECFG parameter or on a per job basis using the resource manager extension NACCESSPOLICY. Finally, a per queue node access policy may also be specified by setting either the NODEACCESSPOLICY or FORCENODEACCESSPOLICY attributes of the CLASSCFG parameter. **FORCENODEACCESSPOLICY** overrides any per job specification in all cases, whereas **NODEACCESSPOLICY** is overridden by per job specification.

By default, nodes are accessible using the setting of the system wide **NODEACCESSPOLICY** parameter unless a specific **ACCESS** policy is specified on a per node basis using the **NODECFG** parameter. Jobs may override this policy and subsequent jobs are bound to conform to the access policies of all jobs currently running on a given node. For example, if the **NODEACCESSPOLICY** parameter is set to **SHARED**, a new job may be launched on an idle node with a job specific access policy of **SINGLEUSER**. While this job runs, the effective node access policy changes to **SINGLEUSER** and subsequent job tasks may only be launched on this node provided they are submitted by the same user. When all single user jobs have completed on that node, the effective node access policy reverts back to **SHARED** and the node can again be used in **SHARED** mode.

For example, to set a global policy of **SINGLETASK** on all nodes except nodes 13 and 14, use the following:

```
# by default, enforce dedicated node access on all nodes
NODEACCESSPOLICY  SINGLETASK
# allow nodes 13 and 14 to be shared
NODECFG[node13]   ACCESS=SHARED
NODECFG[node14]   ACCESS=SHARED
```

Related topics

- Per job naccesspolicy specification via Resource Manager Extensions
- JOBNODEMATCHPOLICY parameter
- NODEAVAILABILITY parameter

Node Availability Policies

- Node Resource Availability Policies
- Node Categorization
- Node Failure/Performance Based Notification
- Node Failure/Performance Based Triggers
- Handling Transient Node Failures
- **Allocated Resource Failure Policy for Jobs** on page 413

Moab enables several features relating to node availability. These include policies that determine how per node resource availability should be reported, how node failures are detected, and what should be done in the event of a node failure.

Node Resource Availability Policies

Moab allows a job to be launched on a given compute node as long as the node is not full or busy. The NODEAVAILABILITYPOLICY parameter allows a site to determine what criteria constitute a node being busy. The legal settings are listed in the following table:

Availability Policy	Description
DEDICATED	The node is considered busy if dedicated resources equal or exceed configured resources.

Availability Policy	Description
UTILIZED	The node is considered busy if utilized resources equal or exceed configured resources.
COMBINED	The node is considered busy if either dedicated or utilized resources equal or exceed configured resources.

The default setting for all nodes is **COMBINED**, indicating that a node can accept workload so long as the jobs that the node was allocated to do not request or use more resources than the node has available. In a load balancing environment, this may not be the desired behavior. Setting the **NODEAVAILABILITYPOLICY** parameter to **UTILIZED** allows jobs to be packed onto a node even if the aggregate resources requested exceed the resources configured. For example, assume a scenario with a 4-processor compute node and 8 jobs requesting 1 processor each. If the resource availability policy was set to **COMBINED**, this node would only allow 4 jobs to start on this node even if the jobs induced a load of less than 1.0 each. With the resource availability policy set to **UTILIZED**, the scheduler continues allowing jobs to start on the node until the node's load average exceeds a per processor load value of 1.0 (in this case, a total load of 4.0). To prevent a node from being over populated within a single scheduling iteration, Moab artificially raises the node's load for one scheduling iteration when starting a new job. On subsequent iterations, the actual measured node load information is used.

Per Resource Availability Policies

By default, the **NODEAVAILABILITYPOLICY** sets a global per node resource availability policy. This policy applies to all resource types on each node such as processors, memory, swap, and local disk. However, the syntax of this parameter is as follows:

`<POLICY>[:<RESOURCETYPE>] ...`

This syntax allows per resource availability specification. For example, consider the following:

```
NODEAVAILABILITYPOLICY  DEDICATED:PROC COMBINED:MEM COMBINED:DISK
...
```

This configuration causes Moab to only consider the quantity of processing resources actually dedicated to active jobs running on each node and ignore utilized processor information (such as CPU load). For memory and disk, both utilized resource information and dedicated resource information should be combined to determine what resources are actually available for new jobs.

Node Categorization

Moab allows organizations to detect and use far richer information regarding node status than the standard batch "idle," "busy," "down states" commonly found. Using node categorization, organizations can record, track, and report on per node and cluster level status including the following categories:

Category	Description
Active	Node is healthy and currently executing batch workload.
BatchFailure	Node is unavailable due to a failure in the underlying batch system (such as a resource manager server or resource manager node daemon).
Benchmark	Node is reserved for benchmarking.
EmergencyMaintenance	Node is reserved for unscheduled system maintenance.
GridReservation	Node is reserved for grid use.
HardwareFailure	Node is unavailable due to a failure in one or more aspects of its hardware configuration (such as a power failure, excessive temperature, memory, processor, or swap failure).
HardwareMaintenance	Node is reserved for scheduled system maintenance.
Idle	Node is healthy and is currently not executing batch workload.
JobReservation	Node is reserved for job use.
NetworkFailure	Node is unavailable due to a failure in its network adapter or in the switch.
Other	Node is in an uncategorized state.
OtherFailure	Node is unavailable due to a general failure.
PersonalReservation	Node is reserved for dedicated use by a personal reservation.
Site[1-8]	Site specified usage categorization.
SoftwareFailure	Node is unavailable due to a failure in a local software service (such as automounter, security or information service such as NIS, local databases, or other required software services).
SoftwareMaintenance	Node is reserved for software maintenance.
StandingReservation	Node is reserved by a standing reservation.

Category	Description
StorageFailure	Node is unavailable due to a failure in the cluster storage system or local storage infrastructure (such as failures in Lustre, GPFS, PVFS, or SAN).
UserReservation	Node is reserved for dedicated use by a particular user or group and may or may not be actively executing jobs.

Node categories can be explicitly assigned by cluster administrators using the mrsvctl -c command to create a reservation and associate a category with that node for a specified timeframe. Further, outside of this explicit specification, Moab automatically mines all configured interfaces to learn about its environment and the health of the resources it is managing. Consequently, Moab can identify many hardware failures, software failures, and batch failures without any additional configuration. However, it is often desirable to make additional information available to Moab to allow it to integrate this information into reports; automatically notify managers, users, and administrators; adjust internal policies to steer workload around failures; and launch various custom triggers to rectify or mitigate the problem.

You can specify the FORCERSVSUBTYPE parameter to require all administrative reservations be associated with a node category at reservation creation time. For example:

```
NODECFG[DEFAULT] ENABLEPROFILING=TRUE
FORCERSVSUBTYPE  TRUE
```

Node health and performance information from external systems can be imported into Moab using the native resource manager interface. This is commonly done using generic metrics or consumable generic resources for performance and node categories or node variables for status information. Combined with arbitrary node messaging information, Moab can combine detailed information from remote services and report this to other external services.

Use the NODECATCREDLIST parameter to generate extended node category based statistics.

Node Failure/Performance Based Notification

Moab can be configured to cause node failures and node performance levels that cross specified thresholds to trigger notification events. This is accomplished using the GEVENTCFG parameter as described in the Generic Event Overview section. For example, the following configuration can be used to trigger an email to administrators each time a node is marked down.

```
GEVENTCFG[nodedown] ACTION=notify REARM=00:20:00
...
```

Node Failure/Performance Based Triggers

Moab supports per node triggers that can be configured to fire when specific events are fired or specific thresholds are met. These triggers can be used to modify internal policies or take external actions. A few examples follow:

- decrease node allocation priority if node throughput drops below threshold X
- launch local diagnostic/recovery script if parallel file system mounts become stale
- reset high performance network adapters if high speed network connectivity fails
- create general system reservation on node if processor or memory failure occurs

As mentioned, Moab triggers can be used to initiate almost any action, from sending mail to updating a database, to publishing data for an SNMP trap, to driving a web service.

Handling Transient Node Failures

Since Moab actively schedules both current and future actions of the cluster, it is often important for it to have a reasonable estimate of when failed nodes will be again available for use. This knowledge is particularly useful for proper scheduling of new jobs and management of resources in regard to backfill. With backfill, Moab determines which resources are available for priority jobs and when the highest priority idle jobs can run. If a node experiences a failure, Moab should have a concept of when this node will be restored.

When Moab analyzes down nodes for allocation, one of two issues may occur with the highest priority jobs. If Moab believes that down nodes will not be recovered for an extended period of time, a transient node failure within a reservation for a priority job may cause the reservation to slide far into the future allowing other lower priority jobs to allocate and launch on nodes previously reserved for it. Moments later, when the transient node failures are resolved, Moab may be unable to restore the early reservation start time as other jobs may already have been launched on previously available nodes.

In the reverse scenario, if Moab recognizes a likelihood that down nodes will be restored too quickly, it may make reservations for top priority jobs that allocate those nodes. Over time, Moab slides those reservations further into the future as it determines that the reserved nodes are not being recovered. While this does not delay the start of the top priority jobs, these unfulfilled reservations can end up blocking other jobs that should have properly been backfilled and executed.

Creating Automatic Reservations

If a node experiences occasional transient failures (often not associated with a node state of down), Moab can automatically create a temporary reservation over the node to allow the transient failure time to clear and prevent Moab from attempting to re-use the node while the failure is active. This reservation behavior is controlled using the NODEFAILURERESERVETIME parameter as in the following example:

```
# reserve nodes for 1 minute if transient failures are detected
NODEFAILURERESERVETIME  00:01:00
```

Blocking Out Down Nodes

If one or more resource managers identify failures and mark nodes as down, Moab can be configured to associate a default *unavailability* time with this failure and the node state *down*. This is accomplished using the NODEDOWNSTATEDELAYTIME parameter. This delay time floats and is measured as a fixed time into the future from the time " NOW"; it is not associated with the time the node was originally marked down. For example, if the delay time was set to 10 minutes, and a node was marked down 20 minutes ago, Moab would still consider the node unavailable until 10 minutes into the future.

While it is difficult to select a good default value that works for all clusters, the following is a general rule of thumb:

- Increase **NODEDOWNSTATEDELAYTIME** if jobs are getting blocked due to priority reservations sliding as down nodes are not recovered.
- Decrease **NODEDOWNSTATEDELAYTIME** if high priority job reservations are getting regularly delayed due to transient node failures.

```
# assume down nodes will not be recovered for one hour
NODEDOWNSTATEDELAYTIME  01:00:00
```

Allocated Resource Failure Policy for Jobs

If a failure occurs within a collection of nodes allocated to a job, Moab can automatically re-allocate replacement resources. This can be configured with JOBACTIONONNODEFAILURE.

How an active job behaves when one or more of its allocated resources fail depends on the allocated resource failure policy. Depending on the type of job, type of resources, and type of middleware infrastructure, a site may choose to have different responses based on the job, the resource, and the type of failure.

Failure Responses

By default, Moab cancels a job when an allocated resource failure is detected. However, you can specify the following actions:

Option	Policy action
CANCEL	Cancels the job
FAIL	Terminates the job as a failed job
HOLD	Places a hold on the job. This option is only applicable if you are using checkpointing
IGNORE	Ignores the failed node, allowing the job to proceed
NOTIFY	Notifies the administrator and user of failure but takes no further action
REQUEUE	Requeues job and allows it to run when alternate resources become available

Policy Precedence

For a given job, the applied policy can be set at various levels with policy precedence applied in the job, class/queue, partition, and then system level. The following table indicates the available methods for setting this policy:

Object	Parameter	Example
Job	RESFAILPOLICY resource manager extension	`> qsub -l resfailpolicy=requeue`
Class/Queue	RESFAILPOLICY attribute of CLASSCFG parameter	`CLASSCFG[batch] RESFAILPOLICY=CANCEL`
Partition	JOBACTIONONNODE FAILURE attribute of PARCFG parameter	`PARCFG[web3] JOBACTIONONNODEFAILURE=NOTIFY`
System	NODEALLOCRESFAILURE POLICY parameter	`NODEALLOCRESFAILUREPOLICY=MIGRATE`

Failure Definition

Any allocated node going down constitutes a failure. However, for certain types of workload, responses to failures may be different depending on whether it is the master task (task 0) or a slave task that fails. To indicate that the associated policy should only take effect if the master task fails, the allocated resource failure policy should be specified with a trailing asterisk (*), as in the following example:

```
CLASSCFG[virtual_services] RESFAILPOLICY=requeue*
```

TORQUE Failure Details

When a node fails becoming unresponsive, the resource manager central daemon identifies this failure within a configurable time frame (default: 60 seconds). Detection of this failure triggers an event that causes Moab to immediately respond. Based on the specified policy, Moab notifies administrators, holds the job, requeues the job, allocates replacement resources to the job, or cancels the job. If the job is canceled or requeued, Moab sends the request to TORQUE, which immediately frees all non-failed resources making them available for use by other jobs. Once the failed node is recovered, it contacts the resource manager central daemon, determines that the associated job has been canceled/requeued, cleans up, and makes itself available for new workload.

Related topics

- Node State Overview
- JOBACTIONONNODEFAILURE parameter
- NODEFAILURERESERVETIME parameter
- NODEDOWNSTATEDELAYTIME parameter (down nodes will be marked unavailable for the specified duration)
- NODEDRAINSTATEDELAYTIME parameter (offline nodes will be marked unavailable for the specified duration)
- NODEBUSYSTATEDELAYTIME parameter (nodes with unexpected background load will be marked unavailable for the specified duration)

- NODEALLOCRESFAILUREPOLICY parameter (action to take if executing jobs have one or more allocated nodes fail)

Task Distribution Policies

Under Moab, task distribution policies are specified at a global scheduler level, a global resource manager level, or at a per job level. In addition, you can set up some aspects of task distribution as defaults on a per class basis.

Related topics

- Node Set Overview
- Node Allocation Overview

Managing Fairness - Throttling Policies, Fairshare, and Allocation Management

Fairness Overview

The concept of cluster fairness varies widely from person to person and site to site. While some interpret it as giving all users equal access to compute resources, more complicated concepts incorporating historical resource usage, political issues, and job value are equally valid. While no scheduler can address all possible definitions of fair, Moab provides one of the industry's most comprehensive and flexible set of tools allowing most sites the ability to address their many and varied fairness management needs.

Under Moab, most fairness policies are addressed by a combination of the facilities described in the following table:

Job Prioritization	
Description:	Specifies what is most important to the scheduler. Using service based priority factors allows a site to balance job turnaround time, expansion factor, or other scheduling performance metrics.
Example:	`SERVICEWEIGHT    1` `QUEUETIMEWEIGHT 10` *Causes jobs to increase in priority by 10 points for every minute they remain in the queue.*

Usage Limits (Throttling Policies)

Description:	Specifies limits on exactly what resources can be used at any given instant.

Example:

```
USERCFG[john]      MAXJOB=3
GROUPCFG[DEFAULT]  MAXPROC=64
GROUPCFG[staff]    MAXPROC=128
```

Allows john to only run 3 jobs at a time. Allows the group staff to use up to 128 total processors and all other groups to use up to 64 processors.

Fairshare

Description:	Specifies usage targets to limit resource access or adjust priority based on historical cluster and grid level resource usage.

Example:

```
USERCFG[steve]  FSTARGET=25.0+
FSWEIGHT        1
FSUSERWEIGHT    10
```

Enables priority based fairshare and specifies a fairshare target for user steve such that his jobs are favored in an attempt to keep his jobs using at least 25.0% of delivered compute cycles.

Allocation Management

Description:	Specifies long term, credential-based resource usage limits.

Example:

```
AMCFG[mam] TYPE=MAM HOST=server.sys.net
```

Enables the Moab Accounting Manager allocation management interface. Within the allocation manager, project or account based allocations may be configured. These allocations may, for example, do such things as allow project X to use up to 100,000 processor-hours per quarter, provide various QoS sensitive charge rates, and share allocation access.

Quality of Service

Description:	Specifies additional resource and service access for particular users, groups, and accounts. QoS facilities can provide special priorities, policy exemptions, reservation access, and other benefits (as well as special charge rates).

Quality of Service	
Example:	`QOSCFG[orion] PRIORITY=1000 XFTARGET=1.2` `QOSCFG[orion] QFLAGS=PREEMPTOR,IGNSYSTEM,RESERVEALWAYS` *Enables jobs requesting the orion QoS a priority increase, an expansion factor target to improve response time, the ability to preempt other jobs, an exemption from system level job size policies, and the ability to always reserve needed resources if it cannot start immediately.*

Standing Reservations	
Description:	Reserves blocks of resources within the cluster for specific, periodic time frames under the constraints of a flexible access control list.
Example:	`SRCFG[jupiter] HOSTLIST=node01[1-4]` `SRCFG[jupiter] STARTTIME=9:00:00 ENDTIME=17:00:00` `SRCFG[jupiter] USERLIST=john,steve ACCOUNTLIST=jupiter` *Reserve nodes* `node011` *through* `node014` *from 9:00 AM until 5:00 PM for use by jobs from user john or steve or from the project jupiter.*

Class/Queue Constraints	
Description:	Associates users, resources, priorities, and limits with cluster classes or cluster queues that can be assigned to or selected by end-users.
Example:	`CLASSCFG[long] MIN.WCLIMIT=24:00:00` `SRCFG[jupiter] PRIORITY=10000` `SRCFG[jupiter] HOSTLIST=acn[1-4][0-9]` *Assigns long jobs a high priority but only allow them to run on certain nodes.*

Selecting the Correct Policy Approach

Moab supports a rich set of policy controls in some cases allowing a particular policy to be enforced in more than one way. For example, cycle distribution can be controlled using usage limits, fairshare, or even queue definitions. Selecting the most correct policy depends on site objectives and needs; consider the following when making such a decision:

- Minimal end-user training
    - Does the solution use an approach familiar to or easily learned by existing users?

- End-user transparency
    - Can the configuration be enabled or disabled without impacting user behavior or job submission?
- Impact on system utilization and system responsiveness
- Solution complexity
    - Is the impact of the configuration readily intuitive, and is it easy to identify possible side effects?
- Solution extensibility and flexibility
    - Will the proposed approach allow the solution to be easily tuned and extended as cluster needs evolve?

Related topics

- Job Prioritization
- Usage Limits (Throttling Policies)
- Fairshare
- Allocation Management
- Quality of Service
- Standing Reservations
- Class/Queue Constraints

Usage Limits/Throttling Policies

A number of Moab policies allow an administrator to control job flow through the system. These throttling policies work as filters allowing or disallowing a job to be considered for scheduling by specifying limits regarding system usage for any given moment. These policies may be specified as global or specific constraints specified on a per user, group, account, QoS, or class basis.

Fairness via Throttling Policies

Moab allows significant flexibility with usage limits, or throttling policies. At a high level, Moab allows resource usage limits to be specified in three primary workload categories: (1) active, (2) idle, and (3) system job limits.

Basic Fairness Policies

Workload category	Description
Active job limits	Constrain the total cumulative resources available to active jobs at a given time.
Idle job limits	Constrain the total cumulative resources available to idle jobs at a given time.
System job limits	Constrain the maximum resource requirements of any single job.

These limits can be applied to any job credential (user, group, account, QoS, and class), or on a system-wide basis. Using the keyword *DEFAULT*, a site may also specify the default setting for the desired user, group, account, QoS, and class. Additionally, you may configure QoS to allow limit overrides to any particular policy.

To run, a job must meet all policy limits. Limits are applied using the ***CFG** set of parameters, particularly USERCFG, GROUPCFG, ACCOUNTCFG, QOSCFG, CLASSCFG, and SYSCFG. Limits are specified by associating the desired limit to the individual or default object. The usage limits currently supported are listed in the following table.

MAXARRAYJOB	
Units	Number of simultaneous active array job sub-jobs.
Description	Limits the number of simultaneously active (starting or running) array sub-jobs a credential can have.
Example	`USERCFG[gertrude] MAXARRAYJOB=10` *Gertrude can have a maximum of 10 active job array sub-jobs.*

MAXGRES	
Units	# of concurrent uses of a generic resource
Description	Limits the concurrent usage of a generic resource to a specific quantity or quantity range.

MAXGRES	
Example	`USERCFG[joe] MAXGRES[matlab]=2` `USERCFG[jim] MAXGRES[matlab]=2,4`

MAXJOB	
Units	# of jobs
Description	Limits the number of jobs a credential may have active (starting or running) at any given time. Moab places a hold on all new jobs submitted by that credential once it has reached its maximum number of allowable jobs. `MAXJOB=0` is not supported. You can, however, achieve similar results by using the HOLD attribute of the USERCFG parameter: `USERCFG[john] HOLD=yes`
Example	`USERCFG[DEFAULT] MAXJOB=8` `GROUPCFG[staff]  MAXJOB=2,4`

MAXMEM	
Units	total memory in MB
Description	Limits the total amount of dedicated memory (in MB) that can be allocated by a credential's active jobs at any given time.
Example	`ACCOUNTCFG[jasper] MAXMEM=2048`

MAXNODE	
Units	# of nodes

MAXNODE	
Description	Limits the total number of compute nodes that can be in use by active jobs at any given time. On some systems (including TORQUE/PBS), nodes have been softly defined rather than strictly defined; that is, a job may request 2 nodes but TORQUE will translate this request into 1 node with 2 processors. This can prevent Moab from enforcing a **MAXNODE** policy correctly for a single job. Correct behavior can be achieved using **MAXPROC**.
Example	`CLASSCFG[batch] MAXNODE=64`

MAXPE	
Units	# of processor equivalents
Description	Limits the total number of dedicated processor-equivalents that can be allocated by active jobs at any given time.
Example	`QOSCFG[base] MAXPE=128`

MAXPROC	
Units	# of processors
Description	Limits the total number of dedicated processors that can be allocated by active jobs at any given time per credential. To set **MAXPROC** per job, use msub -W.
Example	`CLASSCFG[debug] MAXPROC=32`

MAXPS	
Units	<# of processors> * <walltime>
Description	Limits the number of outstanding processor-seconds a credential may have allocated at any given time. For example, if a user has a 4-processor job that will complete in 1 hour and a 2-processor job that will complete in 6 hours, they have 4 * 1 * 3600 + 2 * 6 * 3600 = 16 * 3600 outstanding processor-seconds. The outstanding processor-second usage of each credential is updated each scheduling iteration, decreasing as jobs approach their completion time.

MAXPS	
Example	`USERCFG[DEFAULT] MAXPS=720000`

MAXSUBMITJOBS	
Units	# of jobs
Description	Limits the number of jobs a credential may submit and have in the system at once. Moab will reject any job submitted beyond this limit. If you use a TORQUE resource manager, you should also set `max_user_queuable` in case the user submits jobs via qsub instead of msub. See "**Queue Attributes** on page 2277" in the *TORQUE Administrator Guide* for more information.
Example	`USERCFG[DEFAULT] MAXSUBMITJOBS=5`

MAXWC	
Units	job duration [[[DD:]HH:]MM:]SS
Description	Limits the cumulative remaining walltime a credential may have associated with active jobs. It behaves identically to the **MAXPS** on page 421 limit (listed earlier) only lacking the processor weighting. Like MAXPS, the cumulative remaining walltime of each credential is also updated each scheduling iteration. MAXWC does not limit the maximum wallclock limit per job. For this capability, use **MAX.WCLIMIT** on page 145.
Example	`USERCFG[ops] MAXWC=72:00:00`

The following example demonstrates a simple limit specification:

```
USERCFG[DEFAULT]  MAXJOB=4
USERCFG[john]     MAXJOB=8
```

This example allows user john to run up to 8 jobs while all other users may only run up to 4.

Simultaneous limits of different types may be applied per credential and multiple types of credentials may have limits specified. The next example demonstrates this mixing of limits and is a bit more complicated.

```
USERCFG[steve]    MAXJOB=2 MAXNODE=30
GROUPCFG[staff]   MAXJOB=5
CLASSCFG[DEFAULT] MAXNODE=16
CLASSCFG[batch]   MAXNODE=32
```

This configuration may potentially apply multiple limits to a single job. As discussed previously, a job may only run if it satisfies all applicable limits. Thus, in this example, the scheduler will be constrained to allow at most *2* simultaneous user *steve* jobs with an aggregate node consumption of no more than *30* nodes. However, if the job is submitted to a class other than *batch*, it may be limited further. Here, only *16* total nodes may be used simultaneously by jobs running in any given class with the exception of the class *batch*. If *steve* submitted a job to run in the class *interactive*, for example, and there were jobs already running in this class using a total of 14 nodes, his job would be blocked unless it requested 2 or fewer nodes by the default limit of *16* nodes per class.

Multi-Dimension Fairness Policies and Per Credential Overrides

Multi-dimensional fairness policies allow a site to specify policies based on combinations of job credentials. A common example might be setting a maximum number of jobs allowed per queue per user or a total number of processors per group per QoS. As with basic fairness policies, multi-dimension policies are specified using the ***CFG** parameters or through the identity manager interface. Moab supports the most commonly used multi-dimensional fairness policies (listed in the table below) using the following format:

***CFG**[*X*] **<LIMITTYPE>[<CRED>]**=*<LIMITVALUE>*

***CFG** is one of **USERCFG**, **GROUPCFG**, **ACCOUNTCFG**, **QOSCFG**, or **CLASSCFG**, the **<LIMITTYPE>** policy is one of the policies listed in the table in section 6.2.1.1, and **<CRED>** is of the format *<CREDTYPE>[:<VALUE>]* with *CREDTYPE* being one of *USER*, *GROUP*, *ACCT*, *QoS*, or *CLASS*. The optional *<VALUE>* setting can be used to specify that the policy only applies to a specific credential value. For example, the following configuration sets limits on the class *fast*, controlling the maximum number of jobs any group can have active at any given time and the number of processors in use at any given time for user *steve*.

```
CLASSCFG[fast] MAXJOB[GROUP]=12
CLASSCFG[fast] MAXPROC[USER:steve]=50
CLASSCFG[fast] MAXIJOB[USER]=10
```

The following example configuration may clarify further:

```
# allow class batch to run up the 3 simultaneous jobs
# allow any user to use up to 8 total nodes within class
CLASSCFG[batch] MAXJOB=3 MAXNODE[USER]=8
# allow users steve and bob to use up to 3 and 4 total processors respectively within
class
CLASSCFG[fast] MAXPROC[USER:steve]=3 MAXPROC[USER:bob]=4
```

Multi-dimensional policies cannot be applied on *DEFAULT* credentials.

The table below lists the currently implemented, multi-dimensional usage limit permutations. The "slmt" stands for "Soft Limit" and "hlmt" stands for "Hard Limit."

Multi-dimension usage limit permutations	
ACCOUNTCFG[name]	MAXIJOB[QOS]=hlmt MAXIJOB[QOS:qosname]=hlmt
	MAXIPROC[QOS]=hlmt MAXIPROC[QOS:qosname]=hlmt
	MAXJOB[QOS]=slmt,hlmt MAXJOB[QOS:qosname]=slmt,hlmt
	MAXJOB[USER]=slmt,hlmt MAXJOB[USER:username]=slmt,hlmt
	MAXMEM[USER]=slmt,hlmt MAXMEM[USER:username]=slmt,hlmt
	MAXNODE[USER]=slmt,hlmt MAXNODE[USER:username]=slmt,hlmt
	MAXPE[QOS]=slmt,hlmt MAXPE[QOS:qosname]=slmt,hlmt
	MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=slmt,hlmt
	MAXPROC[QOS]=slmt,hlmt MAXPROC[QOS:qosname]=slmt,hlmt
	MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=slmt,hlmt
	MAXPS[QOS]=slmt,hlmt MAXPS[QOS:qosname]=slmt,hlmt
	MAXPS[USER]=slmt,hlmt MAXPS[USER:username]=slmt,hlmt
	MAXWC[USER]=slmt,hlmt MAXWC[USER:username]=slmt,hlmt

Multi-dimension usage limit permutations	
CLASSCFG[name]	MAXJOB[GROUP]=slmt,hlmt MAXJOB[GROUP:groupname]=slmt,hlmt
	MAXJOB[QOS:qosname]=hlmt
	MAXJOB[USER]=slmt,hlmt MAXJOB[USER:username]=slmt,hlmt
	MAXMEM[GROUP]=slmt,hlmt MAXMEM[GROUP]=slmt,hlmt
	MAXMEM[GROUP]=slmt,hlmt MAXMEM[GROUP:groupname]=slmt,hlmt
	MAXMEM[QOS:qosname]=hlmt
	MAXMEM[USER]=slmt,hlmt MAXMEM[USER:username]=slmt,hlmt
	MAXNODE[GROUP]=slmt,hlmt MAXNODE[GROUP:groupname]=slmt,hlmt
	MAXNODE[QOS:qosname]=hlmt
	MAXNODE[USER]=slmt,hlmt MAXNODE[USER:username]=slmt,hlmt
	MAXPE[GROUP]=slmt,hlmt MAXPE[GROUP:groupname]=slmt,hlmt
	MAXPE[QOS:qosname]=hlmt
	MAXPE[USER]=slmt,hlmt MAXPE[USER:username]=slmt,hlmt
	MAXPROC[GROUP]=slmt,hlmt MAXPROC[GROUP:groupname]=slmt,hlmt

Multi-dimension usage limit permutations	
	MAXPROC[QOS:qosname]=hlmt
	MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=slmt,hlmt
	MAXPS[GROUP]=slmt,hlmt MAXPS[GROUP:groupname]=slmt,hlmt
	MAXPS[QOS:qosname]=hlmt
	MAXPS[USER]=slmt,hlmt MAXPS[USER:username]=slmt,hlmt
	MAXWC[GROUP]=slmt,hlmt MAXWC[GROUP:groupname]=slmt,hlmt
	MAXWC[QOS:qosname]=hlmt
	MAXWC[USER]=slmt,hlmt MAXWC[USER:username]=slmt,hlmt

Multi-dimension usage limit permutations	
GROUPCFG[name]	MAXJOB[CLASS:classname]=slmt,hlmt
	MAXJOB[USER]=slmt,hlmt MAXJOB[USER:username]=slmt,hlmt
	MAXMEM[CLASS:classname]=slmt,hlmt
	MAXMEM[USER]=slmt,hlmt MAXMEM[USER:username]=slmt,hlmt
	MAXNODE[CLASS:classname]=slmt,hlmt
	MAXNODE[USER]=slmt,hlmt MAXNODE[USER:username]=slmt,hlmt
	MAXPE[CLASS:classname]=slmt,hlmt
	MAXPE[USER]=slmt,hlmt MAXPE[USER:username]=slmt,hlmt
	MAXPROC[CLASS:classname]=slmt,hlmt
	MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=slmt,hlmt
	MAXPS[CLASS:classname]=slmt,hlmt
	MAXPS[USER]=slmt,hlmt MAXPS[USER:username]=slmt,hlmt
	MAXWC[CLASS:classname]=slmt,hlmt
	MAXWC[USER]=slmt,hlmt MAXWC[USER:username]=slmt,hlmt

Multi-dimension usage limit permutations	
QOSCFG[name]	MAXIJOB[ACCT]=hlmt MAXIJOB[ACCT:accountname]=hlmt
	MAXIPROC[ACCT]=hlmt MAXIPROC[ACCT:accountname]=hlmt
	MAXJOB[ACCT]=slmt,hlmt MAXJOB[ACCT:accountname]=slmt,hlmt
	MAXJOB[USER]=slmt,hlmt MAXJOB[USER:username]=slmt,hlmt
	MAXMEM[USER]=slmt,hlmt MAXMEM[USER:username]=slmt,hlmt
	MAXNODE[USER]=slmt,hlmt MAXNODE[USER:username]=slmt,hlmt
	MAXPE[ACCT]=slmt,hlmt MAXPE[ACCT:accountname]=slmt,hlmt
	MAXPE[USER]=slmt,hlmt MAXPE[USER:username]=slmt,hlmt
	MAXPROC[ACCT]=slmt,hlmt MAXPROC[ACCT:accountname]=slmt,hlmt
	MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=slmt,hlmt
	MAXPS[ACCT]=slmt,hlmt MAXPS[ACCT:accountname]=slmt,hlmt
	MAXPS[USER]=slmt,hlmt MAXPS[USER:username]=slmt,hlmt
	MAXWC[USER]=slmt,hlmt MAXWC[USER:username]=slmt,hlmt

Multi-dimension usage limit permutations	
USERCFG[name]	MAXJOB[GROUP]=slmt,hlmt MAXJOB[GROUP:groupname]=slmt,hlmt
	MAXMEM[GROUP]=slmt,hlmt MAXMEM[GROUP:groupname]=slmt,hlmt
	MAXNODE[GROUP]=slmt,hlmt MAXNODE[GROUP:groupname]=slmt,hlmt
	MAXPE[GROUP]=slmt,hlmt MAXPE[GROUP:groupname]=slmt,hlmt
	MAXPROC[GROUP]=slmt,hlmt MAXPROC[GROUP:groupname]=slmt,hlmt
	MAXPS[GROUP]=slmt,hlmt MAXPS[GROUP:groupname]=slmt,hlmt
	MAXWC[GROUP]=slmt,hlmt MAXWC[GROUP:groupname]=slmt,hlmt

Override Limits

Like all job credentials, the QoS object may be associated with resource usage limits. However, this credential can also be given special override limits that supersede the limits of other credentials, effectively causing all other limits of the same type to be ignored. See QoS Usage Limits and Overrides for a complete list of policies that can be overridden. The following configuration provides an example of this in the last line:

```
USERCFG[steve]    MAXJOB=2   MAXNODE=30
GROUPCFG[staff]   MAXJOB=5
CLASSCFG[DEFAULT] MAXNODE=16
CLASSCFG[batch]   MAXNODE=32
QOSCFG[hiprio]    OMAXJOB=3  OMAXNODE=64
```

*Only 3 hiprio QoS jobs may run simultaneously and hiprio QoS jobs may run with up to 64 nodes per credential ignoring other credential **MAXNODE** limits.*

Given the preceding configuration, assume a job is submitted with the credentials, user *steve*, group *staff*, class *batch*, and QoS *hiprio*.

Such a job will start so long as running it does not lead to any of the following conditions:

- Total nodes used by user *steve* does not exceed *64*.
- Total active jobs associated with user *steve* does not exceed *2*.
- Total active jobs associated with group *staff* does not exceed *5*.
- Total nodes dedicated to class *batch* does not exceed *64*.
- Total active jobs associated with QoS *hiprio* does not exceed *3*.

While the preceding example is a bit complicated for most sites, similar combinations may be required to enforce policies found on many systems.

Idle Job Limits

Idle (or queued) job limits control which jobs are eligible for scheduling. To be eligible for scheduling, a job must meet the following conditions:

- Be idle as far as the resource manager is concerned (no holds).
- Have all job prerequisites satisfied (no outstanding job or data dependencies).
- Meet all idle job throttling policies.

If a job fails to meet any of these conditions, it will not be considered for scheduling and will not accrue service based job prioritization. (See service component and JOBPRIOACCRUALPOLICY.) The primary purpose of idle job limits is to ensure fairness among competing users by preventing queue stuffing and other similar abuses. Queue stuffing occurs when a single entity submits large numbers of jobs, perhaps thousands, all at once so they begin accruing queue time based priority and remain first to run despite subsequent submissions by other users.

Idle limits are specified in a manner almost identical to active job limits with the insertion of the capital letter *I* into the middle of the limit name. Below are examples of the **MAXIJOB** and **MAXINODE** limits, which are idle limit equivalents to the **MAXJOB** on page 420 and **MAXNODE** on page 420 limits:

MAXIJOB	
Units	# of jobs
Description	Limits the number of idle (eligible) jobs a credential may have at any given time.
Example	`USERCFG[DEFAULT]  MAXIJOB=8` `GROUPCFG[staff]   MAXIJOB=2,4`

MAXINODE	
Units	# of nodes

MAXINODE	
Description	Limits the total number of compute nodes that can be requested by jobs in the eligible/idle queue at any time. Once the limit is exceeded, the remaining jobs will be placed in the blocked queue. The number of nodes is determined by *<tasks>* / *<maximumProcsOnOneNode>* or, if using **JOBNODEMATCHPOLICY** on page 960 *EXACTNODE*, by the number of nodes requested.
Example	`USERCFG[DEFAULT]  MAXINODE=2`

Idle limits can constrain the total number of jobs considered to be eligible on a per credential basis. Further, like active job limits, idle job limits can also constrain eligible jobs based on aggregate requested resources. This could, for example, allow a site to indicate that for a given user, only jobs requesting up to a total of 64 processors, or 3200 processor-seconds would be considered at any given time. Which jobs to select is accomplished by prioritizing all idle jobs and then adding jobs to the eligible list one at a time in priority order until jobs can no longer be added. This eligible job selection is done only once per scheduling iteration, so, consequently, idle job limits only support a single hard limit specification. Any specified soft limit is ignored.

All single dimensional job limit types supported as active job limits are also supported as idle job limits. In addition, Moab also supports MAXIJOB[USER] and MAXIPROC[USER] policies on a per class basis. (See Basic Fairness Policies.)

Example:

```
USERCFG[steve]     MAXIJOB=2
GROUPCFG[staff]    MAXIJOB=5
CLASSCFG[batch]    MAXIJOB[USER]=2 MAXIJOB[USER:john]=6
QOSCFG[hiprio]     MAXIJOB=3
```

Hard and Soft Limits

Hard and soft limit specification allows a site to balance both fairness and utilization on a given system. Typically, throttling limits are used to constrain the quantity of resources a given credential (such as user or group) is allowed to consume. These limits can be very effective in enforcing fair usage among a group of users. However, in a lightly loaded system, or one in which there are significant swings in usage from project to project, these limits can reduce system utilization by blocking jobs even when no competing jobs are queued.

Soft limits help address this problem by providing additional scheduling flexibility. They allow sites to specify two tiers of limits; the more constraining limits soft limits are in effect in heavily loaded situations and reflect tight fairness constraints. The more flexible hard limits specify how flexible the scheduler can be in selecting jobs when there are idle resources available after all jobs meeting the tighter soft limits have started. Soft and hard limits are specified in the format *[<SOFTLIMIT>,] <HARDLIMIT>*. For example, a given site may want to use the following configuration:

```
USERCFG[DEFAULT]  MAXJOB=2,8
```

With this configuration, the scheduler would select all jobs that meet the per user **MAXJOB** *limit of 2. It would then attempt to start and reserve resources for all of these selected jobs. If after doing so there still remain available resources, the scheduler would then select all jobs that meet the less constraining hard per user* **MAXJOB** *limit of 8 jobs. These jobs would then be scheduled and reserved as available resources allow.*

If no soft limit is specified or the soft limit is less constraining than the hard limit, the soft limit is set equal to the hard limit.

Example:

```
USERCFG[steve]    MAXJOB=2,4 MAXNODE=15,30
GROUPCFG[staff]   MAXJOB=2,5
CLASSCFG[DEFAULT] MAXNODE=16,32
CLASSCFG[batch]   MAXNODE=12,32
QOSCFG[hiprio]    MAXJOB=3,5 MAXNODE=32,64
```

Job preemption status can be adjusted based on whether the job violates a soft policy using the ENABLESPVIOLATIONPREEMPTION parameter.

Per-partition Limits

Per-partition scheduling can set limits and enforce credentials and polices on a per-partition basis. Configuration for per-partition scheduling is done on the grid head. In a grid, each Moab cluster is considered a partition. Per-partition scheduling is typically used in a Master/Slave grid.

To enable per-partition scheduling, add the following to `moab.cfg`:

```
PERPARTITIONSCHEDULING TRUE
JOBMIGRATEPOLICY JUSTINTIME
```

With per-partition scheduling, it is recommended that limits go on the specific partitions and not on the global level. If limits are specified on both levels, Moab will take the more constricting of the limits. Also, please note that a DEFAULT policy on the global partition is not overridden by any policy on a specific partition.

Per-partition Limits

You can configure per-job limits and credential usage limits on a per-partition basis in the `moab.cfg` file. Here is a sample configuration for partitions *g02* and *g03* in `moab.cfg`.

```
PARCFG[g02]   CONFIGFILE=/opt/moab/parg02.cfg
PARCFG[g03]   CONFIGFILE=/opt/moab/parg03.cfg
```

You can then add per-partition limits in each partition configuration file:

```
# /opt/moab/parg02.cfg
CLASSCFG[pbatch]   MAXJOB=5
```

```
# /opt/moab/parg03.cfg
CLASSCFG[pbatch]   MAXJOB=10
```

You can configure Moab so that jobs submitted to any partition besides *g02*and *g03* get the default limits in `moab.cfg`:

```
stl

CLASSCFG[pbatch]  MAXJOB=2
```

Supported Credentials and Limits

The user, group, account, QoS, and class credentials are supported in per-partition scheduling.

The following per-job limits are supported:

- MAX.NODE
- MAX.WCLIMIT
- MAX.PROC

The following credential usage limits are supported:

- MAXJOB
- MAXNODE
- MAXPROC
- MAXWC
- MAXSUBMITJOBS

Multi-dimensional limits are supported for the listed credentials and per-job limits. For example:

```
CLASSCFG[pbatch]   MAXJOB[user:frank]=10
```

Usage-based limits

Resource usage limits constrain the amount of resources a given job may consume. These limits are generally proportional to the resources requested and may include walltime, any standard resource, or any specified generic resource. The parameter RESOURCELIMITPOLICY controls which resources are limited, what limit policy is enforced per resource, and what actions the scheduler should take in the event of a policy violation.

Configuring Actions

The **RESOURCELIMITPOLICY** parameter accepts a number of policies, resources, and actions using the format and values defined below.

If walltime is the resource to be limited, be sure that the resource manager is configured to not interfere if a job surpasses its given walltime. For TORQUE, this is done by using $ignwalltime in the configuration on each MOM node.

Format

`RESOURCELIMITPOLICY<RESOURCE>:[<SPOLICY>,]<HPOLICY>:[<SACTION>,]<HACTION>[:[<SVIOLATIONTIME>,]<HVIOLATIONTIME>]...`

Resource	Description
CPUTIME	Maximum total job proc-seconds used by any single job (allows scheduler enforcement of cpulimit).
DISK	Local disk space (in MB) used by any single job task.
JOBMEM	Maximum real memory/RAM (in MB) used by any single job. JOBMEM will only work with the MAXMEM flag.
JOBPROC	Maximum processor load associated with any single job. You must set **MAXPROC** on page 421 to use **JOBPROC**.
MEM	Maximum real memory/RAM (in MB) used by any single job task.
MINJOBPROC	Minimum processor load associated with any single job (action taken if job is using 5% or less of potential CPU usage).
NETWORK	Maximum network load associated with any single job task.
PROC	Maximum processor load associated with any single job task.
SWAP	Maximum virtual memory/SWAP (in MB) used by any single job task.
WALLTIME	Requested job walltime.

Policy	Description
ALWAYS	take action whenever a violation is detected
EXTENDEDVIOLATION	take action only if a violation is detected and persists for greater than the specified time limit
BLOCKEDWORKLOADONLY	take action only if a violation is detected and the constrained resource is required by another job

Action	Description
CANCEL	terminate the job
CHECKPOINT	checkpoint and terminate job
MIGRATE	requeue the job and require a different set of hosts for execution
NOTIFY	notify admins and job owner regarding violation
REQUEUE	terminate and requeue the job
SUSPEND	suspend the job and leave it suspended for an amount of time defined by the MINADMINSTIME parameter

Example 3-93: Notify and then cancel job if requested memory is exceeded

```
# if job exceeds memory usage, immediately notify owner
# if job exceeds memory usage for more than 5 minutes, cancel the job
RESOURCELIMITPOLICY MEM:ALWAYS,EXTENDEDVIOLATION:NOTIFY,CANCEL:00:05:00
```

Example 3-94: Checkpoint job on walltime violations

```
# if job exceeds requested walltime, checkpoint job
RESOURCELIMITPOLICY WALLTIME:ALWAYS:CHECKPOINT
# when checkpointing, send term signal, followed by kill 1 minute later
RMCFG[base] TYPE=PBS CHECKPOINTTIMEOUT=00:01:00 CHECKPOINTSIG=SIGTERM
```

Example 3-95: Cancel jobs that use 5% or less of potential CPU usage for more than 5 minutes

```
RESOURCELIMITPOLICY MINJOBPROC:EXTENDEDVIOLATION:CANCEL:5:00
```

Example 3-96: Migrating a job when it blocks other workload

```
RESOURCELIMITPOLICY JOBPROC:BLOCKEDWORKLOADONLY:MIGRATE
```

Specifying Hard and Soft Policy Violations

Moab is able to perform different actions for both hard and soft policy violations. In most resource management systems, a mechanism does not exist to allow the user to specify both hard and soft limits. To address this, Moab provides the RESOURCELIMITMULTIPLIER parameter that allows per partition and per resource multiplier factors to be specified to generate the actual hard and soft limits to be used. If the factor is less than one, the soft limit will be lower than the specified value and a Moab action will be taken before the specified limit is reached. If the factor is greater than one, the hard limit will be set higher than the specified limit allowing a buffer space before the hard limit action is taken.

In the following example, job owners will be notified by email when their memory reaches 100% of the target, and the job will be canceled if it reaches 125% of the target. For wallclock usage, the job will be requeued when it reaches 90% of the specified limit if another job is waiting for its resources, and it will be checkpointed when it reaches the full limit.

```
RESOURCELIMITPOLICY        MEM:ALWAYS,ALWAYS:NOTIFY,CANCEL
RESOURCELIMITPOLICY        WALLTIME:BLOCKEDWORKLOADONLY,ALWAYS:REQUEUE,CHECKPOINT
RESOURCELIMITMULTIPLIER    MEM:1.25,WALLTIME:0.9
```

Constraining Walltime Usage

While Moab constrains walltime using the parameter RESOURCELIMITPOLICY like other resources, it also allows walltime exception policies which are not available with other resources. In particular, Moab allows jobs to exceed the requested wallclock limit by an amount specified on a global basis using the JOBMAXOVERRUN parameter or on a per credential basis using the **OVERRUN** attribute of the CLASSCFG parameter.

```
JOBMAXOVERRUN     00:10:00
CLASSCFG[debug]   overrun=00:00:30
```

Related topics

- RESOURCELIMITPOLICY parameter
- FSTREE parameter (set usage limits within share tree hierarchy)
- Credential Overview
- JOBMAXOVERRUN parameter
- WCVIOLATIONACTION parameter
- RESOURCELIMITMULTIPLIER parameter

Fairshare

Fairshare allows historical resource utilization information to be incorporated into job feasibility and priority decisions. This feature allows site administrators to set system utilization targets for users, groups, accounts, classes, and QoS levels. Administrators can also specify the time frame over which resource utilization is evaluated in determining whether the goal is being reached. Parameters allow sites to specify the utilization metric, how historical information is aggregated, and the effect of fairshare state on scheduling behavior. You can specify fairshare targets for any credentials (such as user, group, and class) that administrators want such information to affect.

Fairshare Parameters

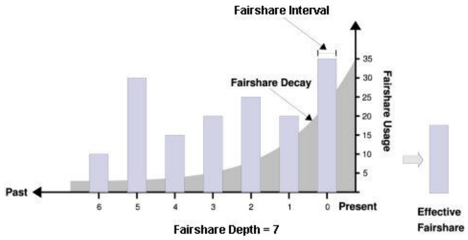
Fairshare is configured at two levels. First, at a system level, configuration is required to determine how fairshare usage information is to be collected and processed. Second, some configuration is required at the credential level to determine how this fairshare information affects particular jobs. The following are system level parameters:

Parameter	Description
FSINTERVAL	Duration of each fairshare window.
FSDEPTH	Number of fairshare windows factored into current fairshare utilization.
FSDECAY	Decay factor applied to weighting the contribution of each fairshare window.
FSPOLICY	Metric to use when tracking fairshare usage.

Credential level configuration consists of specifying fairshare utilization targets using the ***CFG** suite of parameters, including ACCOUNTCFG, CLASSCFG, GROUPCFG, QOSCFG, and USERCFG.

If global (multi-cluster) fairshare is used, Moab must be configured to synchronize this information with an identity manager.

Image 3-3: Effective fairshare over 7 days



FSPOLICY - Specifying the Metric of Consumption

As Moab runs, it records how available resources are used. Each iteration (**RMPOLLINTERVAL** seconds) it updates fairshare resource utilization statistics. Resource utilization is tracked in accordance with the FSPOLICY parameter allowing various aspects of resource consumption information to be measured. This parameter allows selection of both the types of resources to be tracked as well as the method of tracking. It provides the option of tracking usage by dedicated or consumed resources, where dedicated usage tracks what the scheduler assigns to the job and consumed usage tracks what the job actually uses.

Metric	Description
DEDICATEDPES	Usage tracked by processor-equivalent seconds dedicated to each job. This is based on the total number of dedicated processor-equivalent seconds delivered in the system. Useful in dedicated and shared nodes environments.
DEDICATEDPS	Usage tracked by processor seconds dedicated to each job. This is based on the total number of dedicated processor seconds delivered in the system. Useful in dedicated node environments.
DEDICATEDPS%	Usage tracked by processor seconds dedicated to each job. This is based on the total number of dedicated processor seconds *available* in the system.
UTILIZEDPS	Usage tracked by processor seconds used by each job. This is based on the total number of utilized processor seconds delivered in the system. Useful in shared node/SMP environments.

Example 3-97:

An example may clarify the use of the **FSPOLICY** parameter. Assume a 4-processor job is running a parallel `/bin/sleep` for 15 minutes. It will have a dedicated fairshare usage of 1 processor-hour but a consumed fairshare usage of essentially nothing since it did not consume anything. Most often, dedicated fairshare usage is used on dedicated resource platforms while consumed tracking is used in shared SMP environments.

```
FSPOLICY    DEDICATEDPS%
FSINTERVAL  24:00:00
FSDEPTH     28
FSDECAY     0.75
```

Specifying Fairshare Timeframe

When configuring fairshare, it is important to determine the proper timeframe that should be considered. Many sites choose to incorporate historical usage information from the last one to two weeks while others are only concerned about the events of the last few hours. The correct setting is very site dependent and usually incorporates both average job turnaround time and site mission policies.

With Moab's fairshare system, time is broken into a number of distinct fairshare windows. Sites configure the amount of time they want to consider by specifying two parameters, FSINTERVAL and FSDEPTH. The **FSINTERVAL** parameter specifies the duration of each window while the **FSDEPTH** parameter indicates the number of windows to consider. Thus, the total time evaluated by fairshare is simply **FSINTERVAL** * **FSDEPTH**.

Many sites want to limit the impact of fairshare data according to its age. The FSDECAY parameter allows this, causing the most recent fairshare data to contribute more to a credential's total fairshare usage than older data. This parameter is specified as a standard decay factor, which is applied to the fairshare data. Generally, decay factors are specified as a value between 1 and 0 where a value of *1* (the default) indicates no decay should be specified. The smaller the number, the more rapid the decay using the calculation `WeightedValue = Value * <DECAY> ^ <N>` where *<N>* is the window number. The following table shows the impact of a number of commonly used decay factors on the percentage contribution of each fairshare window.

Decay Factor	Win0	Win1	Win2	Win3	Win4	Win5	Win6	Win7
1.00	100%	100%	100%	100%	100%	100%	100%	100%
0.80	100%	80%	64%	51%	41%	33%	26%	21%
0.75	100%	75%	56%	42%	31%	23%	17%	12%
0.50	100%	50%	25%	13%	6%	3%	2%	1%

While selecting how the total fairshare time frame is broken up between the number and length of windows is a matter of preference, it is important to note that more windows will cause the decay factor to degrade the contribution of aged data more quickly.

Managing Fairshare Data

Using the selected fairshare usage metric, Moab continues to update the current fairshare window until it reaches a fairshare window boundary, at which point it rolls the fairshare window and begins updating the new window. The information for each window is stored in its own file located in the Moab statistics directory. Each file is named `FS.<EPOCHTIME>[.<PNAME>]` where *<EPOCHTIME>* is the time the new fairshare window became active (see sample data file) and *<PNAME>* is only used if per-partition share trees are configured. Each window contains utilization information for each entity as well as for total usage.

> Historical fairshare data is recorded in the fairshare file using the metric specified by the FSPOLICY parameter. By default, this metric is processor-seconds.

> Historical fairshare data can be directly analyzed and reported using the mdiag -f -v command.

When Moab needs to determine current fairshare usage for a particular credential, it calculates a decay-weighted average of the usage information for that credential using the most recent fairshare intervals where the number of windows evaluated is controlled by the FSDEPTH parameter. For example, assume the credential of interest is user *john* and the following parameters are set:

```
FSINTERVAL   12:00:00
FSDEPTH      4
FSDECAY      0.5
```

Further assume that the fairshare usage intervals have the following usage amounts:

Fairshare interval	Total user *john* usage	Total cluster usage
0	60	110
1	0	125
2	10	100
3	50	150

Based on this information, the current fairshare usage for user *john* would be calculated as follows:

*Usage = (60 * 1 + .5^1 * 0 + .5^2 * 10 + .5^3 * 50) / (110 + .5^1*125 + .5^2*100 + .5^3*150)*

> The current fairshare usage is relative to the actual resources delivered by the system over the timeframe evaluated, not the resources available or configured during that time.

> Historical fairshare data is organized into a number of data files, each file containing the information for a length of time as specified by the FSINTERVAL parameter. Although **FSDEPTH**, **FSINTERVAL**, and **FSDECAY** can be freely and dynamically modified, such changes may result in unexpected fairshare status for a period of time as the fairshare data files with the old **FSINTERVAL** setting are rolled out.

Using Fairshare Information

Fairshare Targets

Once the global fairshare policies have been configured, the next step involves applying resulting fairshare usage information to affect scheduling behavior. As mentioned in the Fairshare Overview, by specifying fairshare targets, site administrators can configure how fairshare information impacts scheduling behavior. The targets can be applied to user, group, account, QoS, or class credentials using the **FSTARGET** attribute of ***CFG** credential parameters. These targets allow fairshare information to affect job priority and each target can be independently selected to be one of the types documented in the following table:

Target type - Ceiling	
Target modifier	-

Target type - Ceiling	
Job impact	Priority
Format	Percentage Usage
Description	Adjusts job priority down when usage exceeds target. See **How violated ceilings and floors affect fairshare-based priority** on page 387 for more information on how ceilings affect job priority.

Target type - Floor	
Target modifier	+
Job impact	Priority
Format	Percentage Usage
Description	Adjusts job priority up when usage falls below target. See **How violated ceilings and floors affect fairshare-based priority** on page 387 for more information on how floors affect job priority.

Target type - Target	
Target modifier	N/A
Job impact	Priority
Format	Percentage Usage
Description	Adjusts job priority when usage does not meet target.

Setting a fairshare target value of *0* indicates that there is no target and that the priority of jobs associated with that credential should not be affected by the credential's previous fairshare target. If you want a credential's cluster usage near 0%, set the target to a very small value, such as *0.001*.

Example

The following example increases the priority of jobs belonging to user *john* until he reaches 16.5% of total cluster usage. All other users have priority adjusted both up and down to bring them to their target usage of 10%:

```
FSPOLICY          DEDICATEDPS
FSWEIGHT          1
FSUSERWEIGHT      100
USERCFG[john]     FSTARGET=16.5+
USERCFG[DEFAULT]  FSTARGET=10
...
```

Fairshare Caps

Where fairshare targets affect a job's priority and position in the eligible queue, fairshare caps affect a job's eligibility. Caps can be applied to users, accounts, groups, classes, and QoSs using the **FSCAP** attribute of ***CFG** credential parameters and can be configured to modify scheduling behavior. Unlike fairshare targets, if a credential reaches its fairshare cap, its jobs can no longer run and are thus removed from the eligible queue and placed in the blocked queue. In this respect, fairshare targets behave like soft limits and fairshare caps behave like hard limits. Fairshare caps can be absolute or relative as described in the following table. If no modifier is specified, the cap is interpreted as relative.

Absolute Cap	
Cap Modifier:	^
Job Impact:	Feasibility
Format:	Absolute Usage
Description:	Constrains job eligibility as an absolute quantity measured according to the scheduler charge metric as defined by the FSPOLICY parameter

Relative Cap	
Cap Modifier:	%
Job Impact:	Feasibility
Format:	Percentage Usage
Description:	Constrains job eligibility as a percentage of total delivered cycles measured according to the scheduler charge metric as defined by the FSPOLICY parameter.

Example

The following example constrains the *marketing* account to use no more than *16,500* processor seconds during any given floating one week window. At the same time, all other accounts are constrained to use no more than *10%* of the total delivered processor seconds during any given one week window.

```
FSPOLICY            DEDICATEDPS
FSINTERVAL          12:00:00
FSDEPTH             14
ACCOUNTCFG[marketing] FSCAP=16500^
ACCOUNTCFG[DEFAULT]   FSCAP=10
...
```

Priority-Based Fairshare

The most commonly used type of fairshare is priority based fairshare. In this mode, fairshare information does not affect whether a job can run, but rather only the job's priority relative to other jobs. In most cases, this is the desired behavior. Using the standard fairshare target, the priority of jobs of a particular user who has used too many resources over the specified fairshare window is lowered. Also, the standard fairshare target increases the priority of jobs that have not received enough resources.

While the standard fairshare target is the most commonly used, Moab can also specify fairshare ceilings and floors. These targets are like the default target; however, ceilings only adjust priority down when usage is too high and floors only adjust priority up when usage is too low.

Since fairshare usage information must be integrated with Moab's overall priority mechanism, it is critical that the corresponding fairshare priority weights be set. Specifically, the FSWEIGHT component weight parameter and the target type subcomponent weight (such as FSACCOUNTWEIGHT, FSCLASSWEIGHT, FSGROUPWEIGHT, FSQOSWEIGHT, and FSUSERWEIGHT) be specified.

> If these weights are not set, the fairshare mechanism will be enabled but have no effect on scheduling behavior. See the Job Priority Factor Overview for more information on setting priority weights.

Example

```
# set relative component weighting
FSWEIGHT      1
FSUSERWEIGHT  10
FSGROUPWEIGHT 50

FSINTERVAL  12:00:00
FSDEPTH     4
FSDECAY     0.5
FSPOLICY    DEDICATEDPS
# all users should have a FS target of 10%
USERCFG[DEFAULT] FSTARGET=10.0
# user john gets extra cycles
USERCFG[john]    FSTARGET=20.0
# reduce staff priority if group usage exceed 15%
GROUPCFG[staff]  FSTARGET=15.0-
# give group orion additional priority if usage drops below 25.7%
GROUPCFG[orion]  FSTARGET=25.7+
```

Job preemption status can be adjusted based on whether the job violates a fairshare target using the ENABLEFSVIOLATIONPREEMPTION parameter.

Credential-Specific Fairshare Weights

Credential-specific fairshare weights can be set using the **FSWEIGHT** attribute of the ACCOUNT, GROUP, and QOS credentials as in the following example:

```
FSWEIGHT  1000
 ACCOUNTCFG[orion1] FSWEIGHT=100
 ACCOUNTCFG[orion2] FSWEIGHT=200
 ACCOUNTCFG[orion3] FSWEIGHT=-100
 GROUPCFG[staff] FSWEIGHT=10
```

If specified, a per-credential fairshare weight is added to the global component fairshare weight.

The **FSWEIGHT** attribute is only enabled for ACCOUNT, GROUP, and QOS credentials.

Extended Fairshare Examples

Example 3-98: Multi-Cred Cycle Distribution

Example 1 represents a university setting where different schools have access to a cluster. The Engineering department has put the most money into the cluster and therefore has greater access to the cluster. The Math, Computer Science, and Physics departments have also pooled their money into the cluster and have reduced relative access. A support group also has access to the cluster, but since they only require minimal compute time and shouldn't block the higher-paying departments, they are constrained to five percent of the cluster. At this time, users Tom and John have specific high-priority projects that need increased cycles.

```
#global general usage limits - negative priority jobs are considered in scheduling
ENABLENEGJOBPRIORITY   TRUE
# site policy - no job can last longer than 8 hours
USERCFG[DEFAULT] MAX.WCLIMIT=8:00:00
# Note:  default user FS target only specified to apply default user-to-user balance
USERCFG[DEFAULT] FSTARGET=1
# high-level fairshare config
FSPOLICY        DEDICATEDPS
FSINTERVAL      12:00:00
FSDEPTH         32 #recycle FS every 16 days
FSDECAY         0.8 #favor more recent usage info
# qos config
QOSCFG[inst]     FSTARGET=25
QOSCFG[supp]     FSTARGET=5
QOSCFG[premium] FSTARGET=70
# account config (QoS access and fstargets)
# Note:  user-to-account mapping handled via allocation manager
# Note:  FS targets are percentage of total cluster, not percentage of QOS
ACCOUNTCFG[cs]   QLIST=inst    FSTARGET=10
ACCOUNTCFG[math] QLIST=inst    FSTARGET=15

ACCOUNTCFG[phys] QLIST=supp    FSTARGET=5
ACCOUNTCFG[eng]  QLIST=premium FSTARGET=70
# handle per-user priority exceptions
USERCFG[tom]  PRIORITY=100
USERCFG[john] PRIORITY=35
# define overall job priority
USERWEIGHT        10  # user exceptions
# relative FS weights (Note: QOS overrides ACCOUNT which overrides USER)
FSUSERWEIGHT      1
FSACCOUNTWEIGHT   10
FSQOSWEIGHT       100
# apply XFactor to balance cycle delivery by job size fairly
# Note:  queuetime factor also on by default (use QUEUETIMEWEIGHT to adjust)
XFACTORWEIGHT     100
# enable preemption
PREEMPTPOLICY   REQUEUE
# temporarily allow phys to preempt math
ACCOUNTCFG[phys] JOBFLAGS=PREEMPTOR PRIORITY=1000
ACCOUNTCFG[math] JOBFLAGS=PREEMPTEE
```

Hierarchical Fairshare/Share Trees

Moab supports arbitrary depth hierarchical fairshare based on a share tree. In this model, users, groups, classes, and accounts can be arbitrarily organized and their usage tracked and limited. Moab extends common share tree concepts to allow mixing of credential types, enforcement of ceiling and floor style usage targets, and mixing of hierarchical fairshare state with other priority components.

Defining the Tree

The FSTREE parameter can be used to define and configure the share tree used in fairshare configuration. This parameter supports the following attributes:

SHARES	
Format:	<COUNT>[@<PARTITION>][,<COUNT>[@<PARTITION>]]... where <COUNT> is a double and <PARTITION> is a specified partition name.
Description:	Specifies the node target usage or share.
Example:	`FSTREE[Eng]   SHARES=1500.5` `FSTREE[Sales] SHARES=2800`

MEMBERLIST	
Format:	Comma delimited list of child nodes of the format `[<OBJECT_TYPE>]:<OBJECT_ID>` where object types are only specified for *leaf nodes* associated with **user**, **group**, **class**, **qos**, or **acct** credentials.
Description:	Specifies the tree objects associated with this node.
Example:	`FSTREE[root]   SHARES=100      MEMBERLIST=Eng,Sales` `FSTREE[Eng]    SHARES=1500.5   MEMBERLIST=user:john,user:steve,user:bob` `FSTREE[Sales]  SHARES=2800     MEMBERLIST=Sales1,Sales2,Sales3` `FSTREE[Sales1] SHARES=30       MEMBERLIST=user:kellyp,user:sam` `FSTREE[Sales2] SHARES=10       MEMBERLIST=user:ux43,user:ux44,user:ux45` `FSTREE[Sales3] SHARES=60       MEMBERLIST=user:robert,user:tjackson`

Current tree configuration and monitored usage distribution is available using the mdiag -f -v commands.

Controlling Tree Evaluation

Moab provides multiple policies to customize how the share tree is evaluated.

Policy	Description
FSTREETIERMULTIPLIER	Decreases the value of sub-level usage discrepancies. It can be a positive or negative value. When positive, the parent's usage in the tree takes precedence; when negative, the child's usage takes precedence. The usage amount is not changed, only the coefficient used when calculating the value of fstree usage in priority. When using this parameter, it is recommended that you research how it changes the values in `mdiag -p` to determine the appropriate use.
FSTREECAP	Caps lower level usage factors to prevent them from exceeding upper tier discrepancies.

Using FS Floors and Ceilings with Hierarchical Fairshare

All standard fairshare facilities including target floors, target ceilings, and target caps are supported when using hierarchical fairshare.

Multi-Partition Fairshare

Moab supports independent, per-partition hierarchical fairshare targets allowing each partition to possess independent prioritization and usage constraint settings. This is accomplished by setting the **PERPARTITIONSCHEDULING** attribute of the **FSTREE** parameter to *TRUE* in `moab.cfg` and setting `partition="name"` in your `<fstree>` leaf.

```
FSTREE[tree]
  <fstree>
    <tnode partition="slave1" name="root" type="acct" share="100" limits="MAXJOB=6">
      <tnode name="accta" type="acct" share="50" limits="MAXSUBMITJOBS=2 MAXJOB=1">
        <tnode name="fred" type="user" share="1" limits="MAXWC=1:00:00">
        </tnode>
      </tnode>
      <tnode name="acctb" type="acct" share="50" limits="MAXSUBMITJOBS=4 MAXJOB=3">
        <tnode name="george" type="user" share="1" >
        </tnode>
      </tnode>
    </tnode>
    <tnode partition="slave2" name="root" type="acct" share="100"
limits="MAXSUBMITJOBS=6 MAXJOB=5">
      <tnode name="accta" type="acct" share="50">
        <tnode name="paul" type="user" share="1">
        </tnode>
      </tnode>
      <tnode name="acctb" type="acct" share="50">
        <tnode name="ringo" type="user" share="1">
        </tnode>
      </tnode>
    </tnode>
  </fstree>
```

If no partition is specified for a given share value, then this value is assigned to the global partition. If a partition exists for which there are no explicitly specified shares for any node, this partition will use the share distribution assigned to the global partition.

Dynamically Importing Share Tree Data

Share trees can be centrally defined within a database, flat file, information service, or other system and this information can be dynamically imported and used within Moab by setting the **FSTREE** parameter within the **Identity Managers** on page 765. This interface can be used to load current information at startup and periodically synchronize this information with the master source.

To create a fairshare tree in a separate XML file and import it into Moab

1. Create a file to store your fair share tree specification. Give it a descriptive name and store it in your Moab home directory (`$MOABHOMEDIR` or `$MOABHOMEDIR/etc`). In this example, the file is called `fstree.dat`.
2. In the first line of `fstree.dat`, set **FSTREE**`[myTree]` to indicate that this is a fairshare file.
3. Build a tree in XML to match your needs. For example:

```
FSTREE[myTree]
<fstree>
<tnode name="root" share="100">
<tnode name="john" type="user" share="50" limits="MAXJOB=8 MAXPROC=24
MAXWC=01:00:00"></tnode>
<tnode name="jane" type="user" share="50" limits="MAXJOB=5"></tnode>
</tnode>
</fstree>
```

This configuration creates a fairshare tree in which users share a value of 100. Users `john` and `jane` share the value equally, because each has been given 50.

Because 100 is an arbitrary number, users `john` and `jane` could be assigned `10000` and `10000` respectively and still have a 50% share under the parent leaf. To keep the example simple, however, it is recommended that you use 100 as your arbirary share value and distribute the share as percentages. In this case, `john` and `jane` each have 50%.

If the users' numbers do not add up to at least the fairshare value of 100, the remaining value is shared among all users under the tree. For instance, if the tree had a value of 100, user `john` had a value of 50, and user `jane` had a value of 25, then 25% of the fairshare tree value would belong to all other users associated with the tree. By default, tree leaves do not limit who can run under them.

Each value specified in the `tnode` elements must be contained in quotation marks.

4. Optional: Share trees defined within a flat file can be cumbersome; consider running tidy for xml to improve readability. Sample usage:

```
> tidy -i -xml goldy.cfg <filename> <output file>

# Sample output

FSTREE[myTree]
<fstree>
  <tnode name="root" share="100">
    <tnode name="john"   type="user" share="50" limits="MAXJOB=8
    MAXPROC=24 MAXWC=01:00:00">
    </tnode>
    <tnode name="jane" type="user" share="50" limits="MAXJOB=5">
    </tnode>
  </tnode>
</fstree>
```

5. Link the new file to Moab using the IDCFG parameter in your Moab configuration file.

```
IDCFG[myTree] server="FILE:///$MOABH OMEDIR/etc/fstree.dat" REFRESHPERIOD=INFINITY
```

*Moab imports the `myTree` fairshare tree from the `fstree.dat` file. Setting **REFRESHPERIOD** to INFINITY causes Moab to read the file each time it starts or restarts, but other settings (hour, day, month) cause Moab to read the file more often (See Refreshing Identity Manager Data for more information).*

6. To view your fairshare tree configuration, run mdiag -f. If it is configured correctly, the tree information will appear beneath all the information about your fairshare settings configured in `moab.cfg`.

```
> mdiag -f
Share Tree Overview for partition 'ALL'
Name          Usage    Target                (FSFACTOR)
----          -----    ------              -------------
root         100.00    100.00  of   100.00 (node: 1171.81) (0.00)
- john        16.44     50.00  of   100.00 (user: 192.65) (302.04) MAXJOB=8
MAXPROC=24 MAXWC=3600
- jane        83.56     50.00  of   100.00 (user: 979.16) (-302.04)  MAXJOB=5
```

The settings you configured in fstree.dat appear in the output. The tree of 100 is shared equally between users `john` *and* `jane`*.*

Specifying Share Tree Based Limits

Limits can be specified on internal nodes of the share tree using standard credential limit semantics. The following credential usage limits are valid:

- MAXIJOB (Maximum number of idle jobs allowed for the credential)
- **MAXJOB** on page 420
- **MAXMEM** on page 420
- **MAXNODE** on page 420
- **MAXPROC** on page 421
- **MAXSUBMITJOBS** on page 422
- **MAXWC** on page 422

Example 3-99: FSTREE limits example

```
FSTREE[myTree]
<fstree>
  <tnode name="root" share="100">
   <tnode name="john"   type="user" share="50" limits="MAXJOB=8
    MAXPROC=24 MAXWC=01:00:00">
    </tnode>
    <tnode name="jane" type="user" share="50" limits="MAXJOB=5">
    </tnode>
  </tnode>
</fstree>
```

Other Uses of Share Trees

If a share tree is defined, it can be used for purposes beyond fairshare, including organizing general usage and performance statistics for reporting purposes (see showstats -T), enforcement of tree node based usage limits, and specification of resource access policies.

Related topics

- mdiag -f command (provides diagnosis and monitoring of the fairshare facility)
- FSENABLECAPPRIORITY parameter
- ENABLEFSPREEMPTION parameter
- FSTARGETISABSOLUTE parameter

Sample FairShare Data File

```
FS.<EPOCHTIME>
```

```
# FS Data File (Duration:  43200 seconds)  Starting: Sat Jul  8 06:00:20
user                jvella   134087.910
user              reynolds    98283.840
user                gastor    18751.770
user                uannan   145551.260
user               mwillis   149279.140
...
group              DEFAULT   411628.980
group              RedRock  3121560.280
group               Summit   500327.640
group               Arches  3047918.940
acct        Administration   653559.290
acct           Engineering  4746858.620
acct                Shared    75033.020
acct              Research  1605984.910
qos               Deadline  2727971.100
qos           HighPriority  4278431.720
qos               STANDARD    75033.020
class                batch  7081435.840
sched             iCluster  7081435.840
```

The total usage consumed in this time interval is 7081435.840 processor-seconds. Since every job in this example scenario had a user, group, account, and QOS assigned to it, the sum of the usage of all members of each category should equal the total usage value: USERA + USERB + USERC + USERD = GROUPA + GROUPB = ACCTA + ACCTB + ACCTC = QOS0 + QOS1 + QOS2 = SCHED.

Controlling Resource Access - Reservations, Partitions, and QoS Facilities

Advance Reservations

An advance reservation is the mechanism by which Moab guarantees the availability of a set of resources at a particular time. Each reservation consists of three major components: (1) a set of resources, (2) a time frame, and (3) an access control list. It is a scheduler role to ensure that the access control list is not violated during the reservation's lifetime (that is, its time frame) on the resources listed. For example, a reservation may specify that node002 is reserved for user Tom on Friday. The scheduler is thus constrained to make certain that only Tom's jobs can use node002 at any time on Friday. Advance reservation technology enables many features including backfill, deadline based scheduling, grid scheduling, and QOS support.

The mrsvctl command is used to create, modify, query, and release reservations.

- Reservation Overview
- Administrative Reservations

Reservation Overview

Every reservation consists of 3 major components: (1) a set of resources, (2) a time frame, and (3) an access control list. Additionally, a reservation may also have a number of optional attributes controlling its behavior and interaction with other aspects of scheduling. Reservation attribute descriptions follow.

Resources

Under Moab, the resources specified for a reservation are specified by way of a task description. Conceptually, a task can be thought of as an atomic, or indivisible, collection of resources. If reservation resources are unspecified, a task is a node by default. To define a task, specify resources. The resources may include processors, memory, swap, local disk, and so forth. For example, a single task may consist of one processor, 2 GB of memory, and 10 GB of local disk.

A reservation consists of one or more tasks. In attempting to locate the resources required for a particular reservation, Moab examines all feasible resources and locates the needed resources in groups specified by the task description. An example may help clarify this concept:

Reservation A requires four tasks. Each task is defined as 1 processor and 1 GB of memory.

Node X has 2 processors and 3 GB of memory available
Node Y has 2 processors and 1 GB of memory available
Node Z has 2 processors and 2 GB of memory available

When collecting the resources needed for the reservation, Moab examines each node in turn. Moab finds that Node X can support 2 of the 4 tasks needed by reserving 2 processors and 2 GB of memory, leaving 1 GB of memory unreserved. Analysis of Node Y shows that it can only support 1 task reserving 1 processor and 1 GB of memory, leaving 1 processor unreserved. Note that the unreserved memory on Node X cannot be combined with the unreserved processor on Node Y to satisfy the needs of another task because a task requires all resources to be located on the same node. Finally, analysis finds that node Z can support 2 tasks, fully reserving all of its resources.

Both reservations and jobs use the concept of a task description in specifying how resources should be allocated. It is important to note that although a task description is used to allocate resources to a reservation, this description does not in any way constrain the use of those resources by a job. In the above example, a job requesting resources simply sees 4 processors and 4 GB of memory available in reservation A. If the job has access to the reserved resources and the resources meet the other requirements of the job, the job could use these resources according to its own task description and needs.

Currently, the resources that can be associated with reservations include processors, memory, swap, local disk, initiator classes, and any number of arbitrary resources. Arbitrary resources may include peripherals such as tape drives, software licenses, or any other site specific resource.

Time Frame

Associated with each reservation is a time frame. This specifies when the resources will be reserved or dedicated to jobs that meet the reservation's access control list (ACL). The time frame simply consists of a start time and an end time. When configuring a reservation, this information may be specified as a start time together with either an end time or a duration.

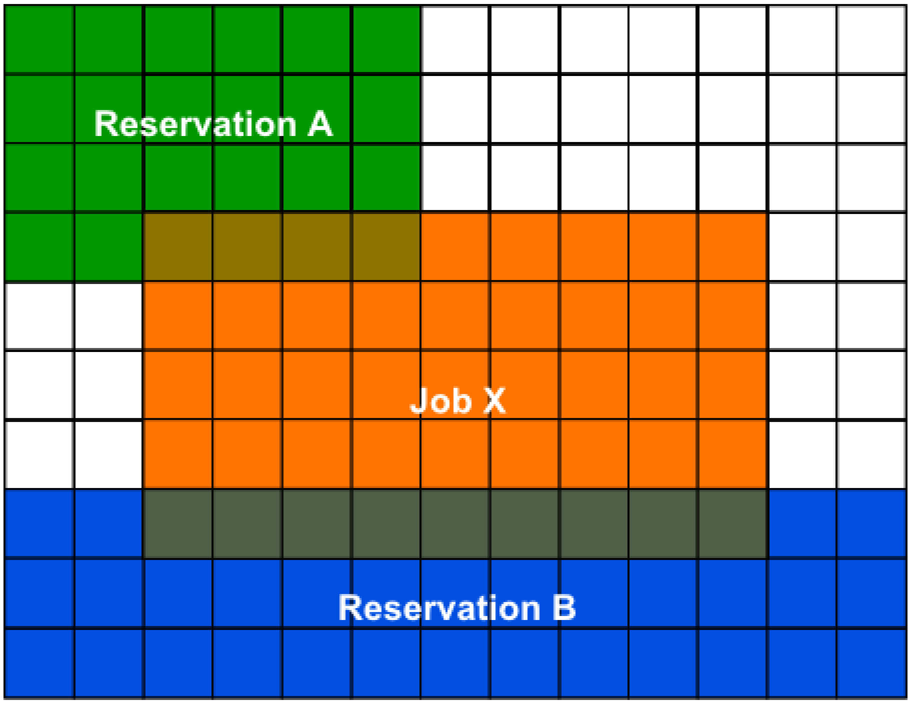
Access Control List

A reservation's access control list specifies which jobs can use a reservation. Only jobs that meet one or more of a reservation's access criteria are allowed to use the reserved resources during the reservation time frame. Currently, the reservation access criteria include the following: users, groups, accounts, classes, QOS, job attributes, job duration, and job templates.

Job to Reservation Binding

While a reservation's ACL will allow particular jobs to use reserved resources, it does not force any job to use these resources. With each job, Moab attempts to locate the best possible combination of available resources whether these are reserved or unreserved. For example, in the following figure, note that job **X**, which meets access criteria for both reservation **A** and **B**, allocates a portion of its resources from each reservation and the remainder from resources outside of both reservations.

Image 3-4: Job X uses resources from reservations A and B



Although by default, reservations make resources available to jobs that meet particular criteria, Moab can be configured to constrain jobs to only run within accessible reservations. This can be requested by the user on a job by job basis using a resource manager extension flag, or it can be enabled administratively via a QoS flag. For example, assume two reservations were created as follows:

```
> mrsvctl -c -a GROUP==staff -d 8:00:00 -h 'node[1-4]'
reservation staff.1 created
```

```
> mrsvctl -c -a USER==john -t 2
reservation john.2 created
```

If the user "john," who happened to also be a member of the group "staff," wanted to force a job to run within a particular reservation, "john" could do so using the **FLAGS** resource manager extension. Specifically, in the case of a PBS job, the following submission would force the job to run within the "staff.1" reservation.

```
> msub -l nodes=1,walltime=1:00:00,flags=ADVRES:staff.1 testjob.cmd
```

Note that for this to work, PBS needs to have resource manager extensions enabled as described in the PBS Resource Manager Extension Overview. (TORQUE has resource manager extensions enabled by default.) If the user wants the job to run on reserved resources but does not care which, the user could submit the job with the following:

```
> msub -l nodes=1,walltime=1:00:00,flags=ADVRES testjob.cmd
```

To enable job to reservation mapping via QoS, the QoS flag USERESERVED should be set in a similar manner.

Use the reservation BYNAME flag to require explicit binding for reservation access.

To lock jobs linked to a particular QoS into a reservation or reservation group, use the REQRID attribute.

Reservation Specification

There are two main types of reservations that sites typically deal with. The first, administrative reservations, are typically one-time reservations created for special purposes and projects. These reservations are created using the mrsvctl or setres commands. These reservations provide an integrated mechanism to allow graceful management of unexpected system maintenance, temporary projects, and time critical demonstrations. This command allows an administrator to select a particular set of resources or just specify the quantity of resources needed. For example an administrator could use a regular expression to request a reservation be created on the nodes "blue0[1-9]" or could simply request that the reservation locate the needed resources by specifying a quantity based request such as "TASKS==20."

The second type of reservation is called a standing reservation. It is specified using the SRCFG parameter and is of use when there is a recurring need for a particular type of resource distribution. Standing reservations are a powerful, flexible, and efficient means for enabling persistent or periodic policies such as those often enabled using classes or queues. For example, a site could use a standing reservation to reserve a subset of its compute resources for quick turnaround jobs during business hours on Monday thru Friday. The Standing Reservation Overview provides more information about configuring and using these reservations.

Reservation Behavior

As previously mentioned, a given reservation may have one or more access criteria. A job can use the reserved resources if it meets at least one of these access criteria. It is possible to stack multiple reservations on the same node. In such a situation, a job can only use the given node if it has access to each active reservation on the node.

Reservation Group

Reservations groups are ways of associating multiple reservations. This association is useful for variable namespace and reservation requests. The reservations in a group inherit the variables from the reservation group head, but if the same variable is set locally on a reservation in the group, the local variable overrides the inherited variable. Variable inheritance is useful for triggers as it provides greater flexibility with automating certain tasks and system behaviors.

Jobs may be bound to a reservation group (instead of a single reservation) by using the resource manager extension ADVRES.

Infinite Jobs and Reservations

To allow infinite walltime jobs, you must have the following scheduler flag set:

```
SCHEDCFG[Moab] FLAGS=allowinfinitejobs
```

You can submit an infinite job by completing:

```
msub -l walltime=INFINITY
```

Or an infinite reservation by completing:

```
mrsvctl -c -d INFINITY
```

Infinite jobs can run in infinite reservations. Infinite walltime also works with job templates and advres.

Output XML for infinite jobs will print "INFINITY" in the ReqAWDuration, and XML for infinite rsvs will print "INFINITY" in duration and endtime.


```
<Data>
   <rsv AUser="jgardner" AllocNodeCount="1" AllocNodeList="n5"
   AllocProcCount="4" AllocTaskCount="1" HostExp="n5"
   LastChargeTime="0" Name="jgardner.1" Partition="base"
   ReqNodeList="n5:1" Resources="PROCS=[ALL]" StatCAPS="0.00"
   StatCIPS="0.00" StatTAPS="0.00" StatTIPS="0.00" SubType="Other"
   Type="User" cost="0.000000" ctime="1302127058"
   duration="INFINITY" endtime="INFINITY" starttime="1302127058">
     <ACL aff="neutral" cmp="%=" name="jgardner.1" type="RSV"></ACL>
     <ACL cmp="%=" name="jgardner" type="USER"></ACL>
     <ACL cmp="%=" name="company" type="GROUP"></ACL>
     <ACL aff="neutral" cmp="%=" name="jgardner.1" type="RSV"></ACL>
     <History>
       <event state="PROCS=4" time="1302127058"></event>
     </History>
   </rsv>
 </Data>
```


Related topics

- Reservation Allocation Policies
- Reservation Re-Allocation Policies

Administrative Reservations

- Annotating Administrative Reservations
- Using Reservation Profiles
- Optimizing Maintenance Reservations

Administrative reservations behave much like standing reservations but are generally created to address non-periodic, one-time issues. All administrative reservations are created using the mrsvctl -c (or setres) command and are persistent until they expire or are removed using the mrsvctl -r (or releaseres) command.

Annotating Administrative Reservations

Reservations can be labeled and annotated using comments allowing other administrators, local users, portals and other services to obtain more detailed information regarding the reservations. Naming and annotations are configured using the `-n` and `-D` options of the mrsvctl command respectively, as in the following example:

```
> mrsvctl -c -D 'testing infiniband performance' -n nettest -h 'r:agt[15-245]'
```

Using Reservation Profiles

You can set up reservation profiles to avoid manually and repetitively inputting standard reservation attributes. Profiles can specify reservation names, descriptions, ACLs, durations, hostlists, triggers, flags, and other aspects that are commonly used. With a reservation profile defined, a new administrative reservation can be created that uses this profile by specifying the `-P` flag as in the following example.

Example 3-100:

```
RSVPROFILE[mtn1] TRIGGER=AType=exec,Action="/tmp/trigger1.sh",EType=start
RSVPROFILE[mtn1] USERLIST=steve,marym
RSVPROFILE[mtn1] HOSTEXP="r:50-250"
```

```
> mrsvctl -c -P mtn1 -s 12:00:00_10/03 -d 2:00:00
```

Example 3-101: Non-Blocking System Reservations with Scheduler Pause

```
RSVPROFILE[pause] TRIGGER=atype=exec,etype=start,action="/opt/moab/bin/mschedctl -p"
RSVPROFILE[pause] TRIGGER=atype=exec,etype=cancel,action="/opt/moab/bin/mschedctl -r"
RSVPROFILE[pause] TRIGGER=atype=exec,etype=end,action="/opt/moab/bin/mschedctl -r"
```

```
> mrsvctl -c -P pause -s 12:00:00_10/03 -d 2:00:00
```

Optimizing Maintenance Reservations

Any reservation causes some negative impact on cluster performance as it further limits the scheduler's ability to optimize scheduling decisions. You can mitigate this impact by using flexible ACLs and triggers.

In particular, a maintenance reservation can be configured to reduce its effective reservation shadow by allowing overlap with checkpointable/preemptible jobs until the time the reservation becomes active. This can be done using a series of triggers that perform the following actions:

- Modify the reservation to disable preemption access.
- Preempt jobs that may overlap the reservation.
- Cancel any jobs that failed to properly checkpoint and exit.

The following example highlights one possible configuration:

```
RSVPROFILE[adm1] JOBATTRLIST=PREEMPTEE
RSVPROFILE[adm1] DESCRIPTION="regular system maintenance"
RSVPROFILE[adm1] TRIGGER=EType=start,Offset=-
300,AType=internal,Action="rsv:-:modify:acl:jattr-=PREEMPTEE"
RSVPROFILE[adm1] TRIGGER=EType=start,Offset=-240,AType=jobpreempt,Action="checkpoint"
RSVPROFILE[adm1] TRIGGER=EType=start,Offset=-60,AType=jobpreempt,Action="cancel"
```

```
> mrsvctl -c -P adm1 -s 12:00:00_10/03 -d 8:00:00 -h ALL
```

This reservation reserves all nodes in the cluster for a period of eight hours. Five minutes before the reservation starts, the reservation is modified to remove access to new preemptible jobs. Four minutes before the reservation starts, preemptible jobs that overlap the reservation are checkpointed. One minute before the reservation, all remaining jobs that overlap the reservation are canceled.

Reservations can also be used to evacuate virtual machines from a nodelist. To do this, you can configure a reservation profile in the `moab.cfg` file that calls an internal trigger to enable the evacuate VM logic. For example:

```
RSVPROFILE[evacvms]
TRIGGER=EType=start,AType=internal,action=node:$(HOSTLIST):evacvms
```

```
> mrsvctl -c -P evacvms -s 12:00:00_10/03 -d 8:00:00 -h ALL
```

Please note that Moab gives its best effort in evacuating VMs; however, if other reservations and policies prevent Moab from locating an alternate location for the VMs to be migrated to, then no action will occur. Administrators can attach additional triggers to the reservation profile to add evacuation logic where needed.

You can also manually create a reservation that evacuates VMs from a nodelist by using the **EVACVMS** reservation flag. For example:

```
> mrsvctl -c -F EVACVMS -s 12:00:00_10/03 -d 8:00:00 -h ALL
```

Related topics

- Backfill
- Preemption
- mrsvctl command

Standing Reservations

Standing reservations build upon the capabilities of advance reservations to enable a site to enforce advanced usage policies in an efficient manner. Standing reservations provide a superset of the capabilities typically found in a batch queuing system's class or queue architecture. For example, queues can be used to allow only particular types of jobs access to certain compute resources. Also, some batch systems allow these queues to be configured so that they only allow this access during certain times of the day or week. Standing reservations allow these same capabilities but with greater flexibility and efficiency than is typically found in a normal queue management system.

Standing reservations provide a mechanism by which a site can dedicate a particular block of resources for a special use on a regular daily or weekly basis. For example, node X could be dedicated to running jobs only from users in the accounting group every Friday from 4 to 10 p.m. See the Reservation Overview for more information about the use of reservations. The Managing Reservations section provides a detailed explanation of the concepts and steps involved in the creation and configuration of standing reservations.

A standing reservation is a powerful means of doing the following:

- Controlling local credential based access to resources.
- Controlling external peer and grid based access to resources.
- Controlling job responsiveness and turnaround.

Related topics

- SRCFG
- Moab Workload Manager for Grids
- mdiag -s (diagnose standing reservations)

Reservation Policies

- Controlling Priority Reservation Creation
- Managing Resource Failures
- Resource Allocation Policy
- Resource Re-Allocation Policy

Controlling Priority Reservation Creation

In addition to standing and administrative reservations, Moab can also create priority reservations. These reservations are used to allow the benefits of out-of-order execution (such as is available with backfill) without the side effect of job starvation. Starvation can occur in any system where the potential exists for a job to be overlooked by the scheduler for an indefinite period. In the case of backfill, small jobs may continue to run on available resources as they become available while a large job sits in the queue, never able to find enough nodes available simultaneously on which to run.

To avoid such situations, priority reservations are created for high priority jobs that cannot run immediately. When making these reservations, the scheduler determines the earliest time the job could start and then reserves these resources for use by this job at that future time.

Priority Reservation Creation Policy

Organizations have the ability to control how priority reservations are created and maintained. It is possible that one job can be at the top of the priority queue for a time and then get bypassed by another job submitted later. The parameter RESERVATIONPOLICY allows a site to determine how existing reservations should be handled when new reservations are made.

Value	Description
HIGHEST	All jobs that have ever received a priority reservation up to the RESERVATIONDEPTH number will maintain that reservation until they run, even if other jobs later bypass them in priority value. For example, if there are four jobs with priorities of 8, 10,12, and 20. `RESERVATIONPOLICY HIGHEST` `RESERVATIONDEPTH 3` Only jobs 20, 12, and 10 get priority reservations. Later, if a job with priority higher than 20 is submitted into the queue, it will also get a priority reservation along with the jobs listed previously. If four jobs higher than 20 were to be submitted into the queue, only three would get priority reservations, in accordance with the condition set in the RESERVATIONDEPTH policy. With *HIGHEST*, Moab may appear to exceed the RESERVATIONDEPTH if it has already scheduled the maximum number of priority reservations and then users submit jobs with higher priority than those already given a priority reservation. Moab keeps all of the previously-created priority reservations and creates new ones for jobs with higher priority (again up to the quantity specified with RESERVATIONDEPTH). This means that, if your RESERVATIONDEPTH is set to 3, Moab can potentially schedule up to 3 new priority reservations each scheduling iteration, as long as new higher-priority jobs are continually submitted. This behavior ensures that the highest-priority jobs receive attention while the former highest-priority jobs do not lose their priority reservation.
CURRENTHIGHEST	Only the current top <RESERVATIONDEPTH> priority jobs receive reservations. Under this policy, all job reservations are destroyed each iteration when the queue is re-prioritized. The top jobs in the queue are then given new reservations.
NEVER	No priority reservations are made.

Priority Reservation Depth

By default, only the highest priority job receives a priority reservation. However, this behavior is configurable via the RESERVATIONDEPTH policy. Moab's default behavior of only reserving the highest priority job allows backfill to be used in a form known as liberal backfill. Liberal backfill tends to maximize system utilization and minimize overall average job turnaround time. However, it does lead to the potential of some lower priority jobs being indirectly delayed and may lead to greater variance in job turnaround time. The RESERVATIONDEPTH parameter can be set to a very large value, essentially enabling what is called conservative backfill where every job that cannot run is given a reservation. Most sites prefer the liberal backfill approach associated with the default RESERVATIONDEPTH of 1 or else select a slightly higher value. It is important to note that to prevent starvation in conjunction with reservations, monotonically increasing priority factors such as queue time or job XFactor should be enabled. See the Prioritization Overview for more information on priority factors.

Another important consequence of backfill and reservation depth is how they affect job priority. In Moab, all jobs are prioritized. Backfill allows jobs to be run out of order and thus, to some extent, job priority to be ignored. This effect, known as priority dilution, can cause many site policies implemented via Moab prioritization policies to be ineffective. Setting the **RESERVATIONDEPTH** parameter to a higher value gives job priority more teeth at the cost of slightly lower system utilization. This lower utilization results from the constraints of these additional reservations, decreasing the scheduler's freedom and its ability to find additional optimizing schedules. Anecdotal evidence indicates that these utilization losses are fairly minor, rarely exceeding 8%.

It is difficult a priori to know the right setting for the **RESERVATIONDEPTH** parameter. Surveys indicate that the vast majority of sites use the default value of 1. Sites that do modify this value typically set it somewhere in the range of 2 to 10. The following guidelines may be useful in determining if and how to adjust this parameter:

Reasons to Increase RESERVATIONDEPTH

- The estimated job start time information provided by the showstart command is heavily used and the accuracy needs to be increased.
- Priority dilution prevents certain key mission objectives from being fulfilled.
- Users are more interested in knowing when their job will run than in having it run sooner.

Reasons to Decrease RESERVATIONDEPTH

- Scheduling efficiency and job throughput need to be increased.

Assigning Per-QoS Reservation Creation Rules

QoS based reservation depths can be enabled via the RESERVATIONQOSLIST parameter. This parameter allows varying reservation depths to be associated with different sets of job QoSs. For example, the following configuration creates two reservation depth groupings:

```
RESERVATIONDEPTH[0]    8
RESERVATIONQOSLIST[0] highprio,interactive,debug
RESERVATIONDEPTH[1]    2
RESERVATIONQOSLIST[1] batch
```

This example causes that the top 8 jobs belonging to the aggregate group of highprio, interactive, and debug QoS jobs will receive priority reservations. Additionally, the top two batch QoS jobs will also receive priority reservations. Use of this feature allows sites to maintain high throughput for important jobs by guaranteeing that a significant proportion of these jobs progress toward starting through use of the priority reservation.

By default, the following parameters are set inside Moab:

```
RESERVATIONDEPTH[DEFAULT]    1
RESERVATIONQOSLIST[DEFAULT] ALL
```

This allows one job with the highest priority to get a reservation. These values can be overwritten by modifying the DEFAULT policy.

Managing Resource Failures

Moab allows organizations to control how to best respond to a number of real-world issues. Occasionally when a reservation becomes active and a job attempts to start, various resource manager race conditions or corrupt state situations will prevent the job from starting. By default, Moab assumes the resource manager is corrupt, releases the reservation, and attempts to re-create the reservation after a short timeout. However, in the interval between the reservation release and the re-creation timeout, other priority reservations may allocate the newly available resources, reserving them before the original reservation gets an opportunity to reallocate them. Thus, when the original job reservation is re-established, its original resource may be unavailable and the resulting new reservation may be delayed several hours from the earlier start time. The parameter RESERVATIONRETRYTIME allows a site that is experiencing frequent resource manager race conditions and/or corruption situations to tell Moab to hold on to the reserved resource for a period of time in an attempt to allow the resource manager to correct its state.

Resource Allocation Policy

By default, when a standing or administrative reservation is created, Moab allocates nodes in accordance with the specified taskcount, node expression, node constraints, and the MINRESOURCE node allocation policy.

Related topics

- Reservation Overview
- Backfill

Configuring and Managing Reservations

- Reservation Attributes
  - Start/End Time
  - Access Control List (ACL)
  - Selecting Resources
  - Flags
- Configuring and Managing Standing Reservations
  - Standing Reservation Attributes
  - Standing Reservation Overview
  - Specifying Reservation Resources
  - Enforcing Policies Via Multiple Reservations
  - Affinity
  - ACL Modifiers
  - Reservation Ownership
  - Partitions
  - Resource Allocation Behavior
  - Rollback Reservations
  - Modifying Resources with Standing Reservations

- Managing Administrative Reservations

Reservation Attributes

All reservations possess a time frame of activity, an access control list (ACL), and a list of resources to be reserved. Additionally, reservations may also possess a number of extension attributes including epilog/prolog specification, reservation ownership and accountability attributes, and special flags that modify the reservation's behavior.

Start/End Time

All reservations possess a start and an end time that define the reservation's active time. During this active time, the resources within the reservation may only be used as specified by the reservation access control list (ACL). This active time may be specified as either a start/end pair or a start/duration pair. Reservations exist and are visible from the time they are created until the active time ends at which point they are automatically removed.

Access Control List (ACL)

For a reservation to be useful, it must be able to limit who or what can access the resources it has reserved.

> By default a reservation may allocate resources that possess credentials that meet the submitter's ACL. In other words, a user's reservation won't necessarily allocate only free and idle nodes. If a reservation exists that coincides with the submitter's ACL, the nodes under that reservation are also considered for allocation. This is referred to as ACL overlap. To make new reservations allocate *only* free and idle nodes, you must use the NOACLOVERLAP flag.

This is handled by way of an ACL. With reservations, ACLs can be based on credentials, resources requested, or performance metrics. In particular, with a standing reservation, the attributes USERLIST, GROUPLIST, ACCOUNTLIST, CLASSLIST, QOSLIST, JOBATTRLIST, PROCLIMIT, MAXTIME, or TIMELIMIT may be specified. (See Affinity and Modifiers.)

> Reservation access can be adjusted based on a job's requested node features by mapping node feature requests to job attributes as in the following example:

```
NODECFG[DEFAULT]   FEATURES=ia64
NODETOJOBATTRMAP   ia64,ia32
SRCFG[pgs]         JOBATTRLIST=ia32
```

```
> mrsvctl -c -a jattr=gpfs\! -h "r:13-500"
```

Selecting Resources

When specifying which resources to reserve, the administrator has a number of options. These options allow control over how many resources are reserved and where they are reserved. The following reservation attributes allow the administrator to define resources.

Task Description

Moab uses the task concept extensively for its job and reservation management. A task is simply an atomic collection of resources, such as processors, memory, or local disk, which must be found on the same node. For example, if a task requires 4 processors and 2 GB of memory, the scheduler must find all processors AND memory on the same node; it cannot allocate 3 processors and 1 GB on one node and 1 processor and 1 GB of memory on another node to satisfy this task. Tasks constrain how the scheduler must collect resources for use in a standing reservation; however, they do not constrain the way in which the scheduler makes these cumulative resources available to jobs. A job can use the resources covered by an accessible reservation in whatever way it needs. If reservation X allocates 6 tasks with 2 processors and 512 MB of memory each, it could support job Y which requires 10 tasks of 1 processor and 128 MB of memory or job Z which requires 2 tasks of 4 processors and 1 GB of memory each. The task constraints used to acquire a reservation's resources are transparent to a job requesting use of these resources.

Example 3-102:

```
SRCFG[test] RESOURCES=PROCS:2,MEM:1024
```

Taskcount

Using the task description, the taskcount attribute defines how many tasks must be allocated to satisfy the reservation request. To create a reservation, a taskcount and/or a hostlist must be specified.

Example 3-103:

```
SRCFG[test] TASKCOUNT=256
```

Hostlist

A hostlist constrains the set of resources available to a reservation. If no taskcount is specified, the reservation attempts to reserve one task on each of the listed resources. If a taskcount is specified that requests fewer resources than listed in the hostlist, the scheduler reserves only the number of tasks from the hostlist specified by the taskcount attribute. If a taskcount is specified that requests more resources than listed in the hostlist, the scheduler reserves the hostlist nodes first and then seeks additional resources outside of this list.

Example 3-104:

```
SRCFG[test] HOSTLIST=node01,node1[3-5]
```

Node Features

Node features can be specified to constrain which resources are considered.

Example 3-105:

```
SRCFG[test] NODEFEATURES=fastos
```

Partition

A partition may be specified to constrain which resources are considered.

Example 3-106:

```
SRCFG[test] PARTITION=core3
```

Flags

Reservation flags allow specification of special reservation attributes or behaviors. Supported flags are listed in the following table:

Flag Name	Description
ACLOVERLAP	**Deprecated (this is now a default flag).** In addition to free or idle nodes, a reservation may also reserve resources that possess credentials that meet the reservation's ACL. To change this behavior, set the **NOACLOVERLAP** on page 466 flag.
ADVRESJOBDESTROY	All jobs that have an ADVRES matching this reservation are canceled when the reservation is destroyed.
ALLOWJOBOVERLAP	A job is allowed to start in a reservation that may end before the job completes. When the reservation ends before the job completes, the job will not be canceled but will continue to run.
BYNAME	Reservation only allows access to jobs that meet reservation ACLs and explicitly request the resources of this reservation using the job ADVRES flag. (See Job to Reservation Binding.)
DEDICATEDRESOURCE (aka EXCLUSIVE)	Reservation placed only on resources that are not reserved by any other reservation including job, system, and user reservation. There are two exception to this: 1. Reserved resources could be allocated when DEDICATEDRESOURCE is combined with **IGNJOBRSV*** 2. Reserved resources could be allocated when a reservation matches the submitter's ACL. In this case, to make DEDICATEDRESOURCE *truly* exclusive, use the NOACLOVERLAP flag. The order that SRCFG reservations are listed in the configuration is important when using *DEDICATEDRESOURCE*, because reservations made afterwards can steal resources later. During configuration, list *DEDICATEDRESOURCE* reservations last to guarantee exclusiveness.

Flag Name	Description
EVACVMS	Reservation will automatically evacuate virtual machines from the reservation nodelist. The same action can be accomplished by using reservation profiles. For more information, see **Optimizing Maintenance Reservations** on page 456.
IGNIDLEJOBS*	Reservation can be placed on top of idle job reservations. This flag is meant to be used in conjunction with *DEDICATEDRESOURCE*.
IGNJOBRSV*	Ignores existing job reservations, allowing the reservation to be forced onto available resources even if it conflicts with existing job reservations. User and system reservation conflicts are still valid. It functions the same as IGNIDLEJOBS plus allows a reservation to be placed on top of an existing running job's reservation. This flag is meant to be used in conjunction with *DEDICATEDRESOURCE*.
IGNRSV*	Request ignores existing resource reservations allowing the reservation to be forced onto available resources even if this conflicts with other reservations. It functions the same as IGNJOBRSV plus allows the reservation to be placed on top of the system reservations. This flag is meant to be used in conjunction with *DEDICATEDRESOURCE*.
IGNSTATE*	Reservation ignores node state when assigning nodes. It functions the same as IGNRSV plus allows the reservation to be placed on nodes that are not currently available. Also ignores resource availability on nodes. IGNSTATE is specified by default when using a HOSTLIST to define nodes. However, if using a HOSTLIST and a TASKCOUNT, you need to specify IGNSTATE if you want Moab to ignore the node state when assigning nodes to the reservation.

<table>
<tr><th>Flag Name</th><th>Description</th></tr>
<tr><td>NOACLOVERLAP</td><td>All resources must be free or idle, with no existing reservations. Moab will not allocate in-use resources even if they match the reservation's ACL.
<code>mrsvctl -c -t 12 -E -F noacloverlap -a user==john</code>
Moab looks for resources that are exclusive (free). Without the flag, Moab would look for resources that are exclusive or that are already running <code>john</code>'s jobs.
This flag is meant to be used in conjunction with DEDICATEDRESOURCE.</td></tr>
<tr><td>NOVMMIGRATION</td><td>If set on a reservation, this prevents VMs from being migrated away from the reservation. If there are multiple reservations on the hypervisor and at least one reservation does not have the NOVMIGRATION flag, then VMs will be migrated.</td></tr>
<tr><td>OWNERPREEMPT</td><td>Jobs by the reservation owner are allowed to preempt non-owner jobs using reservation resources.</td></tr>
<tr><td>OWNERPREEMPTIGNOREMINTIME</td><td>Allows the OWNERPREEMPT flag to "trump" the PREEMPTMINTIME setting for jobs already running on a reservation when the owner of the reservation submits a job. For example: without the OWNERPREEMPTIGNOREMINTIME flag set, a job submitted by the owner of a reservation will not preempt non-owner jobs already running on the reservation until the PREEMPTMINTIME setting (if set) for those jobs is passed.
With the OWNERPREEMPTIGNOREMINTIME flag set, a job submitted by the owner of a reservation immediately preempts non-owner jobs already running on the reservation, regardless of whether PREEMPTMINTIME is set for the non-owner jobs.</td></tr>
<tr><td>REQFULL</td><td>Reservation is only created when all resources can be allocated.</td></tr>
<tr><td>SINGLEUSE</td><td>Reservation is automatically removed after completion of the first job to use the reserved resources.</td></tr>
<tr><td>SPACEFLEX</td><td>Deprecated (this is now a default flag). Reservation is allowed to adjust resources allocated over time in an attempt to optimize resource utilization.</td></tr>
</table>

> * *IGNIDLEJOBS*, *IGNJOBRSV*, *IGNRSV*, and *IGNSTATE* flags are built on one another and form a hierarchy. *IGNJOBRSV* performs the function of *IGNIDLEJOBS* plus its own functions. *IGNRSV* performs the function of *IGNJOBSRV* and *IGNIDLEJOBS* plus its own functions. *IGNSTATE* performs the function of *IGNRSV*, *IGNJOBRSV*, and *IGNIDLEJOBS* plus its own functions. While you can use combinations of these flags, it is not necessary. If you set one flag, you do not need to set other flags that fall beneath it in the hierarchy.

Most flags can be associated with a reservation via the mrsvctl -c -F command or the SRCFG parameter.

Configuring Standing Reservations

Standing reservations allow resources to be dedicated for particular uses. This dedication can be configured to be permanent or periodic, recurring at a regular time of day and/or time of week. There is extensive applicability of standing reservations for everything from daily dedicated job runs to improved use of resources on weekends. By default, standing reservations can overlap other reservations. Unless you set an ignore-type flag (*ACLOVERLAP*, *DEDICATEDRESOURCE*, *IGNIDLEJOBS*, or *IGNJOBRSV*), they are automatically given the *IGNRSV* flag. All standing reservation attributes are specified via the SRCFG parameter using the attributes listed in the table below.

Standing Reservation Attributes

ACCESS	
Format	*DEDICATED* or *SHARED*
Default	---
Description	If set to *SHARED*, allows a standing reservation to use resources already allocated to other non-job reservations. Otherwise, these other reservations block resource access.
Example	`SRCFG[test] ACCESS=SHARED` *Standing reservation test may access resources allocated to existing standing and administrative reservations.* The order that SRCFG reservations are listed in the configuration are important when using *DEDICATED*, because reservations made afterwards can steal resources later. During configuration, list *DEDICATED* reservations last to guarantee exclusiveness.

ACCOUNTLIST	
Format	List of valid, comma delimited account names (see ACL Modifiers).

ACCOUNTLIST	
Default	---
Description	Specifies that jobs with the associated accounts may use the resources contained within this reservation.
Example	`SRCFG[test] ACCOUNTLIST=ops,staff` *Jobs using the account ops or staff are granted access to the resources in standing reservation test.*

CHARGE	
Format	<BOOLEAN>
Default	---
Description	Overrides the default charging behavior. If set to *True*, indicates that this reservation should be charged, even if no **ChargeAccount** or **ChargeUser** are specified (this assumes your Accounting Manager is set up to permit this). If set to *False*, indicates that this reservation should not be charged. It is not necessary to specify **CHARGE**=*True* if **CHARGEACCOUNT** or **CHARGEUSER** is specified.
Example	`SRCFG[sr_gold1] CHARGE=False` *Prevent charges to this reservation (might be used when `AMCFG[] ALWAYSCHARGERESERVATIONS=True`).*

CHARGEACCOUNT	
Format	Any valid account name.
Default	---
Description	Specifies that idle cycles for this reservation should be charged against the specified account (via the Accounting Manager). **CHARGEACCOUNT** must be used in conjunction with CHARGEUSER.

CHARGEACCOUNT	
Example	`SRCFG[sr_gold1] CHARGEACCOUNT=math` `SRCFG[sr_gold1] CHARGEUSER=john` *Moab charges all idle cycles within reservations supporting standing reservation `sr_gold1` to account math.*

CHARGEUSER	
Format	Any valid username.
Default	---
Description	Specifies that idle cycles for this reservation should be charged against the specified user (via the Accounting Manager). CHARGEUSER must be used in conjunction with CHARGEACCOUNT.
Example	`SRCFG[sr_gold1] CHARGEACCOUNT=math` `SRCFG[sr_gold1] CHARGEUSER=john` *Moab charges all idle cycles within reservations supporting standing reservation `sr_gold1` to user john.*

CLASSLIST	
Format	List of valid, comma delimited classes/queues (see ACL Modifiers).
Default	---
Description	Specifies that jobs with the associated classes/queues may use the resources contained within this reservation.
Example	`SRCFG[test] CLASSLIST=!interactive` *Jobs not using the class interactive are granted access to the resources in standing reservation test.*

CLUSTERLIST	
Format	List of valid, comma-delimited peer clusters (see Moab Workload Manager for Grids).
Default	---
Description	Specifies that jobs originating within the listed clusters may use the resources contained within this reservation.
Example	`SRCFG[test] CLUSTERLIST=orion2,orion7` *Moab grants jobs from the listed peer clusters access to the reserved resources.*

COMMENT	
Format	<STRING> If the string contains whitespace, it should be enclosed in single (') or double quotes (").
Default	---
Description	Specifies a descriptive message associated with the standing reservation and all child reservations.
Example	`SRCFG[test] COMMENT='rsv for network testing'` *Moab annotates the standing reservation test and all child reservations with the specified message. These messages show up within Moab client commands, Moab web tools, and graphical administrator tools.*

DAYS	
Format	One or more of the following (comma-delimited): • *Mon* • *Tue* • *Wed* • *Thu* • *Fri* • *Sat* • *Sun* • *[ALL]*

DAYS	
Default	*[ALL]*
Description	Specifies which days of the week the standing reservation is active.
Example	`SRCFG[test] DAYS=Mon,Tue,Wed,Thu,Fri` *Standing reservation test is active Monday through Friday.*

DEPTH	
Format	<INTEGER>
Default	*2*
Description	Specifies the depth of standing reservations to be created (one per period). To satisfy the **DEPTH**, Moab creates new reservations at the beginning of the specified **PERIOD** on page 476. If your reservation ends at the same time that a new **PERIOD** begins, the number of reservations may not match the requested **DEPTH**. To prevent or resolve this issue, set the **ENDTIME** on page 472 a couple minutes before the beginning of the next **PERIOD**. For example, set the **ENDTIME** to *23:58* instead of *00:00*.
Example	`SRCFG[test] PERIOD=DAY DEPTH=6` *Specifies that six reservations will be created for standing reservation test.*

DISABLE	
Format	<BOOLEAN>
Default	*FALSE*
Description	Specifies that the standing reservation should no longer spawn child reservations.
Example	`SRCFG[test] PERIOD=DAY DEPTH=7 DISABLE=TRUE` *Specifies that reservations are created for standing reservation test for today and the next six days.*

ENDTIME	
Format	[[[DD:]HH:]MM:]SS
Default	*24:00:00*
Description	Specifies the time of day the standing reservation period ends (end of day or end of week depending on **PERIOD**).
Example	`SRCFG[test] STARTTIME=8:00:00` `SRCFG[test] ENDTIME=17:00:00` `SRCFG[test] PERIOD=DAY` *Standing reservation test is active from 8:00 AM until 5:00 PM.*

FLAGS	
Format	Comma-delimited list of zero or more flags listed in the reservation flags overview.
Default	---
Description	Specifies special reservation attributes. See Managing Reservations - Flags for details.
Example	`SRCFG[test] FLAGS=BYNAME,DEDICATEDRESOURCE` *Jobs may only access the resources within this reservation if they explicitly request the reservation by name. Further, the reservation is created to not overlap with other reservations.*

GROUPLIST	
Format	One or more comma-delimited group names.
Default	*[ALL]*
Description	Specifies the groups allowed access to this standing reservation (see ACL Modifiers).
Example	`SRCFG[test] GROUPLIST=staff,ops,special` `SRCFG[test] CLASSLIST=interactive` *Moab allows jobs with the listed group IDs or which request the job class interactive to use the resources covered by the standing reservation.*

HOSTLIST	
Format	One or more comma delimited host names or *host expressions* or the string "class:<classname>".
Default	---
Description	Specifies the set of hosts that the scheduler can search for resources to satisfy the reservation. If specified using the "class:X" format, Moab only selects hosts that support the specified class. If **TASKCOUNT** is also specified, only **TASKCOUNT** tasks are reserved. Otherwise, all matching hosts are reserved. The **HOSTLIST** attribute is treated as host regular expression so `foo10` will map to `foo10`, `foo101`, `foo1006`, and so forth. To request an exact host match, the expression can be bounded by the carat and dollar symbol expression markers as in `^foo10$`.
Example	`SRCFG[test] HOSTLIST=node001,node002,node003` `SRCFG[test] RESOURCES=PROCS:2;MEM:512` `SRCFG[test] TASKCOUNT=2` *Moab reserves a total of two tasks with 2 processors and 512 MB each, using resources located on node001, node002, and/or node003.* `SRCFG[test] HOSTLIST=node01,node1[3-5]` *The reservation will consume all nodes that have "node01" somewhere in their names and all nodes that have both "node1" and either a "3," "4," or "5" in their names.* `SRCFG[test] HOSTLIST=r:node[1-6]` *The reservation will consume all nodes with names that begin with "node" and end with any number 1 through 6. In other words, it will reserve node1, node2, node3, node4, node5, and node6.*

JOBATTRLIST	
Format	Comma-delimited list of one or more of the following job attributes: • *PREEMPTEE* • *INTERACTIVE* • any generic attribute configured through NODECFG.
Default	---

JOBATTRLIST	
Description	Specifies job attributes that grant a job access to the reservation. Values can be specified with a "!="assignment to only allow jobs NOT requesting a certain feature inside the reservation. To enable/disable reservation access based on requested node features, use the parameter NODETOJOBATTRMAP.
Example	`SRCFG[test] JOBATTRLIST=PREEMPTEE` *Preemptible jobs can access the resources reserved within this reservation.*

MAXJOB	
Format	<INTEGER>
Default	---
Description	Specifies the maximum number of jobs that can run in the reservation.
Example	`SRCFG[test] MAXJOB=1` *Only one job will be allowed to run in this reservation.*

MAXTIME	
Format	[[[DD:]HH:]MM:]SS[+]
Default	---
Description	Specifies the maximum time for jobs allowable. Can be used with Affinity to attract jobs with same **MAXTIME**.
Example	`SRCFG[test] MAXTIME=1:00:00+` *Jobs with a time of 1:00:00 are attracted to this reservation.*

NODEFEATURES	
Format	Comma-delimited list of node features.
Default	---
Description	Specifies the required node features for nodes that are part of the standing reservation.
Example	`SRCFG[test] NODEFEATURES=wide,fddi` *All nodes allocated to the standing reservation must have both the wide and fddi node attributes.*

OWNER	
Format	<CREDTYPE>:<CREDID> Where *<CREDTYPE>* is one of *USER*, *GROUP*, *ACCT*, *QoS*, *CLASS* or *CLUSTER* and *<CREDTYPE>* is a valid credential id of that type.
Default	---
Description	Specifies the owner of the reservation. Setting ownership for a reservation grants the user management privileges, including the power to release it. Setting a *USER* as the *OWNER* of a reservation gives that user privileges to query and release the reservation. For sandbox reservations, sandboxes are applied to a specific peer only if *OWNER* is set to *CLUSTER:<PEERNAME>*.
Example	`SRCFG[test] OWNER=ACCT:jupiter` *User jupiter owns the reservation and may be granted special privileges associated with that ownership.*

PARTITION	
Format	Valid partition name.
Default	*[ALL]*

PARTITION	
Description	Specifies the partition in which to create the standing reservation.
Example	`SRCFG[test] PARTITION=OLD` *The standing reservation will only select resources from partition OLD.*

PERIOD	
Format	One of *DAY*, *WEEK*, or *INFINITY*.
Default	*DAY*
Description	Specifies the `period` of the standing reservation.
Example	`SRCFG[test] PERIOD=WEEK` Each standing reservation covers a one week period.

PROCLIMIT	
Format	<QUALIFIER><INTEGER> *<QUALIFIER>* may be one of the following <, <=, ==, >=, >
Default	---
Description	Specifies the processor limit for jobs requesting access to this standing reservation.
Example	`SRCFG[test] PROCLIMIT<=4` *Jobs requesting 4 or fewer processors are allowed to run.*

PSLIMIT	
Format	<QUALIFIER><INTEGER> *<QUALIFIER>* may be one of the following <, <=, ==, >=, >

PSLIMIT	
Default	---
Description	Specifies the processor-second limit for jobs requesting access to this standing reservation.
Example	`SRCFG[test] PSLIMIT<=40000` *Jobs requesting 40000 or fewer processor-seconds are allowed to run.*

QOSLIST	
Format	Zero or more valid, comma-delimited QoS names.
Default	---
Description	Specifies that jobs with the listed QoS names can access the reserved resources.
Example	`SRCFG[test] QOSLIST=hi,low,special` *Moab allows jobs using the listed QOS's access to the reserved resources.*

REQUIREDTPN	
Format	<QUALIFIER><INTEGER> <*QUALIFIER*> may be one of the following <, <=, ==, >=, >
Default	---
Description	Restricts access to reservations based on the job's TPN (tasks per node).
Example	`SRCFG[test] REQUIREDTPN==4` *Jobs with tpn=4 or ppn=4 would be allowed within the reservation, but any other TPN value would not. (For more information, see* **TPN (Exact Tasks Per Node)** on page 480*.)*

RESOURCES

Format	Semicolon delimited <ATTR>:<VALUE> pairs where <*ATTR*> may be one of *PROCS*, *MEM*, *SWAP*, or *DISK*.
Default	*PROCS:-1* (All processors available on node)
Description	Specifies what resources constitute a single standing reservation task. (Each task must be able to obtain all of its resources as an atomic unit on a single node.) Supported resources currently include the following: • *PROCS* (number of processors) • *MEM* (real memory in MB) • *DISK* (local disk in MB) • *SWAP* (virtual memory in MB)
Example	`SRCFG[test] RESOURCES=PROCS:1;MEM:512` *Each standing reservation task reserves one processor and 512 MB of real memory.*

ROLLBACKOFFSET

Format	[[[DD:]HH:]MM:]SS
Default	---
Description	Specifies the minimum time in the future at which the reservation may start. This offset is rollback meaning the start time of the reservation will continuously roll back into the future to maintain this offset. Rollback offsets are a good way of providing guaranteed resource access to users under the conditions that they must commit their resources in the future or lose dedicated access. See QoS for more info about quality of service and service level agreements; also see Rollback Reservation Overview. Neither credlock nor advres is compatible on the jobs submitted for this reservation.
Example	`SRCFG[ajax] ROLLBACKOFFSET=24:00:00 TASKCOUNT=32` `SRCFG[ajax] PERIOD=INFINITY ACCOUNTLIST=ajax` *The standing reservation guarantees access to up to 32 processors within 24 hours to jobs from the ajax account.* Adding an asterisk to the **ROLLBACKOFFSET** value pins rollback reservation start times when an idle reservation is created in the rollback reservation. For example: `SRCFG[staff] ROLLBACKOFFSET=18:00:00* PERIOD=INFINITY`

RSVACCESSLIST	
Format	<RESERVATION>[,...]
Default	---
Description	A list of reservations to which the specified reservation has access.
Example	`SRCFG[test] RSVACCESSLIST=rsv1,rsv2,rsv3`

RSVGROUP	
Format	<STRING>
Default	---
Description	See section Reservation Group for a detailed description.
Example	`SRCFG[test] RSVGROUP=rsvgrp1` `SRCFG[ajax] RSVGROUP=rsvgrp1`

STARTTIME	
Format	[[[DD:]HH:]MM:]SS
Default	*00:00:00:00* (midnight)
Description	Specifies the time of day/week the standing reservation becomes active. Whether this indicates a time of day or time of week depends on the setting of the **PERIOD** attribute. If specified within a reservation profile, a value of 0 indicates the reservation should start at the earliest opportunity.
Example	`SRCFG[test] STARTTIME=08:00:00` `SRCFG[test] ENDTIME=17:00:00` `SRCFG[test] PERIOD=DAY` *The standing reservation will be active from 8:00 a.m. until 5:00 p.m. each day.*

TASKCOUNT	
Format	<INTEGER>
Default	*0* (unlimited tasks)
Description	Specifies how many tasks should be reserved for the reservation.
Example	`SRCFG[test] RESOURCES=PROCS:1;MEM:256` `SRCFG[test] TASKCOUNT=16` *Standing reservation test reserves 16 tasks worth of resources; in this case, 16 processors and 4 GB of real memory.*

TIMELIMIT	
Format	[[[DD:]HH:]MM:]SS
Default	*-1* (no time based access)
Description	Specifies the maximum allowed overlap between the standing reservation and a job requesting resource access.
Example	`SRCFG[test] TIMELIMIT=1:00:00` *Moab allows jobs to access up to one hour of resources in the standing reservation.*

TPN (Exact Tasks Per Node)	
Format	<INTEGER>
Default	*0* (no TPN constraint)
Description	Specifies the exact number of tasks per node that must be available on eligible nodes.
Example	`SRCFG[2] TPN=4` `SRCFG[2] RESOURCES=PROCS:2;MEM:256` *Moab must locate four tasks on each node that is to be part of the reservation. That is, each node included in standing reservation 2 must have 8 processors and 1 GB of memory available.*

TRIGGER	
Format	See **Creating a trigger** on page 727 for syntax.
Default	N/A
Description	Specifies event triggers to be launched by the scheduler under the scheduler's ID. These triggers can be used to conditionally cancel reservations, modify resources, or launch various actions at specified event offsets. See **About object triggers** on page 724 for more detail.
Example	`SRCFG[fast] TRIGGER=EType=start,Offset=5:00:00,AType=exec,Action="/usr/local/domail.pl"` *Moab launches the `domail.pl` script 5 hours after any fast reservation starts.*

USERLIST	
Format	Comma-delimited list of users.
Default	---
Description	Specifies which users have access to the resources reserved by this reservation (see ACL Modifiers).
Example	`SRCFG[test] USERLIST=bob,joe,mary` *Users bob, joe and mary can all access the resources reserved within this reservation.*

Standing Reservation Overview

A standing reservation is similar to a normal administrative reservation in that it also places an access control list on a specified set of resources. Resources are specified on a per-task basis and currently include processors, local disk, real memory, and swap. The access control list supported for standing reservations includes users, groups, accounts, job classes, and QoS levels. Standing reservations can be configured to be permanent or periodic on a daily or weekly basis and can accept a daily or weekly start and end time. Regardless of whether permanent or recurring on a daily or weekly basis, standing reservations are enforced using a series of reservations, extending a number of periods into the future as controlled by the **DEPTH** attribute of the SRCFG parameter.

The following examples demonstrate possible configurations specified with the **SRCFG** parameter.

Example 3-107: Basic Business Hour Standing Reservation

```
SRCFG[interactive] TASKCOUNT=6 RESOURCES=PROCS:1,MEM:512
SRCFG[interactive] PERIOD=DAY DAYS=MON,TUE,WED,THU,FRI
SRCFG[interactive] STARTTIME=9:00:00 ENDTIME=17:00:00
SRCFG[interactive] CLASSLIST=interactive
```

When using the SRCFG parameter, attribute lists must be delimited using the comma (,), pipe (|), or colon (:) characters; they cannot be space delimited. For example, to specify a multi-class ACL, specify:

```
SRCFG[test] CLASSLIST=classA,classB
```

Only one **STARTTIME** and one **ENDTIME** value can be specified per reservation. If varied start and end times are desired throughout the week, complementary standing reservations should be created. For example, to establish a reservation from 8:00 p.m. until 6:00 a.m. the next day during business days, two reservations should be created-one from 8:00 p.m. until midnight, and the other from midnight until 6:00 a.m. Jobs can run across reservation boundaries allowing these two reservations to function as a single reservation that spans the night. The following example demonstrates how to span a reservation across 2 days on the same nodes:

```
SRCFG[Sun] PERIOD=WEEK
SRCFG[Sun] STARTTIME=00:20:00:00 ENDTIME=01:00:00:00
SRCFG[Sun] HOSTLIST=node01,node02,node03

SRCFG[Mon] PERIOD=WEEK
SRCFG[Mon] STARTTIME=01:00:00:00 ENDTIME=01:06:00:00
SRCFG[Sun] HOSTLIST=node01,node02,node03
```

The preceding example fully specifies a reservation including the quantity of resources requested using the **TASKCOUNT** and **RESOURCES** attributes. In all cases, resources are allocated to a reservation in units called tasks where a task is a collection of resources that must be allocated together on a single node. The **TASKCOUNT** attribute specifies the number of these tasks that should be reserved by the reservation. In conjunction with this attribute, the **RESOURCES** attribute defines the reservation task by indicating what resources must be included in each task. In this case, the scheduler must locate and reserve 1 processor and 512 MB of memory together on the same node for each task requested.

As mentioned previously, a standing reservation reserves resources over a given time frame. The **PERIOD** attribute may be set to a value of *DAY*, *WEEK*, or *INFINITY* to indicate the period over which this reservation should recur. If not specified, a standing reservation recurs on a daily basis. If a standing reservation is configured to recur daily, the attribute **DAYS** may be specified to indicate which days of the week the reservation should exist. This attribute takes a comma-delimited list of days where each day is specified as the first three letters of the day in all capital letters: *MON* or *FRI*. The preceding example specifies that this reservation is periodic on a daily basis and should only exist on business days.

The time of day during which the requested tasks are to be reserved is specified using the **STARTTIME** and **ENDTIME** attributes. These attributes are specified in standard military time HH:MM:SS format and both **STARTTIME** and **ENDTIME** specification is optional defaulting to midnight at the beginning and end of the day respectively. In the preceding example, resources are reserved from 9:00 a.m. until 5:00 p.m. on business days.

The final aspect of any reservation is the access control list indicating who or what can use the reserved resources. In the preceding example, the **CLASSLIST** attribute is used to indicate that jobs requesting the class "interactive" should be allowed to use this reservation.

Specifying Reservation Resources

In most cases, only a small subset of standing reservation attributes must be specified in any given case. For example, by default, **RESOURCES** is set to `PROCS=-1` which indicates that each task should reserve all of the processors on the node on which it is located. This, in essence, creates a one task equals one node mapping. In many cases, particularly on uniprocessor systems, this default behavior may be easiest to work with. However, in SMP environments, the **RESOURCES** attribute provides a powerful means of specifying an exact, multi-dimensional resource set.

> An examination of the parameters documentation shows that the default value of **PERIOD** is *DAYS*. Thus, specifying this parameter in the preceding above was unnecessary. It was used only to introduce this parameter and indicate that other options exist beyond daily standing reservations.

Example 3-108: Host Constrained Standing Reservation

Although the first example did specify a quantity of resources to reserve, it did not specify where the needed tasks were to be located. If this information is not specified, Moab attempts to locate the needed resources anywhere it can find them. The Example 1 reservation essentially discovers hosts where the needed resources can be found. If the **SPACEFLEX** reservation flag is set, then the reservation continues to float to the best hosts over the life of the reservation. Otherwise, it will be locked to the initial set of allocated hosts.

If a site wanted to constrain a reservation to a subset of available resources, this could be accomplished using the **HOSTLIST** attribute. The **HOSTLIST** attribute is specified as a comma-separated list of hostnames and constrains the scheduler to only select tasks from the specified list. This attribute can exactly specify hosts or specify them using host regular expressions. The following example demonstrates a possible use of the **HOSTLIST** attribute:

```
SRCFG[interactive] DAYS=MON,TUE,WED,THU,FRI
SRCFG[interactive] PERIOD=DAY
SRCFG[interactive] STARTTIME=10:00:00 ENDTIME=15:00:00
SRCFG[interactive] RESOURCES=PROCS:2,MEM:256
SRCFG[interactive] HOSTLIST=node001,node002,node005,node020
SRCFG[interactive] TASKCOUNT=6
SRCFG[interactive] CLASSLIST=interactive
```

*Note that the **HOSTLIST** attribute specifies a non-contiguous list of hosts. Any combination of hosts may be specified and hosts may be specified in any order. In this example, the **TASKCOUNT** attribute is also specified. These two attributes both apply constraints on the scheduler with **HOSTLIST** specifying where the tasks can be located and **TASKCOUNT** indicating how many total tasks may be allocated. In this example, six tasks are requested but only four hosts are specified. To handle this, if adequate resources are available, the scheduler may attempt to allocate more than one task per host. For example, assume that each host is a quad-processor system with 1 GB of memory. In such a case, the scheduler could allocate up to two tasks per host and even satisfy the **TASKCOUNT** constraint without using all of the hosts in the hostlist.*

> It is important to note that even if there is a one to one mapping between the value of **TASKCOUNT** and the number of hosts in **HOSTLIST**, the scheduler will not necessarily place one task on each host. If, for example, node001 and node002 were 8 processor SMP hosts with 1 GB of memory, the scheduler could locate up to four tasks on each of these hosts fully satisfying the reservation taskcount without even partially using the remaining hosts. (Moab will place tasks on hosts according to the policy specified with the NODEALLOCATIONPOLICY parameter.) If the hostlist provides more resources than what is required by the reservation as specified via **TASKCOUNT**, the scheduler will simply select the needed resources within the set of hosts listed.

Enforcing Policies Via Multiple Reservations

Single reservations enable multiple capabilities. Combinations of reservations can further extend a site's capabilities to impose specific policies.

Example 3-109: Reservation Stacking

If **HOSTLIST** is specified but **TASKCOUNT** is not, the scheduler will pack as many tasks as possible onto all of the listed hosts. For example, assume the site added a second standing reservation named *debug* to its configuration that reserved resources for use by certain members of its staff using the following configuration:

```
SRCFG[interactive] DAYS=MON,TUE,WED,THU,FRI
SRCFG[interactive] PERIOD=DAY
SRCFG[interactive] STARTTIME=10:00:00 ENDTIME=15:00:00
SRCFG[interactive] RESOURCES=PROCS:2,MEM:256
SRCFG[interactive] HOSTLIST=node001,node002,node005,node020
SRCFG[interactive] TASKCOUNT=6
SRCFG[interactive] CLASSLIST=interactive
SRCFG[debug]       HOSTLIST=node001,node002,node003,node004
SRCFG[debug]       USERLIST=helpdesk
SRCFG[debug]       GROUPLIST=operations,sysadmin
SRCFG[debug]       PERIOD=INFINITY
```

The new standing reservation is quite simple. Since **RESOURCES** is not specified, it will allocate all processors on each host that is allocated. Since **TASKCOUNT** is not specified, it will allocate every host listed in **HOSTLIST**. Since **PERIOD** is set to *INFINITY*, the reservation is always in force and there is no need to specify **STARTTIME**, **ENDTIME**, or **DAYS**.

The standing reservation has two access parameters set using the attributes USERLIST and **GROUPLIST**. This configuration indicates that the reservation can be accessed if any one of the access lists specified is satisfied by the job. In essence, reservation access is logically OR'd allowing access if the requester meets any of the access constraints specified. In this example, jobs submitted by either user *helpdesk* or any member of the groups *operations* or *sysadmin* can use the reserved resources (See ACL Modifiers).

Unless ACL Modifiers are specified, access is granted to the logical *OR* of access lists specified within a standing reservation and granted to the logical *AND* of access lists across different standing reservations. A comparison of the standing reservations *interactive* and *debug* in the preceding example indicates that they both can allocate hosts *node001* and *node002*. If *node001* had both of these reservations in place simultaneously and a job attempted to access this host during business hours when standing reservation *interactive* was active. The job could only use the *doubly* reserved resources if it requests the run class *interactive* and it meets the constraints of reservation *debug*—that is, that it is submitted by user *helpdesk* or by a member of the group *operations* or *sysadmin*.

As a rule, the scheduler does not stack reservations unless it must. If adequate resources exist, it can allocate reserved resources side by side in a single SMP host rather than on top of each other. In the case of a 16 processor SMP host with two 8 processor standing reservations, 8 of the processors on this host will be allocated to the first reservation, and 8 to the next. Any configuration is possible. The 16 processor hosts can also have 4 processors reserved for user "John," 10 processors reserved for group "Staff," with the remaining 2 processors available for use by any job.

Stacking reservations is not usually required but some site administrators choose to do it to enforce elaborate policies. There is no problem with doing so as long as you can keep things straight. It really is not too difficult a concept; it just takes a little getting used to. See the Reservation Overview section for a more detailed description of reservation use and constraints.

As mentioned earlier, by default the scheduler enforces standing reservations by creating a number of reservations where the number created is controlled by the **DEPTH** attribute. Each night at midnight, the scheduler updates its periodic non-floating standing reservations. By default, **DEPTH** is set to 2, meaning when the scheduler starts up, it will create two 24-hour reservations covering a total of two days' worth of time-a reservation for today and one for tomorrow. For daily reservations, at midnight, the reservations roll, meaning today's reservation expires and is removed, tomorrow's reservation becomes today's, and the scheduler creates a new reservation for the next day.

With this model, the scheduler continues creating new reservations in the future as time moves forward. Each day, the needed resources are always reserved. At first, all appears automatic but the standing reservation **DEPTH** attribute is in fact an important aspect of reservation rollback, which helps address certain site specific environmental factors. This attribute remedies a situation that might occur when a job is submitted and cannot run immediately because the system is backlogged with jobs. In such a case, available resources may not exist for several days out and the scheduler must reserve these future resources for this job. With the default **DEPTH** setting of two, when midnight arrives, the scheduler attempts to roll its standing reservations but a problem arises in that the job has now allocated the resources needed for the standing reservation two days out. Moab cannot reserve the resources for the standing reservation because they are already claimed by the job. The standing reservation reserves what it can but because all needed resources are not available, the resulting reservation is now smaller than it should be, or is possibly even empty.

If a standing reservation is smaller than it should be, the scheduler will attempt to add resources each iteration until it is fully populated. However, in the case of this job, the job is not going to release its reserved resources until it completes and the standing reservation cannot claim them until this time. The **DEPTH** attribute allows a site to specify how deep into the future a standing reservation should reserve its resources allowing it to claim the resources first and prevent this problem. If a partial standing reservation is detected on a system, it may be an indication that the reservation's **DEPTH** attribute should be increased.

In Example 3, the **PERIOD** attribute is set to *INFINITY*. With this setting, a single, permanent standing reservation is created and the issues of resource contention do not exist. While this eliminates the contention issue, infinite length standing reservations cannot be made periodic.

Example 3-110: Multiple ACL Types

In most cases, access lists within a reservation are logically OR'd together to determine reservation access. However, exceptions to this rule can be specified by using the required ACL marker-the asterisk (*). Any ACL marked with this symbol is required and a job is only allowed to use a reservation if it meets all required ACLs and at least one non-required ACL (if specified). A common use for this facility is in conjunction with the **TIMELIMIT** attribute. This attribute controls the length of time a job may use the resources within a standing reservation. This access mechanism can be AND'd or OR'd to the cumulative set of all other access lists as specified by the required ACL marker. Consider the following example configuration:

```
SRCFG[special] TASKCOUNT=32
SRCFG[special] PERIOD=WEEK
SRCFG[special] STARTTIME=1:08:00:00
SRCFG[special] ENDTIME=5:17:00:00
SRCFG[special] NODEFEATURES=largememory
SRCFG[special] TIMELIMIT=1:00:00*
SRCFG[special] QOSLIST=high,low,special-
SRCFG[special] ACCCOUNTLIST=!projectX,!projectY
```

The above configuration requests 32 tasks which translate to 32 nodes. The **PERIOD** attribute makes this reservation periodic on a weekly basis while the attributes **STARTTIME** and **ENDTIME** specify the week offsets when this reservation is to start and end (Note that the specification format has changed to DD:HH:MM:SS.). In this case, the reservation starts on Monday at 8:00 a.m. and runs until Friday at 5:00 p.m. The reservation is enforced as a series of weekly reservations that only cover the specified time frame. The **NODEFEATURES** attribute indicates that each of the reserved nodes must have the node feature "largememory" configured.

As described earlier, **TIMELIMIT** indicates that jobs using this reservation can only use it for one hour. This means the job and the reservation can only overlap for one hour. Clearly jobs requiring an hour or less of wallclock time meet this constraint. However, a four-hour job that starts on Monday at 5:00 a.m. or a 12-hour job that starts on Friday at 4:00 p.m. also satisfies this constraint. Also, note the **TIMELIMIT** required ACL marker, *; it is set indicating that jobs must not only meet the **TIMELIMIT** access constraint but must also meet one or more of the other access constraints. In this example, the job can use this reservation if it can use the access specified via **QOSLIST** or **ACCOUNTLIST**; that is, it is assigned a QoS of *high*, *low*, or *special* , or the submitter of the job has an account that satisfies the *!projectX* and *!projectY* criteria. See the QoS Overview for more info about QoS configuration and usage.

Affinity

Reservation ACLs allow or deny access to reserved resources but they may be configured to also impact a job's affinity for a particular reservation. By default, jobs gravitate toward reservations through a mechanism known as positive affinity. This mechanism allows jobs to run on the most constrained resources leaving other, unreserved resources free for use by other jobs that may not be able to access the reserved resources. Normally this is a desired behavior. However, sometimes, it is desirable to reserve resources for use only as a last resort-using the reserved resources only when there are no other resources available. This last resort behavior is known as negative affinity. Note the '-' (hyphen or negative sign) following the *special* in the **QOSLIST** values. This special mark indicates that QoS *special* should be granted access to this reservation but should be assigned negative affinity. Thus, the **QOSLIST** attribute specifies that QoS *high* and *low* should be granted access with positive affinity (use the reservation first where possible) and QoS *special* granted access with negative affinity (use the reservation only when no other resources are available).

Affinity status is granted on a per access object basis rather than a per access list basis and always defaults to positive affinity. In addition to negative affinity, neutral affinity can also be specified using the equal sign (=) as in `QOSLIST[0] normal= high debug= low-`.

When a job matches multiple ACLs for a reservation, the final node affinity for the node, job, and reservation combination is based on the last matching ACL entry found in the configuration file.

For example, given the following reservation ACLs, a job matching both will receive a negative affinity:

```
SRCFG[res1] USERLIST=joe+ MAXTIME<=4:00:00-
```

With the following reservation ACLs, a job matching both will receive a positive affinity:

```
SRCFG[res1] MAXTIME<=4:00:00- USERLIST=joe+
```

ACL Modifiers

ACL modifiers allow a site to change the default behavior of ACL processing. By default, a reservation can be accessed if one or more of its ACLs can be met by the requestor. This behavior can be changed using the "deny" or "required" ACL modifier, as in the following tables:

Not	
Symbol:	! (exclamation point)
Description	If attribute is met, the requestor is denied access regardless of any other satisfied ACLs.
Example	`SRCFG[test] GROUPLIST=staff USERLIST=!steve` *Allow access to all staff members other than steve.*

Required	
Symbol:	* (asterisk)
Description	All required ACLs must be satisfied for requestor access to be granted.
Example	`SRCFG[test] QOSLIST=*high MAXTIME=*2:00:00` *Only jobs in QoS high that request less than 2 hours of walltime are granted access.*

XOR	
Symbol:	^ (carat)
Description	All attributes of the type specified other than the ones listed in the ACL satisfy the ACL.

XOR	
Example	`SRCFG[test] QOSLIST=^high` *All jobs other than those requesting QoS high are granted access.*

CredLock	
Symbol:	& (ampersand)
Description	Matching jobs will be required to run on the resources reserved by this reservation. You can use this modifier on accounts, classes, groups, qualities of service, and users.
Example	`SRCFG[test] USERLIST=&john` *All of user john's jobs must run in this reservation.*

HPEnable (hard policy enable)	
Symbol:	~ (tilde)
Description	ACLs marked with this modifier are ignored during soft policy scheduling and are only considered for hard policy scheduling once all eligible soft policy jobs start.
Example	`SRCFG[johnspace] USERLIST=john CLASSLIST=~debug` *All of user john's jobs are allowed to run in the reservation at any time. Debug jobs are also allowed to run in this reservation but are only considered after all of John's jobs are given an opportunity to start. User john's jobs are considered before debug jobs regardless of job priority.* If HPEnable and Not markers are used in conjunction, then specified credentials are *blocked-out* of the reservation during soft-policy scheduling.

Note the **ACCOUNTLIST** values in Example 3-110 are preceded with an exclamation point, or NOT symbol. This indicates that all jobs with accounts other than *projectX* and *projectY* meet the account ACL. Note that if a !<X> value (!projectX) appears in an ACL line, that ACL is satisfied by any object not explicitly listed by a NOT entry. Also, if an object matches a NOT entry, the associated job is excluded from the reservation even if it meets other ACL requirements. For example, a QoS 3 job requesting account *projectX* is denied access to the reservation even though the job QoS matches the QoS ACL.

Example 3-111: Binding Users to Reservations at Reservation Creation

```
# create a 4 node reservation for john and bind all of john's jobs to that reservation
> mrsvctl -c -a user=&john -t 4
```

Reservation Ownership

Reservation ownership allows a site to control who owns the reserved resources during the reservation time frame. Depending on needs, this ownership may be identical to, a subset of, or completely distinct from the reservation ACL. By default, reservation ownership implies resource accountability and resources not consumed by jobs are accounted against the reservation owner. In addition, ownership can also be associated with special privileges within the reservation.

Ownership is specified using the **OWNER** attribute in the format `<CREDTYPE>:<CREDID>`, as in **OWNER**=*USER:john*. To enable *john*'s jobs to preempt other jobs using resources within the reservation, the **SRCFG** attribute **FLAG** should be set to OWNERPREEMPT. In the example below, the *jupiter* project chooses to share resources with the *saturn* project but only when it does not currently need them.

Example 3-112: Limited Shared Access

```
ACCOUNTCFG[jupiter] PRIORITY=10000
SRCFG[jupiter] HOSTLIST=node0[1-9]
SRCFG[jupiter] PERIOD=INFINITY
SRCFG[jupiter] ACCOUNTLIST=jupiter,saturn-
SRCFG[jupiter] OWNER=ACCT:jupiter
SRCFG[jupiter] FLAGS=OWNERPREEMPT
```

Partitions

A reservation can be used in conjunction with a partition. Configuring a standing reservation on a partition allows constraints to be (indirectly) applied to a partition.

Example 3-113: Time Constraints by Partition

The following example places a 3-day wall-clock limit on two partitions and a 64 processor-hour limit on jobs running on partition *small*.

```
SRCFG[smallrsv] PARTITION=small MAXTIME=3:00:00:00 PSLIMIT<=230400 HOSTLIST=ALL
SRCFG[bigrsv] PARTITION=big MAXTIME=3:00:00:00 HOSTLIST=ALL
```

Resource Allocation Behavior

As mentioned, standing reservations can operate in one of two modes, floating, or non-floating (essentially node-locked). A floating reservation is created when the flag **SPACEFLEX** is specified. If a reservation is non-floating, the scheduler allocates all resources specified by the **HOSTLIST** parameter regardless of node state, job load, or even the presence of other standing reservations. Moab interprets the request for a non-floating reservation as, "I want a reservation on these exact nodes, no matter what!"

If a reservation is configured to be floating, the scheduler takes a more relaxed stand, searching through all possible nodes to find resources meeting standing reservation constraints. Only Idle, Running, or Busy nodes are considered and further, only considered if no reservation conflict is detected. The reservation attribute **ACCESS** modifies this behavior slightly and allows the reservation to allocate resources even if reservation conflicts exist.

> If a **TASKCOUNT** is specified with or without a **HOSTEXPRESSION**, Moab will, by default, only consider "up" nodes for allocation. To change this behavior, the reservation flag IGNSTATE can be specified as in the following example:
>
> ```
> SRCFG[nettest] GROUPLIST=sysadm
> SRCFG[nettest] FLAGS=IGNSTATE
> SRCFG[nettest] HOSTLIST=node1[3-8]
> SRCFG[nettest] STARTTIME=9:00:00
> SRCFG[nettest] ENDTIME=17:00:00
> ```

> Access to existing reservations can be controlled using the reservation flag IGNRSV.

Other standing reservation attributes not covered here include **PARTITION** and **CHARGEACCOUNT**. These parameters are described in some detail in the parameters documentation.

Example 3-114: Using Reservations to Guarantee Turnover

In some cases, it is desirable to make certain a portion of a cluster's resources are available within a specific time frame. The following example creates a floating reservation belonging to the *jupiter* account that guarantees 16 tasks for use by jobs requesting up to one hour.

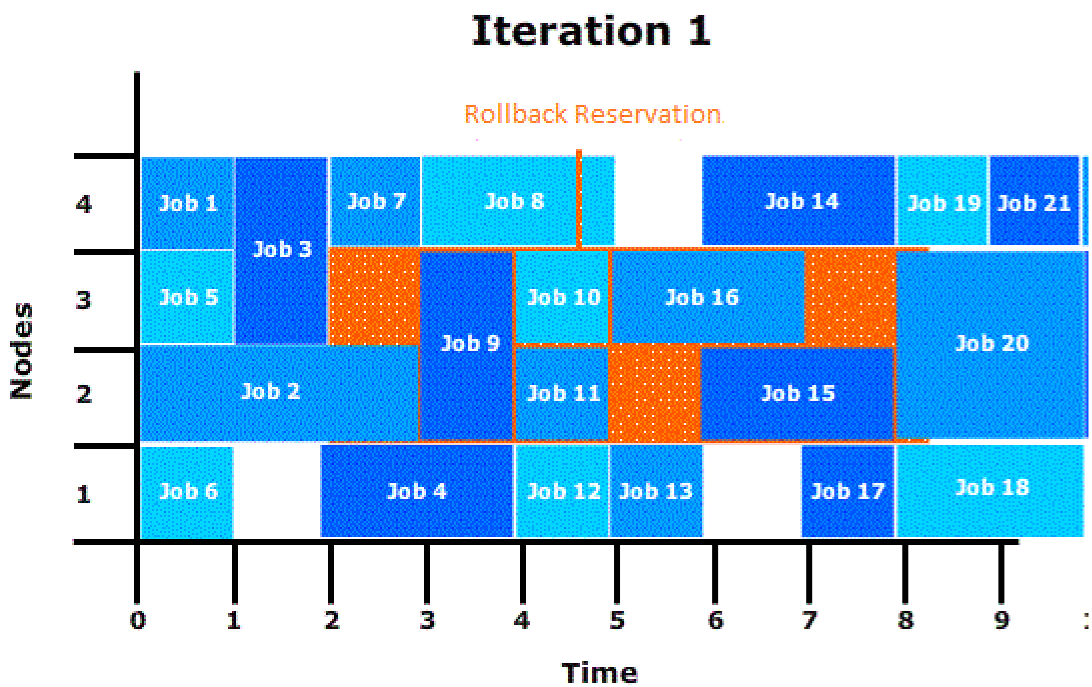
```
SRCFG[shortpool] OWNER=ACCT:jupiter
SRCFG[shortpool] FLAGS=SPACEFLEX
SRCFG[shortpool] MAXTIME=1:00:00
SRCFG[shortpool] TASKCOUNT=16
SRCFG[shortpool] STARTTIME=9:00:00
SRCFG[shortpool] ENDTIME=17:00:00
SRCFG[shortpool] DAYS=Mon,Tue,Wed,Thu,Fri
```

This reservation enables a capability similar to what was known in early Maui releases as "shortpool." The reservation covers every weekday from 9:00 a.m. to 5:00 p.m., reserving 16 tasks and allowing jobs to overlap the reservation for up to one hour. The *SPACEFLEX* flag indicates that the reservation may be dynamically modified--over time to re-locate to more optimal resources. In the case of a reservation with the **MAXTIME** ACL, this would include migrating to resources that are in use but that free up within the **MAXTIME** time frame. Additionally, because the **MAXTIME** ACL defaults to positive affinity, any jobs that fit the ACL attempt to use available reserved resources first before looking elsewhere.

Rollback Reservations

Rollback reservations are enabled using the ROLLBACKOFFSET attribute and can be used to allow users guaranteed access to resources, but the guaranteed access is limited to a time-window in the future. This functionality forces users to commit their resources in the future or lose access.

Image 3-5: Rollback reservation over 3 iterations



Example 3-115: Rollback Reservations

```
SRCFG[ajax] ROLLBACKOFFSET=24:00:00 TASKCOUNT=32
SRCFG[ajax] PERIOD=INFINITY ACCOUNTLIST=ajax
```

Adding an asterisk to the **ROLLBACKOFFSET** value pins rollback reservation start times when an idle reservation is created in the rollback reservation. For example: `SRCFG[staff] ROLLBACKOFFSET=18:00:00* PERIOD=INFINITY`.

Modifying Resources with Standing Reservations

Moab can customize compute resources associated with a reservation during the life of the reservation. This can be done generally using the TRIGGER attribute, or it can be done for operating systems using the shortcut attribute OS. If set, Moab dynamically reprovisions allocated reservation nodes to the requested operating system as shown in the following example:

```
SRCFG[provision] PERIOD=DAY DAY=MON,WED,FRI STARTTIME=7:00:00 ENDTIME=10:00:00
SRCFG[provision] OS=rhel4  # provision nodes to use redhat during reservation, restore
when done
```

Managing Administrative Reservations

A default reservation with no ACL is termed an *administrative* reservation, but is occasionally referred to as a *system* reservation. It blocks access to all jobs because it possesses an empty access control list. It is often useful when performing administrative tasks but cannot be used for enforcing resource usage policies.

Administrative reservations are created and managed using the mrsvctl command. With this command, all aspects of reservation time frame, resource selection, and access control can be dynamically modified. The mdiag -r command can be used to view configuration, state, allocated resource information as well as identify any potential problems with the reservation. The following table briefly summarizes commands used for common actions. More detailed information is available in the command summaries.

Action	Command
create reservation	mrsvctl -c <RSV_DESCRIPTION>
list reservations	mrsvctl -l
release reservation	mrsvctl -r <RSVID>
modify reservation	mrsvctl -m <ATTR>=<VAL> <RSVID>
query reservation configuration	mdiag -r <RSVID>
display reservation hostlist	mrsvctl -q resources <RSVID>

Related topics

- SRCFG (configure standing reservations)
- RSVPROFILE (create reservation profiles)

Personal Reservations

- Enabling Personal Reservation Management
- Reservation Accountability and Defaults
  - Reservation Allocation and Charging
  - Setting Reservation Default Attributes
- Reservation Limits
- Reservation and Job Binding
  - Constraining a job to only run in a particular reservation
  - Constraining a Reservation to Only Accept Certain Jobs

By default, advance reservations are only available to scheduler administrators. While administrators may create and manage reservations to provide resource access to end-users, end-users cannot create, modify, or destroy these reservations. Moab extends the ability to manage reservations to end-users and provides control facilities to keep these features manageable. Reservations created by end-users are called personal reservations or user reservations.

Enabling Personal Reservation Management

User, or personal, reservations can be enabled on a per QoS basis by setting the ENABLEUSERRSV flag as in the following example:

```
QOSCFG[titan]    QFLAGS=ENABLEUSERRSV # allow 'titan' QOS jobs to create user
reservations
USERCFG[DEFAULT] QDEF=titan           # allow all users to access 'titan' QOS
...
```

If set, end-users are allowed to create, modify, cancel, and query reservations they own. As with jobs, users may associate a personal reservation with any QoS or account to which they have access. This is accomplished by specifying per reservation accountable credentials as in the following example:

```
> mrsvctl -c -S AQOS=titan -h node01 -d 1:00:00 -s 1:30:00
Note:  reservation test.126 created
```

As in the preceding example, a non-administrator user who wants to create a reservation must *ALWAYS* specify an accountable QoS with the mrsvctl -S flag. This specified QoS must have the **ENABLEUSERRSV**flag. By default, a personal reservation is created with an ACL of only the user who created it.

Example 3-116: Allow All Users in Engineering Group to Create Personal Reservations

```
QOSCFG[rsv]     QFLAGS=ENABLEUSERRSV # allow 'rsv' QOS jobs to create user
reservations
GROUPCFG[sales] QDEF=rsv             # allow all users in group sales to access 'rsv'
QOS
...
```

Example 3-117: Allow Specific Users to Create Personal Reservations

```
# special qos has higher job priority and ability to create user reservations
QOSCFG[special] QFLAGS=ENABLEUSERRSV
QOSCFG[special] PRIORITY=1000
# allow betty and steve to use the special qos
USERCFG[betty]  QDEF=special
USERCFG[steve]  QLIST=fast,special,basic QDEF=rsv
...
```

Reservation Accountability

Personal reservations must be configured with a set of accountable credentials. These credentials (user, group, account, and so forth) indicate who is responsible for the resources dedicated by the reservation. If resources are dedicated by a reservation but not consumed by a job, these resources can be charged against the specified accountable credentials. Administrators are allowed to create reservations and specify any accountable credentials for that reservation. While end-users can also be allowed to create and otherwise modify personal reservations, they are only allowed to create reservations with accountable credentials to which they have access. Further, while administrators may manage any reservation, end-users may only control reservations they own.

Like jobs, reservation accountable credentials specify which credentials are charged for reservation usage and what policies are enforced as far as usage limits and allocation management is concerned. (See the mrsvctl command documentation for more information on setting personal reservation credentials.) While similar to jobs, personal reservations do have a separate set of usage limits and different allocation charging policies.

Setting Reservation Default Attributes

Organizations can use reservation profiles to set default attributes for personal reservations. These attributes can include reservation aspects such as management policies, charging credentials, ACLs, host constraints, and time frame settings.

Reservation Limits

Allowing end-users the ability to create advance reservations can lead to potentially unfair and unproductive resource usage. This results from the fact that by default, there is nothing to prevent a user from reserving all resources in a given system or reserving resources during time slots that would greatly impede the scheduler's ability to schedule jobs efficiently. Because of this, it is highly advised that sites initially place either usage or allocation based constraints on the use of personal reservations. This can be achieved using Moab Accounting Manager (see the Moab Accounting Manager Administrator Guide).

Reservation and Job Binding

Moab allows job-to-reservation binding to be configured at an administrator or end-user level. This binding constrains how job to reservation mapping is allowed.

Constraining a job to only run in a particular reservation

Jobs may be bound to a particular reservation at submit time (using the RM extension ADVRES) or dynamically using the mjobctl command (See Job to Reservation Mapping.). In either case, once bound to a reservation, a job may only run in that reservation even if other resources may be found outside of that reservation. The mjobctl command may also be used to dynamically release a job from reservation binding.

Example 3-118: Bind job to reservation

```
> mjobctl -m flags+=advres:grid.3 job1352
```

Example 3-119: Release job from reservation binding

```
> mjobctl -m flags-=advres job1352
```

Constraining a Reservation to Only Accept Certain Jobs

Binding a job to a reservation is independent of binding a reservation to a job. For example, a reservation may be created for user "steve." User "steve" may then submit a number of jobs including one that is bound to that reservation using the **ADVRES** attribute. However, this binding simply forces that one job to use the reservation, it does not prevent the reservation from accepting other jobs submitted by user "steve." To prevent these other jobs from using the reserved resources, reservation to job binding must occur. This binding is accomplished by specifying either general job binding or specific job binding.

General job binding is the most flexible form of binding. Using the BYNAME attribute, a reservation may be created that only accepts jobs specifically bound to it.

Specific job binding is more constraining. This form of binding causes the reservation to only accept specific jobs, regardless of other job attributes and is set using the **JOB** reservation ACL.

Example 3-120: Configure a reservation to accept only jobs that are bound to it

```
> mrsvctl -m flags+=byname grid.3
```

Example 3-121: Remove general reservation to job binding

```
> mrsvctl -m flags-=byname grid.3
```

Example 3-122: Configure a reservation to accept a specific job

```
> mrsvctl -m -a JOB=3456 grid.3
```

Example 3-123: Remove a specific reservation to job binding

```
> mrsvctl -m -a JOB=3456 grid.3 --flags=unset
```

Partitions

- Partition Overview
- Defining Partitions
- Managing Partition Access
- Requesting Partitions
- Per-Partition Settings
- Miscellaneous Partition Issues

Partition Overview

Partitions are a logical construct that divide available resources. Any single resource (compute node) may only belong to a single partition. Often, natural hardware or resource manager bounds delimit partitions such as in the case of disjoint networks and diverse processor configurations within a cluster. For example, a cluster may consist of 256 nodes containing four 64 port switches. This cluster may receive excellent interprocess communication speeds for parallel job tasks located within the same switch but sub-stellar performance for tasks that span switches. To handle this, the site may choose to create four partitions, allowing jobs to run within any of the four partitions but not span them.

While partitions do have value, it is important to note that within Moab, the standing reservation facility provides significantly improved flexibility and should be used in the vast majority of politically motivated cases where partitions may be required under other resource management systems. Standing reservations provide time flexibility, improved access control features, and more extended resource specification options. Also, another Moab facility called Node Sets allows intelligent aggregation of resources to improve per job node allocation decisions. In cases where system partitioning is considered for such reasons, node sets may be able to provide a better solution.

Still, one key advantage of partitions over standing reservations and node sets is the ability to specify partition specific policies, limits, priorities, and scheduling algorithms although this feature is rarely required. An example of this need may be a cluster consisting of 48 nodes owned by the Astronomy Department and 16 nodes owned by the Mathematics Department. Each department may be willing to allow sharing of resources but wants to specify how their partition will be used. As mentioned, many of Moab's scheduling policies may be specified on a per partition basis allowing each department to control the scheduling goals within their partition.

The partition associated with each node should be specified as indicated in the Node Location section. With this done, partition access lists may be specified on a per job or per QoS basis to constrain which resources a job may have access to. (See the QoS Overview for more information.) By default, QoSs and jobs allow global partition access. Note that by default, a job may only use resources within a single partition.

If no partition is specified, Moab creates one partition per resource manager into which all resources corresponding to that resource manager are placed. (This partition is given the same name as the resource manager.)

> A partition may not span multiple resource managers. In addition to these resource manager partitions, a pseudo-partition named " [ALL]" is created that contains the aggregate resources of all partitions.

> While the resource manager partitions are real partitions containing resources not explicitly assigned to other partitions, the " [ALL]" partition is only a convenience object and is not a real partition; thus it cannot be requested by jobs or included in configuration ACLs.

Defining Partitions

Node to partition mappings can be established directly using the NODECFG parameter or indirectly using the FEATUREPARTITIONHEADER parameter. If using direct mapping, this is accomplished as shown in the example that follows.

```
NODECFG[node001]   PARTITION=astronomy
NODECFG[node002]   PARTITION=astronomy
...
NODECFG[node049]   PARTITION=math
...
```

> By default, Moab creates two partitions, "DEFAULT" and "[ALL]." These are used internally, and consume spots in the 31-partition maximum defined in the **MMAX_PAR** parameter. If more partitions are needed, you can adjust the maximum partition count. See Adjusting Default Limits for information on increasing the maximum number of partitions.

Managing Partition Access

Partition access can be constrained by credential ACLs and by limits based on job resource requirements.

Credential Based Access

Determining who can use which partition is specified using the ***CFG** parameters (USERCFG, GROUPCFG, ACCOUNTCFG, QOSCFG, CLASSCFG, and SYSCFG). These parameters allow you to select a partition access list on a credential or system wide basis using the **PLIST** attribute. By default, the access associated with any given job is the logical OR of all partition access lists assigned to the job's credentials.

For example, assume a site with two partitions, *general*, and *test*. The site management would like everybody to use the *general* partition by default. However, one user, Steve, needs to perform the majority of his work on the test partition. Two special groups, staff and management will also need access to use the test partition from time to time but will perform most of their work in the general partition. The following example configuration enables the needed user and group access and defaults for this site:

```
SYSCFG[base]      PLIST=general:test
USERCFG[DEFAULT]  PLIST=general
USERCFG[steve]    PLIST=general:test
GROUPCFG[staff]   PLIST=general:test
GROUPCFG[mgmt]    PLIST=general:test
```

While using a logical OR approach allows sites to add access to certain jobs, some sites prefer to work the other way around. In these cases, access is granted by default and certain credentials are then restricted from accessing various partitions. To use this model, a system partition list must be specified as in the following example:

```
SYSCFG[base]     PLIST=general,test&
USERCFG[demo]    PLIST=test&
GROUPCFG[staff] PLIST=general&
```

In the preceding example, note the ampersand (&). This character, which can be located anywhere in the **PLIST** line, indicates that the specified partition list should be logically AND'd with other partition access lists. In this case, the configuration limits jobs from user *demo* to running in partition *test* and jobs from group *staff* to running in partition *general*. All other jobs are allowed to run in either partition.

> When using AND-based partition access lists, the base system access list must be specified with **SYSCFG**.

Per Job Resource Limits

Access to partitions can be constrained based on the resources requested on a per job basis with limits on both minimum and maximum resources requested. All limits are specified using PARCFG. See Usage Limits for more information on the available limits.

```
PARCFG[amd]      MAX.PROC=16
PARCFG[pIII]     MAX.WCLIMIT=12:00:00 MIN.PROC=4
PARCFG[aix]      MIN.NODE=12
```

Requesting Partitions

Users may request to use any partition they have access to on a per job basis. This is accomplished using the resource manager extensions since most native batch systems do not support the partition concept. For example, on a TORQUE system, a job submitted by a member of the group *staff* could request that the job run in the *test* partition by adding the line `-l partition=test` to the `qsub` command line. See the resource manager extension overview for more information on configuring and using resource manager extensions.

Per-Partition Settings

The following settings can be specified on a per-partition basis using the PARCFG parameter:

Setting	Description
GMETRIC	Specifies a generic metric to apply to the partition. It is configured like a Moab parameter, with the gmetric name inside square brackets. Specify multiple gmetrics by separating each configuration with a space. For example: `PARCFG[par1] GMETRIC[GM1]=20 GMETRIC[GM2]=10` *Partition `par1` has a `GM1` metric of 20 and a `GM2` metric of 10.*
JOBNODEMATCHPOLICY	Specifies the JOBNODEMATCHPOLICY to be applied to jobs that run in the specified partition.
NODEACCESSPOLICY	Specifies the NODEACCESSPOLICY to be applied to jobs that run in the specified partition.
NODEALLOCATIONPOLICY	Specifies the NODEALLOCATIONPOLICY to be applied to jobs that run in the specified partition.
USETTC	Specifies whether TTC specified at submission should be used and displayed by the scheduler.
VMCREATEDURATION	Specifies the maximum amount of time VM creation can take before Moab considers it a failure (in [HH[:MM[:SS]). If no value is set, there is no maximum limit.
VMDELETEDURATION	Specifies the maximum amount of time VM deletion can take before Moab considers it a failure (in [HH[:MM[:SS]). If no value is set, there is no maximum limit.
VMMIGRATEDURATION	Specifies the maximum amount of time VM migration can take before Moab considers it a failure (in [HH[:MM[:SS]). If no value is set, there is no maximum limit.

Miscellaneous Partition Issues

A brief caution: Use of partitions has been quite limited in recent years as other, more effective approaches are selected for site scheduling policies. Consequently, some aspects of partitions have received only minor testing. Still, note that partitions are fully supported and any problem found will be rectified.

Related topics

- Standing Reservations
- Node Sets
- FEATUREPARTITIONHEADER parameter
- PARCFG parameter

Quality of Service (QoS) Facilities

This section describes how to do the following:

- Allow key projects access to special services (such as preemption, resource dedication, and advance reservations).
- Provide access to special resources by requested QoS.
- Enable special treatment within priority and fairshare facilities by requested QoS.
- Provide exemptions to usage limits and other policies by requested QoS.
- Specify delivered service and response time targets.
- Enable job deadline guarantees.
- Control the list of QoSs available to each user and job.
- Enable special charging rates based on requested or delivered QoS levels.
- Enable limits on the extent of use for each defined QoS.
- Monitor current and historical usage for each defined QoS.

It contains the following sub-sections:

- QoS Overview
- QoS Enabled Privileges
  - Special Prioritization
  - Service Access and Constraints
  - Usage Limits and Overrides
  - Service Access Thresholds
  - Preemption Management
- Managing QoS Access

- Requesting QoS Services at Job Submission
- Restricting Access to Special Attributes

QoS Overview

Moab's QoS facility allows a site to give special treatment to various classes of jobs, users, groups, and so forth. Each QoS object can be thought of as a container of special privileges ranging from fairness policy exemptions, to special job prioritization, to special functionality access. Each QoS object also has an extensive access list of users, groups, and accounts that can access these privileges.

Sites can configure various QoSs each with its own set of priorities, policy exemptions, and special resource access settings. They can then configure user, group, account, and class access to these QoSs. A given job will have a default QoS and may have access to several additional QoSs. When the job is submitted, the submitter may request a specific QoS or just allow the default QoS to be used. Once a job is submitted, a user may adjust the QoS of the job at any time using the setqos command. The setqos command will only allow the user to modify the QoS of that user's jobs and only change the QoS to a QoS that this user has access to. Moab administrators may change the QOS of any job to any value.

Jobs can be granted access to QoS privileges if the QoS is listed in the system default configuration QDEF (QoS default) or QLIST (QoS access list), or if the QoS is specified in the QDEF or QLIST of a user, group, account, or class associated with that job. Alternatively, a user may access QoS privileges if that user is listed in the QoSs MEMBERULIST attribute.

The mdiag -q command can be used to obtain information about the current QoS configuration including specified credential access.

QoS Enabled Privileges

The privileges enabled via QoS settings may be broken into the following categories:

- **Special Prioritization** on page 500
- **Service Access and Constraints** on page 501
- **Usage Limits and Overrides** on page 504
- **Service Access Thresholds** on page 505
- **Preemption Management** on page 505

All privileges are managed via the QOSCFG parameter.

Special Prioritization

Attribute name	Description
FSTARGET	Specifies QoS fairshare target.
FSWEIGHT	Sets QoS fairshare weight offset affecting a job's fairshare priority component.

Attribute name	Description
PRIORITY	Assigns priority to all jobs requesting particular QoS.
QTTARGET	Sets QoS queuetime target affecting a job's target priority component and QoS delivered.
QTWEIGHT	Sets QoS queuetime weight offset affecting a job's service priority component.
XFTARGET	Sets QoS XFactor target affecting a job's target priority component and QoS delivered.
XFWEIGHT	Sets QoS XFactor weight offset affecting a job's service priority component.

Example 3-124:

```
# assign priority for all qos geo jobs

QOSCFG[geo]  PRIORITY=10000
```

Service Access and Constraints

The QoS facility can be used to enable special services and to disable default services. These services are enabled/disabled by setting the QoS **QFLAGS** attribute.

Flag Name	Description
DEADLINE	Job may request an absolute or relative completion deadline and Moab will reserve resources to meet that deadline. (An alternative priority based deadline behavior is discussed in the PRIORITY FACTORS section.)
DEDICATED	Moab dedicates all resources of an allocated node to the job meaning that the job will not share a node's compute resources with any other job.
ENABLEUSERRSV	Allow user or personal reservations to be created and managed.
IGNALL	Scheduler ignores all resource usage policies for jobs associated with this QoS.

Flag Name	Description
JOBPRIOACCRUALPOLICY	Specifies how Moab should track the dynamic aspects of a job's priority. The two valid values are *ACCRUE* and *RESET*. • *ACCRUE* indicates that the job will accrue queuetime based priority from the time it is submitted unless it violates any of the policies not specified in JOBPRIOEXCEPTIONS. • *RESET* indicates that it will accrue priority from the time it is submitted unless it violates any of the JOBPRIOEXCEPTIONS. However, with *RESET*, if the job does violate JOBPRIOEXCEPTIONS then its queuetime based priority will be reset to 0. **JOBPRIOACCRUALPOLICY** is a global parameter, but can be configured to work only in **QOSCFG**: `QOSCFG[arrays]  JOBPRIOACCRUALPOLICY=ACCRUE` The following old **JOBPRIOACCRUALPOLICY** values have been deprecated and should be adjusted to the following values: • QUEUEPOLICY = ACCRUE and JOBPRIOEXCEPTIONS SOFTPOLICY,HARDPOLICY • QUEUEPOLICYRESET = RESET and JOBPRIOEXCEPTIONS SOFTPOLICY,HARDPOLICY • ALWAYS = ACCRUE and JOBPRIOEXCEPTIONS ALL • FULLPOLICY = ACCRUE and JOBPRIOEXCEPTIONS NONE • FULLPOLICYRESET = RESET and JOBPRIOEXCEPTIONS NONE
JOBPRIOEXCEPTIONS	Specifies exceptions for calculating a job's dynamic priority (QUEUETIME, XFACTOR, TARGETQUEUETIME). Valid values are a comma delimited list of any of the following: *DEFER*, *DEPENDS*, *SOFTPOLICY*, *HARDPOLICY*, *IDLEPOLICY*, *USERHOLD*, *BATCHHOLD*, and *SYSTEMHOLD* (*ALL* or *NONE* can also be specified on their own). Normally, when a job violates a policy, is placed on hold, or has an unsatisfied dependency, it will not accrue priority. Exceptions can be configured to allow a job to accrue priority in spite of any of these violations. With *DEPENDS* a job will increase in priority even if there exists an unsatisfied dependency. With *SOFTPOLICY*, *HARDPOLICY*, or *IDLEPOLICY* a job can accrue priority despite violating a specific limit. With *DEFER*, *USERHOLD*, *BATCHHOLD*, or *SYSTEMHOLD* a job can accrue priority despite being on hold. **JOBPRIOEXCEPTIONS** is a global parameter, but can be configured to work only in QOSCFG: `QOSCFG[arrays]  JOBPRIOEXCEPTIONS=IDLEPOLICY`
NOBF	Job is not considered for backfill.

Flag Name	Description
NORESERVATION	Job should never reserve resources regardless of priority.
NTR	Job is prioritized as next to run (NTR) and backfill is disabled to prevent other jobs from jumping in front of ones with the NTR flag. It is important to note that jobs marked with this flag should not be blocked. If they are, Moab will stop scheduling because if a job is marked with this flag, no other jobs will be run until the flagged NTR (Next to Run) job starts. Consider using the **PRIORITY** attribute of the **QOSCFG[<QOSID>]** on page 1001 parameter instead, when possible. Or, as you may encounter a scheduling delay for NTR-flagged jobs to start, consider using the RESERVATIONDEPTH and RESERVATIONQOSLIST parameters to provide better scheduling flow. See **Reservation Policies** on page 458 (especially the section on Assigning Per-QoS Reservation Creation Rules) for more information.
PREEMPTCONFIG	User jobs may specify options to alter how preemption impacts the job such as min-preempttime.
PREEMPTEE	Job may be preempted by higher priority **PREEMPTOR** jobs.
PREEMPTFSV	Job may be preempted by higher priority **PREEMPTOR** jobs if it exceeds its fairshare target when started.
PREEMPTOR	Job may preempt lower priority **PREEMPTEE** jobs.
PREEMPTSPV	Job may be preempted by higher priority **PREEMPTOR** jobs if it currently violates a soft usage policy limit.
PROVISION	If the job cannot locate available resources with the needed OS or software, the scheduler may provision a number of nodes to meet the needed OS or software requirements.
RESERVEALWAYS	Job should create resource reservation regardless of job priority.
RUNNOW	Boosts a job's system priority and makes the job a preemptor. RUNNOW overrides resource restrictions such as MAXJOB or MAXPROC.
TRIGGER	The job is able to directly specify triggers.

Flag Name	Description
USERESERVED[:<RSVID>]	Job may only use resources within accessible reservations. If *<RSVID>* is specified, job may only use resources within the specified reservation.

Example 3-125: For lowprio QoS job, disable backfill and make job preemptible

```
QOSCFG[lowprio]  QFLAGS=NOBF,PREEMPTEE
```

Example 3-126: Bind all jobs to chemistry reservation

```
QOSCFG[chem-b]  QFLAGS=USERESERVED:chemistry
```

Other QoS Attributes

In addition to the flags, there are attributes that alter service access.

Attribute name	Description
SYSPRIO	Sets the system priority on jobs associated with this QoS.

Example 3-127: All jobs submitted under a QoS sample receive a system priority of 1

```
QOSCFG[sample]  SYSPRIO=1
```

Per QoS Required Reservations

If desired, jobs associated with a particular QoS can be locked into a reservation or reservation group using the **REQRID** attribute. For example, to force jobs using QoS *jasper* to only use the resources within the *failsafe* standing reservation, use the following:

```
QOSCFG[jasper] REQRID=failsafe
...
```

Usage Limits and Overrides

All credentials, including QoS, allow specification of job usage limits as described in the Basic Fairness Policies overview. In such cases, jobs are constrained by the most limiting of all applicable policies. With QoSs, an override limit may also be specified and with this limit, jobs are constrained by the override, regardless of other limits specified. The following parameters can override the throttling policies from other credentials:

OMAXJOB, **OMAXNODE**, **OMAXPE**, **OMAXPROC**, **OMAXPS**, **OMAXJPROC**, **OMAXJPS**, **OMAXJWC**, and **OMAXJNODE**.

(See Usage Limits/Throttling Policies Override Limits.)

Example 3-128:

```
# staff QoS should have a limit of 48 jobs, ignoring the user limit
USERCFG[DEFAULT]    MAXJOB=10
QOSCFG[staff]       OMAXJOB=48
```

Service Access Thresholds

Jobs can be granted access to services such as preemption and reservation creation, and they can be granted access to resource reservations. However, with QoS thresholds, this access can be made conditional on the current queuetime and XFactor metrics of an idle job. The following table lists the available QoS service thresholds:

Threshold attribute	Description
PREEMPTQTTHRESHOLD	A job with this QoS becomes a preemptor if the specified queuetime threshold is reached.
PREEMPTXFTHRESHOLD	A job with this QoS becomes a preemptor if the specified XFactor threshold is reached.
RSVQTTHRESHOLD	A job with this QoS can create a job reservation to guarantee resource access if the specified queuetime threshold is reached.
RSVXFTHRESHOLD	A job with this QoS can create a job reservation to guarantee resource access if the specified XFactor threshold is reached.
ACLQTTHRESHOLD	A job with this QoS can access reservations with a corresponding QoS ACL only if the specified queuetime threshold is reached.
ACLXFTHRESHOLD	A job with this QoS can access reservations with a corresponding QoS ACL only if the specified XFactor threshold is reached.
TRIGGERQTTHRESHOLD	If a job with this QoS fails to run before this threshold is reached, any failure triggers associated with this QoS will fire.

Preemption Management

Job preemption facilities can be controlled on a per-QoS basis using the PREEMPTEE and PREEMPTOR flags. Jobs that are preemptible can optionally be constrained to only be preempted in a particular manner by specifying the QoS **PREEMPTPOLICY** attribute as in the following example:

```
QOSCFG[special] QFLAGS=PREEMPTEE PREEMPTPOLICY=CHECKPOINT
```

For preemption to be effective, a job must be marked as a preemptee and must be enabled for the requested preemption type. For example, if the PREEMPTPOLICY is set to suspend, a potential target job must be both a preemptee and marked with the job flag **SUSPENDABLE**. (See suspension for more information.) If the target job is not suspendable, it will be either requeued or canceled. Likewise, if the **PREEMPTPOLICY** is set to *requeue*, the job will be requeued if it is marked restartable. Otherwise, it will be canceled.

The minimum time a job must run before being considered eligible for preemption can also be configured on a per-QoS basis using the **PREEMPTMINTIME** parameter, which is analogous to the JOBPREEMPTMINACTIVETIME. Conversely, **PREEMPTMAXTIME** sets a threshold for which a job is no longer eligible for preemption; see JOBPREEMPTMAXACTIVETIME for analogous details.

The **PREEMPTEES** attribute allows you to specify which QoSs that a job in a specific QoS is allowed to preempt. The **PREEMPTEES** list is a comma-delimited list of QoS IDs. When a **PREEMPTEES** attribute is specified, a job using that QoS can only preempt jobs using QoSs listed in the **PREEMPTEES** list. In turn, those QoSs must be flagged as *PREEMPTEE* as in the following example:

```
QOSCFG[a] QFLAGS=PREEMPTOR PREEMPTEES=b,c
QOSCFG[b] QFLAGS=PREEMPTEE
QOSCFG[c] QFLAGS=PREEMPTEE
```

In the example, jobs in the 'a' QoS can only preempt jobs in the b and c QoSs.

Managing QoS Access

Specifying Credential Based QoS Access

You can define the privileges allowed within a QoS by using the **QOSCFG** parameter; however, in most cases access to the QoS is enabled via credential specific ***CFG** parameters, specifically the USERCFG, GROUPCFG, ACCOUNTCFG, and CLASSCFG parameters, which allow defining QoS access lists and QoS defaults. Specify credential specific QoS access by using the **QLIST** and/or **QDEF** attributes of the associated credential parameter.

QOS Access via Logical OR

To enable QoS access, the **QLIST** and/or **QDEF** attributes of the appropriate user, group, account, or class/queue should be specified as in the following example:

```
# user john's jobs can access QOS geo, chem, or staff with geo as default
USERCFG[john]     QDEF=geo   QLIST=geo,chem,staff
# group system jobs can access the development qos
GROUPCFG[systems] QDEF=development
# class batch jobs can access the normal qos
CLASSCFG[batch]   QDEF=normal
```

By default, jobs may request a QoS if access to that QoS is allowed by any of the job's credentials. (In the previous example, a job from user *john* submitted to the class *batch* could request QoSs *geo*, *chem*, *staff*, or *normal*).

QOS Access via Logical AND

If desired, QoS access can be masked or logically AND'd if the QoS access list is specified with a terminating ampersand (&) as in the following example:

```
# user john's jobs can access QOS geo, chem, or staff with geo as default
USERCFG[john]     QDEF=geo   QLIST=geo,chem,staff
# group system jobs can access the development qos
GROUPCFG[systems] QDEF=development
# class batch jobs can access the normal qos
CLASSCFG[batch]   QDEF=normal
# class debug jobs can only access the development or lowpri QoSs regardless of other
credentials
CLASSCFG[debug]   QLIST=development,lowpri&
```

Specifying QoS Based Access

QoS access may also be specified from within the QoS object using the QoS **MEMBERULIST** attribute as in the following example:

```
# define qos premiere and grant access to users steve and john
QOSCFG[premiere]  PRIORITY=1000  QFLAGS=PREEMPTOR  MEMBERULIST=steve,john
```

By default, if a job requests a QoS that it cannot access, Moab places a hold on that job. The QOSREJECTPOLICY can be used to modify this behavior.

Requesting QoS Services at Job Submission

By default, jobs inherit a default QoS based on the user, group, class, and account associated with the job. If a job has access to multiple QoS levels, the submitter can explicitly request a particular QoS using the QoS resource manager extension as in the following example:

```
> msub -l nodes=1,walltime=100,qos=special3 job.cmd
```

Restricting Access to Special Attributes

By default, Moab allows all users access to special attributes such as node access policy. By enabling the QoS facility **SPECATTRS**, the access to these policies can be restricted. For example, to enable the facility, in the moab.cfg file, specify `QOSCFG[DEFAULT] SPECATTRS=`. Then, to allow access to the special attributes, indicate which special attributes a specific QoS may access.

```
QOSCFG[DEFAULT] SPECATTRS=
QOSCFG[high]  SPECATTRS=NACCESSPOLICY
```

Related topics

- Credential Overview
- Allocation Management Overview
- Rollback Reservations
- Job Deadlines
- Using QoS preemption

Optimizing Scheduling Behavior – Backfill and Node Sets

Optimization Overview

Moab optimizes cluster performance. Every policy, limit, and feature is designed to allow maximum scheduling flexibility while enforcing the required constraints. A driving responsibility of the scheduler is to do all in its power to maximize system use and to minimize job response time while honoring the policies that make up the site's mission goals.

However, as all jobs are not created equal, optimization must be abstracted slightly further to incorporate this fact. Cluster optimization must also focus on targeted cycle delivery. In the scientific HPC community, the true goal of a cluster is to maximize delivered research. For businesses and other organizations, the purposes may be slightly different, but all organizations agree on the simple tenet that the cluster should optimize the site's mission goals.

To obtain this goal, the scheduler has several levels of optimization it performs:

Level	Description
Workload Ordering	Prioritizing workload and utilizing backfill
Intelligent Resource Allocation	Selecting those resources that best meet the job's needs or best enable future jobs to run (see node allocation)
Maximizing Intra-Job Efficiency	Selecting the type of nodes, collection of nodes, and proximity of nodes required to maximize job performance by minimizing both job compute and inter-process communication time (see node sets and node allocation)
Job Pree-mption	Preempting jobs to allow the most important jobs to receive the best response time (see pree-mption)
Utilizing Flexible Policies	Using policies that minimize blocking and resource fragmentation while enforcing needed con-straints (see soft throttling policies and reservations)

Backfill

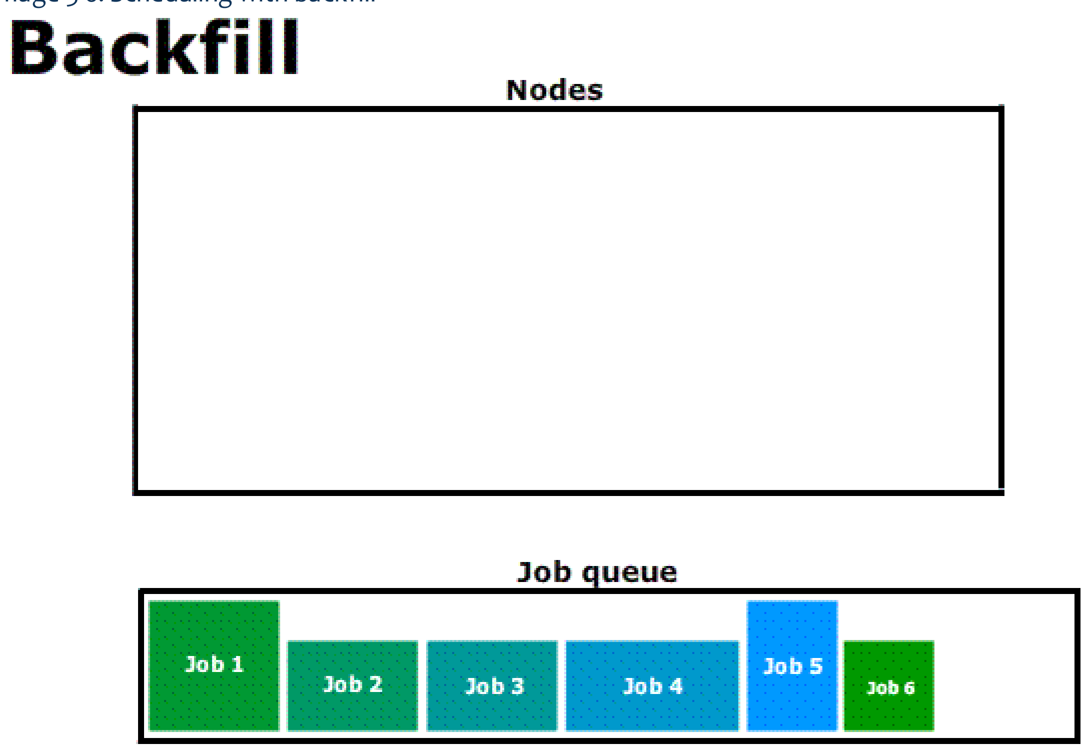
- Backfill Overview
- Backfill Algorithms
- Configuring Backfill

Backfill Overview

Backfill is a scheduling optimization that allows a scheduler to make better use of available resources by running jobs out of order. When Moab schedules, it prioritizes the jobs in the queue according to a number of factors and then orders the jobs into a highest priority first (or priority FIFO) sorted list. It starts the jobs one by one stepping through the priority list until it reaches a job it cannot start. Because all jobs and reservations possess a start time and a wallclock limit, Moab can determine the completion time of all jobs in the queue. Consequently, Moab can also determine the earliest the needed resources will become available for the highest priority job to start.

Backfill operates based on this earliest job start information. Because Moab knows the earliest the highest priority job can start, and which resources it will need at that time, it can also determine which jobs can be started without delaying this job. Enabling backfill allows the scheduler to start other, lower-priority jobs so long as they do not delay the highest priority job. If backfill is enabled, Moab protects the highest priority job's start time by creating a job reservation to reserve the needed resources at the appropriate time. Moab then can start any job that will not interfere with this reservation.

Image 3-6: Scheduling with backfill



Backfill offers significant scheduler performance improvement. In a typical large system, enabling backfill increases system utilization by about 20% and improves turnaround time by an even greater amount. Because of the way it works, essentially filling in holes in node space, backfill tends to favor smaller and shorter running jobs more than larger and longer running ones. It is common to see over 90% of these small and short jobs backfilled. Consequently, sites will see marked improvement in the level of service delivered to the small, short jobs and moderate to little improvement for the larger, long ones.

With most algorithms and policies, there is a trade-off. Backfill is not an exception but the negative effects are minor. Because backfill locates jobs to run from throughout the idle job queue, it tends to diminish the influence of the job prioritization a site has chosen and thus may negate some desired workload steering attempts through this prioritization. Although by default the start time of the highest priority job is protected by a reservation, there is nothing to prevent the third priority job from starting early and possibly delaying the start of the second priority job. This issue is addressed along with its trade-offs later in this section.

Another problem is a little more subtle. Consider the following scenario involving a two-processor cluster. Job A has a four-hour wallclock limit and requires one processor. It started one hour ago (time zero) and will reach its wallclock limit in three more hours. Job B is the highest priority idle job and requires two processors for one hour. Job C is the next highest priority job and requires one processor for two hours. Moab examines the jobs and correctly determines that job A must finish in three hours and thus, the earliest job B can start is in three hours. Moab also determines that job C can start and finish in less than this amount of time. Consequently, Moab starts job C on the idle processor at time one. One hour later (time two), job A completes early. Apparently, the user overestimated the amount of time job A would need by a few hours. Since job B is now the highest priority job, it should be able to run. However, job C, a lower priority job was started an hour ago and the resources needed for job B are not available. Moab re-evaluates job B's reservation and determines that it can slide forward an hour. At time three, job B starts.

In review, backfill provided positive benefits. Job A successfully ran to completion. Job C was started immediately. Job B was able to start one hour sooner than its original target time, although, had backfill not been enabled, job B would have been able to run two hours earlier.

The scenario just described occurs quite frequently because user estimates for job duration are generally inaccurate. Job wallclock estimate accuracy, or wallclock accuracy, is defined as the ratio of wall time required to actually run the job divided by the wall time requested for the job. Wallclock accuracy varies from site to site but the site average is rarely better than 50%. Because the quality of the walltime estimate provided by the user is so low, job reservations for high priority jobs are often later than they need to be.

Although there do exist some minor drawbacks with backfill, its net performance impact on a site's workload is very positive. While a few of the highest priority jobs may get temporarily delayed, their position as highest priority was most likely accelerated by the fact that jobs in front of them were able to start earlier due to backfill. Studies have shown that only a very small number of jobs are truly delayed and when they are, it is only by a fraction of their total queue time. At the same time, many jobs are started significantly earlier than would have occurred without backfill.

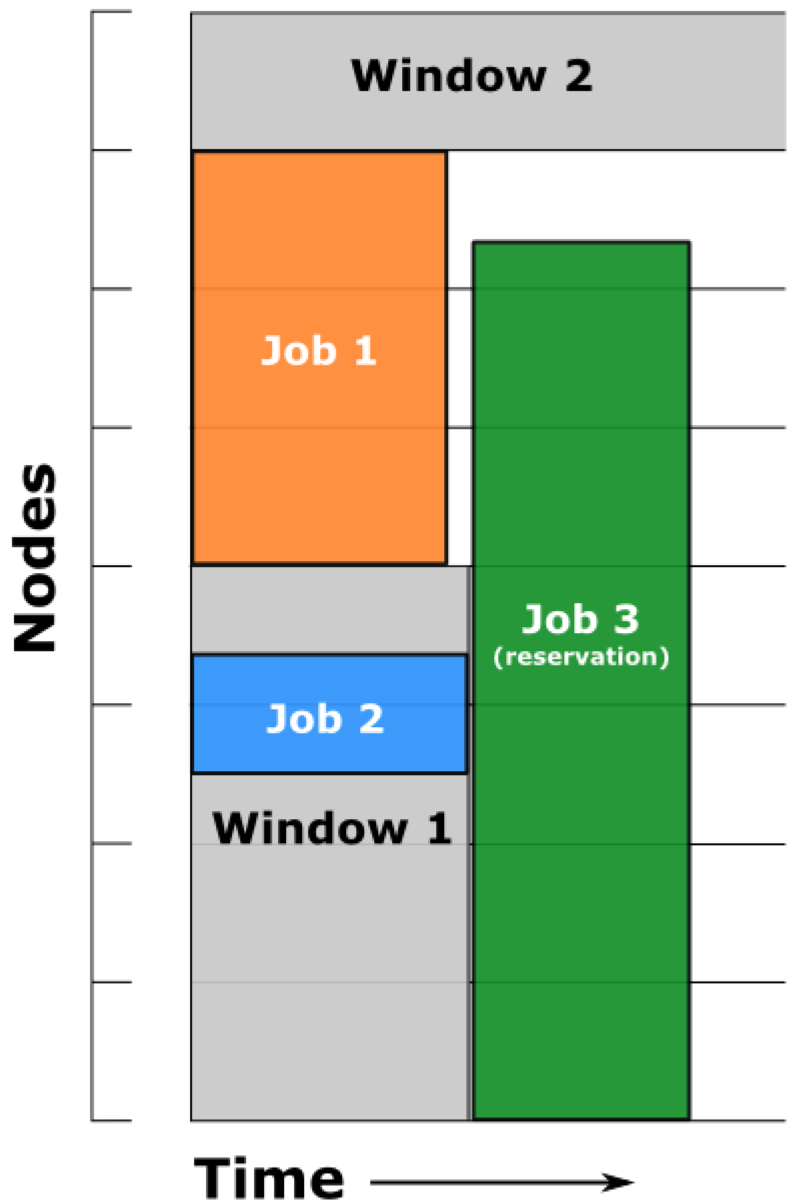
The following image demonstrates how Moab might schedule a queue using backfill.

Backfill Algorithms

The algorithm behind Moab backfill scheduling is straightforward, although there are a number of issues and parameters that should be highlighted. First of all, Moab makes two backfill scheduling passes. For each pass, Moab selects a list of jobs that are eligible for backfill. On the first pass, only those jobs that meet the constraints of the soft fairness throttling policies are considered and scheduled. The second pass expands this list of jobs to include those that meet the hard (less constrained) fairness throttling policies.

The second important concept regarding Moab backfill is the concept of backfill windows. The figure below shows a simple batch environment containing two running jobs and a reservation for a third job. The present time is represented by the leftmost end of the box with the future moving to the right. The light gray boxes represent currently idle nodes that are eligible for backfill. For this example, let's assume that the space represented covers 8 nodes and a 2 hour time frame. To determine backfill windows, Moab analyzes the idle nodes essentially looking for largest node-time rectangles. It determines that there are two backfill windows. The first window, Window 1, consists of 4 nodes that are available for only one hour (because some of the nodes are blocked by the reservation for Job 3). The second window contains only one node but has no time limit because this node is not blocked by the reservation for Job 3. It is important to note that these backfill windows overlap.

Image 3-7: Backfillable nodes create backfill windows 1 and 2



Once the backfill windows have been determined, Moab begins to traverse them. The current behavior is to traverse these windows widest window first (most nodes to fewest nodes). As each backfill window is evaluated, Moab applies the backfill algorithm specified by the BACKFILLPOLICY parameter.

If the *FIRSTFIT* algorithm is applied, the following steps are taken:

1. The list of feasible backfill jobs is filtered, selecting only those that will actually fit in the current backfill window.
2. The first job is started.
3. While backfill jobs and idle resources remain, repeat step 1.

If *NONE* is set, the backfill policy is disabled.

Other backfill policies behave in a generally similar manner. The parameters documentation provides further details.

Liberal versus Conservative Backfill

By default, Moab reserves only the highest priority job resulting in a liberal and aggressive backfill. This reservation guarantees that backfilled jobs will not delay the highest priority job, although they may delay other jobs. The parameter RESERVATIONDEPTH controls how conservative or liberal the backfill policy is. This parameter controls how deep down the queue priority reservations will be made. While increasing this parameter improves guarantees that priority jobs will not be bypassed, it reduces the freedom of the scheduler to backfill resulting in somewhat lower system utilization. The significance of the trade-offs should be evaluated on a site by site basis.

Configuring Backfill

Backfill Policies

Backfill is enabled in Moab by specifying the BACKFILLPOLICY parameter. By default, backfill is enabled in Moab using the *FIRSTFIT* algorithm. However, this parameter can also be set to *NONE* (disabled).

The number of reservations that protect the resources required by priority jobs can be controlled using RESERVATIONDEPTH. This depth can be distributed across job QoS levels using RESERVATIONQOSLIST.

Backfill Chunking

In a batch environment saturated with serial jobs, serial jobs will, over time, dominate the resources available for backfill at the expense of other jobs. This is due to the time-dimension fragmentation associated with running serial jobs. For example, given an environment with an abundance of serial jobs, if a multi-processor job completes freeing processors, one of three things will happen:

1. The freed resources are allocated to another job requiring the same number of processors.
2. Additional jobs may complete at the same time allowing a larger job to allocate the aggregate resources.
3. The freed resources are allocated to one or more smaller jobs.

In environments where the scheduling iteration is much higher than the average time between completing jobs, case 3 occurs far more often than case 2, leading to smaller and smaller jobs populating the system over time.

To address this issue, the scheduler incorporates the concept of chunking. Chunking allows the scheduler to favor case 2 maintaining a more controlled balance between large and small jobs. The idea of chunking involves establishing a time-based threshold during which resources available for backfill are aggregated. This threshold is set using the parameter BFCHUNKDURATION. When resources are freed, they are made available only to jobs of a certain size (set using the parameter BFCHUNKSIZE) or larger. These resources remain protected from smaller jobs until either additional resources are freed up and a larger job can use the aggregate resources, or until the **BFCHUNKDURATION** threshold time expires.

> Backfill chunking is only activated when a job of size **BFCHUNKSIZE** or larger is blocked in backfill due to lack of resources.

It is important to note that the optimal settings for these parameters is very site-specific and will depend on the workload (including the average job turnaround time, job size, and mix of large to small jobs), cluster resources, and other scheduling environmental factors. Setting too restrictive values needlessly reduces utilization while settings that are too relaxed do not allowed the desired aggregation to occur.

> Backfill chunking is only enabled in conjunction with the *FIRSTFIT* backfill policy.

Virtual Wallclock Time Scaling

In most environments, users submit jobs with rough estimations of the wallclock times. Within the HPC industry, a job typically runs for 40% of its specified wallclock time. Virtual Wallclock Time Scaling takes advantage of this fact to implement a form of optimistic backfilling. Jobs that are eligible for backfilling and not restricted by other policies are virtually scaled by the BFVIRTUALWALLTIMESCALINGFACTOR (assuming that the jobs finish before this new virtual wallclock limit). The scaled jobs are then compared to backfill windows to see if there is space and time for them to be scheduled. The scaled jobs are only scheduled if there is no possibility that it will conflict with a standing or administrator reservation. Conflicts with such reservations occur if the virtual wallclock time overlaps a reservation, or if the original non-virtual wallclock time overlaps a standing or administrator reservation. Jobs that can fit into an available backfill window without having their walltime scaled are backfilled "as-is" (meaning, without virtually scaling the original walltime).

> Virtual Wallclock Time Scaling is only enabled when the BFVIRTUALWALLTIMESCALINGFACTOR parameter is defined.

If a virtually-scaled job fits into a window, and is backfilled, it will run until completion or until it comes within one scheduling iteration (RMPOLLINTERVAL defines the exact time of an iteration) of the virtual wallclock time expiration. In the latter case the job's wallclock time is restored to its original time and Moab checks and resolves conflicts caused by this "expansion." Conflicts may occur when the backfilled job is restored to its full duration resulting in reservation overlap. The BFVIRTUALWALLTIMECONFLICTPOLICY parameter controls how Moab handles these conflicts.

If the **BFVIRTUALWALLTIMECONFLICTPOLICY** parameter is set to *NONE* or is not specified, the overlapped job reservations are rescheduled.

Related topics

- BACKFILLDEPTH Parameter
- BACKFILLPOLICY Parameter

- BFMINVIRTUALWALLTIME
- Reservation Policy Overview

Node Set Overview

Node Set Usage Overview

While backfill improves the scheduler's performance, this is only half the battle. The efficiency of a cluster, in terms of actual work accomplished, is a function of both scheduling performance and individual job efficiency. In many clusters, job efficiency can vary from node to node as well as with the node mix allocated. Most parallel jobs written in popular languages such as MPI or PVM do not internally load balance their workload and thus run only as fast as the slowest node allocated. Consequently, these jobs run most effectively on homogeneous sets of nodes. However, while many clusters start out as homogeneous, they quickly evolve as new generations of compute nodes are integrated into the system. Research has shown that this integration, while improving scheduling performance due to increased scheduler selection, can actually decrease average job efficiency.

A feature called node sets allows jobs to request sets of common resources without specifying exactly what resources are required. Node set policy can be specified globally or on a per-job basis. In addition to their use in forcing jobs onto homogeneous nodes, these policies may also be used to guide jobs to one or more types of nodes on which a particular job performs best, similar to job preferences available in other systems. For example, an I/O intensive job may run best on a certain range of processor speeds, running slower on slower nodes, while wasting cycles on faster nodes. A job may specify `ANYOF:FEATURE:bigmem,fastos` to request nodes with the `bigmem` or `fastos` feature. Alternatively, if a simple feature-homogeneous node set is desired, `ONEOF:FEATURE` may be specified. On the other hand, a job may request a feature based node set with the configuration `ONEOF:FEATURE:bigmem,fastos`, in which case Moab will first attempt to locate adequate nodes where all nodes contain the `bigmem` feature. If such a set cannot be found, Moab will look for sets of nodes containing the other specified features. In highly heterogeneous clusters, the use of node sets improves job throughput by 10 to 15%.

Node sets can be requested on a system wide or per job basis. System wide configuration is accomplished via the **NODESET*** parameters while per job specification occurs via the resource manager extensions.

> The GLOBAL node is included in all feature node sets.

When creating node sets, you have the option of using a fixed configuration or of creating node sets dynamically (by using the msub command). This topic explains how to set up both node set use cases.

Node Set Configuration Examples

Global node sets are defined using the NODESETPOLICY, NODESETATTRIBUTE, NODESETLIST, and NODESETISOPTIONAL parameters. As stated before, you can create node sets dynamically (see **Dynamic example** on page 518) or with a fixed configuration (see **Fixed configuration example** on page 516). The use of these parameters can be best highlighted with two examples.

Fixed configuration example

In this example, a large site possesses a Myrinet based interconnect and wishes to, whenever possible, allocate nodes within Myrinet switch boundaries. To accomplish this, they could assign node attributes to each node indicating which switch it was associated with (`switchA`, `switchB`, and so forth) and then use the following system wide node set configuration:

```
NODESETPOLICY     ONEOF
NODESETATTRIBUTE  FEATURE
NODESETISOPTIONAL TRUE
NODESETLIST       switchA,switchB,switchC,switchD
...
```

Node Set Policy

In the preceding example, the NODESETPOLICY parameter is set to the policy *ONEOF* and tells Moab to allocate nodes within a single attribute set. Other node set policies are listed in the following table:

Policy	Description
ANYOF	Select resources from all sets contained in node set list. The job could span multiple node sets.
FIRSTOF	Select resources from first set to match specified constraints.
ONEOF	Select a single set that contains adequate resources to support job.

Node Set Attribute

The example's NODESETATTRIBUTE parameter is set to *FEATURE*, specifying that the node sets are to be constructed along node feature boundaries.

You could also set the **NODESETATTRIBUTE** to *VARATTR*, specifying that node sets are to be constructed according to VARATTR values on the job.

Node Set Constraint Handling

The next parameter, NODESETISOPTIONAL, indicates that Moab should not delay the start time of a job if the desired node set is not available but adequate idle resources exist outside of the set. Setting this parameter to *TRUE* basically tells Moab to attempt to use a node set if it is available, but if not, run the job as soon as possible anyway.

> Setting **NODESETISOPTIONAL** to *FALSE* will force the job to always run in a complete nodeset regardless of any start delay this imposes.

Node Set List

Finally, the NODESETLIST value of `switchA switchB...` tells Moab to only use node sets based on the listed feature values. This is necessary since sites will often use node features for many purposes and the resulting node sets would be of little use for switch proximity if they were generated based on irrelevant node features indicating things such as processor speed or node architecture.

To add nodes to the **NODESETLIST**, you must configure features on your nodes using the NODECFG **FEATURES** on page 558 attribute.

```
NODECFG[node01] FEATURES=switchA
NODECFG[node02] FEATURES=switchA
NODECFG[node03] FEATURES=switchB
```

Nodes node01 and node02 contain the switchA feature, and node node03 contains the switchB feature.

Node Set Priority

When resources are available in more than one resource set, the NODESETPRIORITYTYPE parameter allows control over how the best resource set is selected. Legal values for this parameter are described in the following table:

Priority Type	Description	Details
AFFINITY	Avoid a resource set with negative affinity.	Choosing this type causes Moab to select a node set with no negative affinity nodes (nodes that have a reservation that with negative affinity). If all node sets have negative affinity, then Moab will select the first matching node set.

Priority Type	Description	Details
BESTFIT	Select the smallest resource set possible.	Choosing this type causes Moab, when selecting a node set, to eliminate sets that do not have all the required resources. From the remaining sets, Moab chooses the set with the least amount of resources. This priority type most closely matches the job requirements in order to waste the least amount of resources. This type minimizes fragmentation of larger resource sets.
MINLOSS	Select the resource set that results in the minimal wasted resources assuming no internal job load balancing is available. (Assumes parallel jobs only run as fast as the slowest allocated node.)	Choosing this type works only when using the following configuration: NODESETATTRIBUTE *FEATURE* In a *SHAREDMEM* environment (See **Moab-NUMA Integration Guide** on page 1233 for more information.), Moab will select the node set based on NUMA properties (the smallest feasible node set).
WORSTFIT	Select the largest resource set possible.	This type causes Moab, when choosing a node set, to eliminate sets that do not have all the required resources. From the remaining sets, Moab chooses the set with the greatest amount of resources. This type minimizes fragmentation of smaller resource sets, but increases fragmentation of larger resource sets.

Dynamic example

In this example, a site wants to be able to dynamically specify which *VARATTR* values the node set will be based on. To accomplish this, they could use the following configuration in the `moab.cfg` file:

```
NODESETISOPTIONAL FALSE
NODESETPOLICY     FIRSTOF
NODESETATTRIBUTE  VARATTR
```

Node Set Attribute

The example's NODESETATTRIBUTE parameter is set to *VARATTR* specifying that the node sets are to be constructed by job VARATTR values that are specified dynamically in the msub command.

Node Set Policy

In the preceding example, the NODESETPOLICY parameter is set to the policy *FIRSTOF* and tells Moab to allocate nodes from the first set that matches specified constraints.

Node Set Constraint Handling

The parameter, NODESETISOPTIONAL, indicates that Moab should not delay the start time of a job if the desired node set is not available but adequate idle resources exist outside of the set. Setting this parameter to *FALSE* will force the job to always run in a complete node set regardless of any start delay this imposes.

msub example

With the configuration (above) set in the `moab.cfg`, Moab is configured for dynamic node sets. You can create node sets dynamically by using the msub -l command. (For more information, see **Resource Manager Extensions** on page 618.) Use the following format:

```
msub -l nodeset=FIRSTOF:VARATTR:<var>[=<value>],...
```

For example, if you wanted to create a dynamic node set for the Provo datacenter:

```
msub -l nodeset=FIRSTOF:VARATTR:datacenter=Provo
```

This command causes Moab to set datacenter=Provo as the node set.

> You can specify more than one VARATTR in the command. For example, if you want to create a dynamic node set for the Provo datacenter and the SaltLake datacenter:
>
> ```
> msub -l nodeset=FIRSTOF:VARATTR:datacenter=Provo:datacenter=SaltLake
> ```

If you specify only `datacenter` (without specifying a value, such as `=Provo`), Moab will look up all possible values (values reported on the node for that VARATTR), and then choose one. So if, for example, you have nodes that have VARATTRs `datacenter=Provo`, `datacenter=SaltLake`, and `datacenter=StGeorge`, then specifying `msub -l nodeset=FIRSTOF:VARATTR:datacenter` will cause the job to run in Provo *or* SaltLake *or* StGeorge.

You should also note that Moab also adds the VARATTR (whether you specify it or if Moab chooses it) to the required attribute (REQATTR) of the job. For example, if you specify `datacenter=Provo` as the VARATTR, `datacenter=Provo` will also be added to the job REQATTR. Likewise, if you specify only `datacenter`, and Moab chooses `datacenter=SaltLake`, then `datacenter=SaltLake` will be added to the job REQATTR.

If you do not request a VARATTR in the nodeset of the msub -l command, the job will run as if it did not use node sets at all, and nothing will be added to its REQATTR.

> If you manually specify a different REQATTR on a job (for example, `datacenter=SaltLake`) from the node set VARATTR (for example, `datacenter=Provo`), the job will never run.

NODESETPLUS

Moab supports additional NodeSet behavior by specifying the NODESETPLUS parameter. Possible values when specifying this parameter are *SPANEVENLY* and *DELAY*.

> Neither *SPANEVENLY* nor *DELAY* will work with multi-req jobs or preemption.

Value	Description
SPANEVENLY	Moab attempts to fit all jobs within one node set, or it spans any number of node sets evenly. When a job specifies a NODESETDELAY, Moab attempts to contain the job within a single node set; if unable to do so, it spans node sets evenly, unless doing so would delay the job beyond the requested **NODESETDELAY**.
DELAY	Moab attempts to schedule the job within a nodeset for the configured NODESETDELAY. If Moab cannot find space for the job to start within **NODESETDELAY** (Moab considers future workload to determine if space will open up in time and might create a future reservation), then Moab schedules the job and ignores the nodeset requirement.

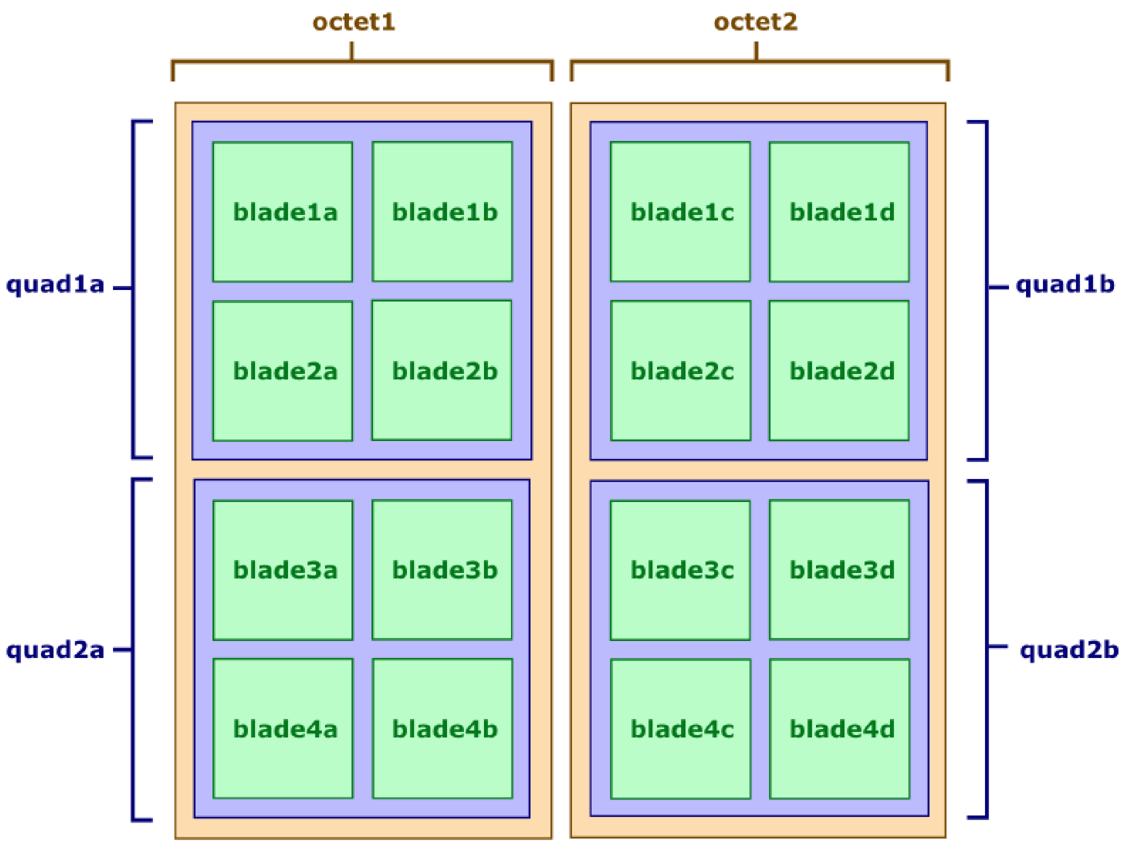
Nested Node Sets

Moab attempts to fit jobs on node sets in the order they are specified in the **NODESETLIST**. You can create nested node sets by listing your node sets in a specific order. Here is an example of a "smallest to largest" nested node set:

```
NODESETPOLICY ONEOF
NODESETATTRIBUTE FEATURE
NODESETISOPTIONAL FALSE
NODESETLIST blade1a,blade1b,blade2a,blade2b,blade3a,
blade3b,blade4a,blade4b,quad1a,quad1b,quad2a,
quad2b,octet1,octet2,sixteen
```

The accompanying cluster would look like this:

Image 3-8: Octet, quad, and blade node sets on a cluster



In this example, Moab tries to fit the job on the nodes in the blade sets first. If that doesn't work, it moves up to the nodes in the quad sets (a set of four blade sets). If the quads are insufficient, it tries the nodes in the octet sets (a set of four quad node sets).

Requesting Node Sets for Job Submission

On a per job basis, each user can specify the equivalent of all parameters except **NODESETDELAY**. As mentioned previously, this is accomplished using the resource manager extensions.

Configuring Node Sets for Classes

Classes can be configured with a default node set. In the configuration file, specify **DEFAULT.NODESET** with the following syntax: `DEFAULT.NODESET=<SETTYPE>:<SETATTR>[:<SETLIST>[,<SETLIST>]...]`. For example, in a heterogeneous cluster with two different types of processors, the following configuration confines jobs assigned to the `amd` class to run on either `ATHLON` or `OPTERON` processors:

```
CLASSCFG[amd] DEFAULT.NODESET=ONEOF:FEATURE:ATHLON,OPTERON
...
```

Related topics

- Resource Manager Extensions
- CLASSCFG
- Partition Overview

Evaluating System Performance - Statistics, Profiling, and Testing

Moab Performance Evaluation Overview

Moab Workload Manager tracks numerous performance statistics for jobs, accounting, users, groups, accounts, classes, QoS, the system, and so forth. These statistics can be accessed through various commands or Moab Cluster Manager/Monitor.

Accounting: Job and System Statistics

Moab provides extensive accounting facilities that allow resource usage to be tracked by resources (compute nodes), jobs, users, and other objects. The accounting facilities may be used in conjunction with, and correlated with, the accounting records provided by the resource and allocation manager.

Moab maintains both raw persistent data and a large number of processed in memory statistics allowing instant summaries of cycle delivery and system utilization. With this information, Moab can assist in accomplishing any of the following tasks:

- Determining cumulative cluster performance over a fixed time frame.
- Graphing changes in cluster utilization and responsiveness over time.
- Identifying which compute resources are most heavily used.
- Charting resource usage distribution among users, groups, projects, and classes.
- Determining allocated resources, responsiveness, and failure conditions for jobs completed in the past.
- Providing real-time statistics updates to external accounting systems.

This section describes how to accomplish each of these tasks using Moab tools and accounting information.

- Accounting Overview
- Real-Time Statistics
- FairShare Usage Statistics

Accounting Overview

Moab provides accounting data correlated to most major objects used within the cluster scheduling environment. These records provide job and reservation accounting, resource accounting, and credential-based accounting.

Job and Reservation Accounting

As each job or reservation completes, Moab creates a complete persistent trace record containing information about who ran, the time frame of all significant events, and what resources were allocated. In addition, actual execution environment, failure reports, requested service levels, and other pieces of key information are also recorded. A complete description of each accounting data field can be found within section Workload Traces.

Resource Accounting

The load on any given node is available historically allowing identification of not only its usage at any point in time, but the actual jobs which were running on it. Moab Cluster Manager can show load information (assuming load is configured as a generic metric), but not the individual jobs that were running on a node at some point in the past. For aggregated, historical statistics covering node usage and availability, the showstats command may be run with the `-n` flag.

Credential Accounting

Current and historical usage for users, groups, account, QoSs, and classes are determined in a manner similar to that available for evaluating nodes. For aggregated, historical statistics covering credential usage and availability, the showstats command may be run with the corresponding credential flag.

If needed, detailed credential accounting can also be enabled globally or on a credential by credential basis. With detailed credential accounting enabled, real-time information regarding per-credential usage over time can be displayed. To enable detailed per credential accounting, the **ENABLEPROFILING** attribute must be specified for credentials that are to be monitored. For example, to track detailed credentials, the following should be used:

```
USERCFG[DEFAULT]      ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]       ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]     ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]     ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]   ENABLEPROFILING=TRUE
```

Credential level profiling operates by maintaining a number of time-based statistical records for each credential. The parameters PROFILECOUNT and PROFILEDURATION control the number and duration of the statistical records.

Real-Time Statistics

Moab provides real-time statistical information about how the machine is running from a scheduling point of view. The showstats command is actually a suite of commands providing detailed information on an overall scheduling basis as well as a per user, group, account and node basis. This command gets its information from in memory statistics that are loaded at scheduler start time from the scheduler checkpoint file. (See Checkpoint/Restart for more information.) This checkpoint file is updated periodically and when the scheduler is shut down allowing statistics to be collected over an extended time frame. At any time, real-time statistics can be reset using the mschedctl -f command.

In addition to the showstats command, the showstats -f command also obtains its information from the in memory statistics and checkpoint file. This command displays a processor-time based matrix of scheduling performance for a wide variety of metrics. Information such as backfill effectiveness or average job queue time can be determined on a job size/duration basis.

FairShare Usage Statistics

Regardless of whether fairshare is enabled, detailed credential based fairshare statistics are maintained. Like job traces, these statistics are stored in the directory pointed to by the STATDIR parameter. Fairshare stats are maintained in a separate statistics file using the format `FS.<EPOCHTIME>` (`FS.982713600`, for example) with one file created per fairshare window. (See the Fairshare Overview for more information.) These files are also flat text and record credential based usage statistics. Information from these files can be seen via the mdiag -f command.

Related topics

- Simulation Overview
- Generic Consumable Resources
- Object Variables
- Generic Event Counters

Testing New Versions and Configurations

- MONITOR Mode
- INTERACTIVE Mode

MONITOR Mode

Moab supports a scheduling mode called *MONITOR*. In this mode, the scheduler initializes, contacts the resource manager and other peer services, and conducts scheduling cycles exactly as it would if running in *NORMAL* or production mode. Job are prioritized, reservations created, policies and limits enforced, and administrator and end-user commands enabled. The key difference is that although live resource management information is loaded, *MONITOR* mode disables Moab's ability to start, preempt, cancel, or otherwise modify jobs or resources. Moab continues to attempt to schedule exactly as it would in *NORMAL* mode but its ability to actually impact the system is disabled. Using this mode, a site can quickly verify correct resource manager configuration and scheduler operation. This mode can also be used to validate new policies and constraints. In fact, Moab can be run in *MONITOR* mode on a production system while another scheduler or even another version of Moab is running on the same system. This unique ability can allow new versions and configurations to be fully tested without any exposure to potential failures and with no cluster downtime.

To run Moab in *MONITOR* mode, simply set the **MODE** attribute of the **SCHEDCFG** parameter to *MONITOR* and start Moab. Normal scheduler commands can be used to evaluate configuration and performance. Diagnostic commands can be used to look for any potential issues. Further, the Moab log file can be used to determine which jobs Moab attempted to start, and which resources Moab attempted to allocate.

If another instance of Moab is running in production and a site administrator wants to evaluate an alternate configuration or new version, this is easily done but care should be taken to avoid conflicts with the primary scheduler. Potential conflicts include statistics files, logs, checkpoint files, and user interface ports. One of the easiest ways to avoid these conflicts is to create a new test directory with its own log and stats subdirectories. The new `moab.cfg` file can be created from scratch or based on the existing *moab.cfg* file already in use. In either case, make certain that the **PORT** attribute of the **SCHEDCFG** parameter differs from that used by the production scheduler by at least two ports. If testing with the production binary executable, the MOABHOMEDIR environment variable should be set to point to the new test directory to prevent Moab from loading the production `moab.cfg` file.

INTERACTIVE Mode

INTERACTIVE mode allows for evaluation of new versions and configurations in a manner different from *MONITOR* mode. Instead of disabling all resource and job control functions, Moab sends the desired change request to the screen and asks for permission to complete it. For example, before starting a job, Moab may print something like the following to the screen:

```
Command:  start job 1139.ncsa.edu on node list test013,test017,test018,test021
Accept:  (y/n) [default: n]?
```

The administrator must specifically accept each command request after verifying it correctly meets desired site policies. Moab then executes the specified command. This mode is highly useful in validating scheduler behavior and can be used until configuration is appropriately tuned and all parties are comfortable with the scheduler's performance. In most cases, sites will want to set the scheduling mode to *NORMAL* after verifying correct behavior.

Related topics

- Testing New Releases and Policies
- Side-by-Side Mode

General Job Administration

Job Holds

Holds and Deferred Jobs

Moab supports job holds applied by users (user holds), administrators (system holds), and resource managers (batch holds). There is also a temporary hold known as a job defer.

User Holds

User holds are very straightforward. Many, if not most, resource managers provide interfaces by which users can place a hold on their own job that tells the scheduler not to run the job while the hold is in place. Users may use this capability because the job's data is not yet ready, or they want to be present when the job runs to monitor results. Such user holds are created by, and under the control of a non-privileged user and may be removed at any time by that user. As would be expected, users can only place holds on their jobs. Jobs with a user hold in place will have a Moab state of `Hold` or `UserHold` depending on the resource manager being used.

System Holds

The system hold is put in place by a system administrator either manually or by way of an automated tool. As with all holds, the job is not allowed to run so long as this hold is in place. A batch administrator can place and release system holds on any job regardless of job ownership. However, unlike a user hold, normal users cannot release a system hold even on their own jobs. System holds are often used during system maintenance and to prevent particular jobs from running in accordance with current system needs. Jobs with a system hold in place will have a Moab state of `Hold` or `SystemHold` depending on the resource manager being used.

Batch Holds

Batch holds are placed on a job by the scheduler itself when it determines that a job cannot run. The reasons for this vary but can be displayed by issuing the checkjob*<JOBID>* command. Possible reasons are included in the following list:

- No Resources — The job requests resources of a type or amount that do not exist on the system.
- System Limits — The job is larger or longer than what is allowed by the specified system policies.

- Bank Failure — The allocations bank is experiencing failures.
- No Allocations — The job requests use of an account that is out of allocations and no fallback account has been specified.
- RM Reject — The resource manager refuses to start the job.
- RM Failure — The resource manager is experiencing failures.
- Policy Violation — The job violates certain throttling policies preventing it from running now and in the future.
- No QOS Access — The job does not have access to the QoS level it requests.

Jobs which are placed in a batch hold will show up within Moab in the state `BatchHold`.

Job Defer

In most cases, a job violating these policies is not placed into a batch hold immediately; rather, it is deferred. The parameter DEFERTIME indicates how long it is deferred. At this time, it is allowed back into the idle queue and again considered for scheduling. If it again is unable to run at that time or at any time in the future, it is again deferred for the timeframe specified by DEFERTIME. A job is released and deferred up to DEFERCOUNT times at which point the scheduler places a batch hold on the job and waits for a system administrator to determine the correct course of action. Deferred jobs have a Moab state of `Deferred`. As with jobs in the BatchHold state, the reason the job was deferred can be determined by use of the checkjob command.

At any time, a job can be released from any hold or deferred state using the releasehold command. The Moab logs should provide detailed information about the cause of any batch hold or job deferral.

> Under Moab, the reason a job is deferred or placed in a batch hold is stored in memory but is not checkpointed. Thus this information is available only until Moab is recycled at which point the checkjob command no longer displays this reason information.

Related topics

- DEFERSTARTCOUNT - number of job start failures allowed before job is deferred

Job Priority Management

Job priority management is controlled via both configured and manual intervention mechanisms.

- Priority Configuration - see Job Prioritization
- Manual Intervention with setspri

Suspend/Resume Handling

When supported by the resource manager, Moab can suspend and resume jobs. A user can suspend his/her own jobs, but only an administrator can resume them. By default, a job is suspended for one minute before it can resume. You can modify this default time using the MINADMINSTIME parameter.

A job must be marked as `suspendable` for Moab to suspend and resume it. To do so, either submit the job with the *suspendable* flag attached to it or configure a credential to pass the flag to its associated jobs. These methods are demonstrated in the examples below:

```
msub -l flags=suspendable
```

```
GROUPCFG[default] JOBFLAGS=SUSPENDABLE
```

Once the job is suspendable, Moab allows you to suspend jobs using the two following methods: (1) manually on the command line and (2) automatically in the `moab.cfg` file.

To manually suspend jobs, use the mjobctl command as demonstrated in the following example:

```
> mjobctl -s job05
```

Moab suspends `job05`, preventing it from running immediately in the job queue.

If you are an administrator and want to resume a job, use the mjobctl command as demonstrated in the following example:

```
> mjobctl -r job05
```

Moab removes `job05` from a suspended state and allows it to run.

You can also configure the Moab preemption policy to suspend and resume jobs automatically by setting the PREEMPTPOLICY parameter to *SUSPEND*. A sample Moab configuration looks like this:

```
PREEMPTPOLICY SUSPEND
...
USERCFG[tom] JOBFLAGS=SUSPENDABLE
```

Moab suspends jobs submitted by user *tom* if necessary to make resources available for jobs with higher priority.

> If your resource manager has a native interface, you must configure JOBSUSPENDURL to suspend and resume jobs.

For more information about suspending and resuming jobs in Moab, see the following sections:

- manual preemption with the mjobctl command
- Job preemption

Checkpoint/Restart Facilities

Checkpointing records the state of a job, allowing for it to restart later without interruption to the job's execution. Checkpointing can be performed manually, as the result of triggers or events, or in conjunction with various QoS policies.

Moab's ability to checkpoint is dependent upon both the cluster's resource manager and operating system. In most cases, two types of checkpoint are enabled, including (1) checkpoint and continue and (2) checkpoint and terminate. While either checkpointing method can be activated using the mjobctl command, only the checkpoint and terminate type is used by internal scheduling and event managements facilities.

Checkpointing behavior can be configured on a per-resource manager basis using various attributes of the RMCFG parameter.

Related topics

- Job Preemption Overview
- PREEMPTPOLICY Parameter
- Resource Manager CHECKPOINTSIG Attribute
- Resource Manager CHECKPOINTTIMEOUT Attribute

Job Dependencies

Basic Job Dependency Support

By default, basic single step job dependencies are supported through completed/failed step evaluation. Basic dependency support does not require special configuration and is activated by default. Dependent jobs are only supported through a resource manager and therefore submission methods depend upon the specific resource manager being used. For the TORQUE qsub command, the semantics listed in the section below can be used with the `-W x=depend=<STRING>` or `-W depend=<STRING>` flag; for the Moab msub command, the `-l depend=<STRING>` or `-W x=depend=<STRING>` flag. For other resource managers, consult the resource manager specific documentation.

> If you are submitting a job with a dependency in a grid environment via msub, you need to use the `-l` option instead of `-W`. You must always use `-W depend=` or `-W x=depend=` for qsub dependencies.

Job Dependency Syntax

Dependency	Format	Description
after	`after:<job>[:<job>]...`	Job may start at any time after specified jobs have started execution.
afterany	`afterany:<job>[:<job>]...`	Job may start at any time after all specified jobs have completed regardless of completion status.

Dependency	Format	Description
afterok	`afterok:<job>[:<job>]...`	Job may be start at any time after all specified jobs have successfully completed.
afternotok	`afternotok:<job>[:<job>]...`	Job may start at any time after all specified jobs have completed unsuccessfully.
before	`before:<job>[:<job>]...`	Job may start at any time before specified jobs have started execution.
beforeany	`beforeany:<job>[:<job>]...`	Job may start at any time before all specified jobs have completed regardless of completion status.
beforeok	`beforeok:<job>[:<job>]...`	Job may start at any time before all specified jobs have successfully completed.
beforenotok	`beforenotok:<job>[:<job>]...`	Job may start at any time before any specified jobs have completed unsuccessfully.
on	`on:<count>`	Job may start after *<count>* dependencies on other jobs have been satisfied.
synccount	`synccount:<count>`	Job is the first in a set of jobs to be executed at the same time. *<count>* is the number of additional jobs in the set, which can be up to 5. synccount is valid for single-request jobs with TORQUE as the resource manager.
syncwith	`syncwith:<job>`	Job is an additional member of a set of jobs to be executed at the same time. Moab supports up to 5 jobs. syncwith is valid for single-request jobs with TORQUE as the resource manager.

`<job>={JOBNAME.jobname|jobid}`

When using JobName dependencies, prepend "JOBNAME." to avoid ambiguity.

The `before` dependencies do not work with jobs submitted with msub; they work only with qsub.

Any of the dependencies containing `before` must be used in conjunction with the `on` dependency. So, if job A must run before job B, job B must be submitted with *depend=on:1*, as well as job A having *depend=before:A*. This means job B cannot run until one dependency of another job on job B has been fulfilled. This prevents job B from running until job A can be successfully submitted.

When you submit a dependency job and the dependency is not met, the job will remain idle in the queue indefinitely. To configure Moab to automatically cancel these failed dependency jobs, set the **CANCELFAILEDDEPENDENCYJOBS** on page 1368 scheduler flag.

Related topics

- Job Deadlines

Job Defaults and Per Job Limits

Job Defaults

Job defaults can be specified on a per queue basis. These defaults are specified using the CLASSCFG parameter. The following table shows the applicable attributes:

Attribute	Format	Example
DEFAULT.FEATURES	comma-delimited list of node features	`CLASSCFG[batch] DEFAULT.FEATURES=fast,io` *Jobs submitted to class batch will request nodes features fast and io.*
DEFAULT.WCLIMIT	`[[[DD:]HH:]MM:]SS`	`CLASSCFG[batch] DEFAULT.WCLIMIT=1:00:00` *Jobs submitted to class batch will request one hour of walltime by default.*

Per Job Maximum Limits

Job maximum limits can be specified on a per queue basis. These defaults are specified using the CLASSCFG parameter. The following table shows the applicable attributes:

Attribute	Format	Example
MAX.WCLIMIT	`[[[DD:]HH:]MM:]SS`	`CLASSCFG[batch] MAX.WCLIMIT=1:00:00` *Jobs submitted to class batch can request no more than one hour of walltime.*

Per Job Minimum Limits

Furthermore, minimum job defaults can be specified with the CLASSCFG parameter. The following table shows the applicable attributes:

Attribute	Format	Example
MIN.PROC	*<integer>*	`CLASSCFG[batch] MIN.PROC=10` *Jobs submitted to class batch can request no less than ten processors.*

Related topics

- Usage-based Limits

General Job Policies

- Multi-Node Support
- Multi-Req Support
- Job Size Policy
- Malleable Job Support
- Enabling Job User Proxy

There are a number of configurable policies that help control advanced job functions. These policies help determine allowable job sizes and structures.

Multi-Node Support

You can configure the ability to allocate resources from multiple nodes to a job with the MAX.NODE limit.

Multi-Req Support

Jobs can specify multiple types of resources for allocation. For example, a job could request 4 nodes with 256 MB of memory and 8 nodes with feature `fast` present.

Resources specified in a multi-req job are delimited with a plus sign (+).

> Neither *SPANEVENLY* nor *DELAY* values of the NODESETPLUS parameter will work with multi-req jobs or preemption.

Example 3-129:

```
-l nodes=4:ppn=1+10:ppn=5+2:ppn=2
```

*This example requests 4 nodes with 1 proc each, 10 nodes with 5 procs each, and 2 nodes with 2 procs each. The total number of processors requested is (4*1) + (10*5) + (2*2), or 58 processors.*

Example 3-130:

```
-l nodes=15+1:ppn=4
```

The job submitted in this example requests a total of 16 nodes. 15 of these nodes have no specific requirements, but the remaining node must have 4 processors.

Example 3-131:

```
-l nodes=3:fast+1:io
```

The job requests a total of 4 nodes: 3 nodes with the fast feature and 1 node with the io feature.

Job Size Policy

Moab allows jobs to request resource ranges. Using this range information, the scheduler is able to maximize the amount of resources available to the job while minimizing the amount of time the job is blocked waiting for resources. The JOBSIZEPOLICY parameter can be used to set this behavior according to local site needs.

Job resource ranges may only be specified when using a local queue as described in the Using a Local Queue section.

Malleable Job Support

A job can specify whether it is able to use more processors or less processors and what effect, if any, that has on its wallclock time. For example, a job may run for 10 minutes on 1 processor, 5 minutes on 2 processors and 3 minutes on 3 processors. When a job is submitted with a task request list attached, Moab determines which task request fits best and molds the job based on its specifications. To submit a job with a task request list and allow Moab to mold it based on the current scheduler environment, use the TRL flag in the Resource Manager Extension.

Enabling Job User Proxy

By default, user proxying is disabled. To be enabled, it must be authorized using the **PROXYLIST** attribute of the USERCFG parameter. This parameter can be specified either as a comma-delimited list of users or as the keyword *validate*. If the keyword *validate* is specified, the RMCFG attribute **JOBVALIDATEURL** should be set and used to confirm that the job's owner can proxy to the job's execution user. An example script performing this check for ssh-based systems is provided in the `tools` directory (See Job Validate Tool Overview.).

For some resource managers (RM), proxying must also be enabled at the RM level. The following example shows how ssh-based proxying can be accomplished in a Moab+TORQUE with SSH environment.

To validate proxy users, Moab must be running as root.

Example 3-132: SSH Proxy Settings

```
USERCFG[DEFAULT] PROXYLIST=validate
RMCFG[base]  TYPE=<resource manager>
JOBVALIDATEURL=exec://$HOME/tools/job.validate.sshproxy.pl
```

```
> qmgr -c 's s allow_proxy_user=true'
> su - testuser
> qsub -I -u testuser2
qsub: waiting for job 533.igt.org to start
qsub: job 533.igt.org ready
testuser2@igt:~$
```

In this example, the validate tool, '`job.validate.sshproxy.pl`', can verify proxying is allowed by becoming the submit user and determining if the submit user can achieve passwordless access to the specified execution user. However, site-specific tools can use any method to determine proxy access including a flat file look-up, database lookup, querying of an information service such as NIS or LDAP, or other local or remote tests. For example, if proxy validation is required but end-user accounts are not available on the management node running Moab, the job validate service could perform the validation test on a representative remote host such as a login host.

This feature supports qsub only.

The job validate tool is highly flexible allowing any combination of job attributes to be evaluated and tested using either local or remote validation tests. The validate tool allows not only pass/fail responses but also allows the job to be modified, or rejected in a custom manner depending on the site or the nature of the failure.

Related topics

- Usage Limits

Using a Local Queue

Moab allows jobs to be submitted directly to the scheduler. With a local queue, Moab is able to directly manage the job or translate it for resubmission to a standard resource manager queue. There are multiple advantages to using a local queue:

- Jobs may be translated from one resource manager job submission language to another (such as submitting a PBS job and running it on an LSF cluster).
- Jobs may be migrated from one local resource manager to another.
- Jobs may be migrated to remote systems using Moab peer-to-peer functionality.
- Jobs may be dynamically modified and optimized by Moab to improve response time and system utilization.
- Jobs may be dynamically modified to account for system hardware failures or other issues.
- Jobs may be dynamically modified to conform to site policies and constraints.
- Grid jobs are supported.

Local Queue Configuration

A local queue is configured just like a standard resource manager queue. It may have defaults, limits, resource mapping, and credential access constraints. The following table describes the most common settings:

Default queue	
Format	`RMCFG[internal] DEFAULTCLASS=<CLASSID>`
Description	The job class/queue assigned to the job if one is not explicitly requested by the submitter. All jobs submitted directly to Moab are initially received by the pseudo-resource manager *internal*. Therefore, default queue configuration may only be applied to it.
Example	`RMCFG[internal] DEFAULTCLASS=batch`

Class default resource requirements	
Format	`CLASSCFG[<CLASSID>] DEFAULT.FEATURES=<X> CLASSCFG[<CLASSID>] DEFAULT.MEM=<X> CLASSCFG[<CLASSID>] DEFAULT.NODE=<X> CLASSCFG[<CLASSID>] DEFAULT.NODESET=<X> CLASSCFG[<CLASSID>] DEFAULT.PROC=<X> CLASSCFG [<CLASSID>] DEFAULT.WCLIMIT=<X>`
Description	The settings assigned to the job if not explicitly set by the submitter. Default values are available for node features, per task memory, node count, nodeset configuration, processor count, and wallclock limit.
Example	`CLASSCFG[batch] DEFAULT.WCLIMIT=4 DEFAULT.FEATURES=matlab` or `CLASSCFG[batch] DEFAULT.WCLIMIT=4` `CLASSCFG[batch] DEFAULT.FEATURES=matlab`

Class maximum resource limits	
Format	`CLASSCFG[<CLASSID>] MAX.FEATURES=<X> CLASSCFG[<CLASSID>] MAX.NODE=<X> CLASSCFG[<CLASSID>] MAX.PROC=<X> CLASSCFG[<CLASSID>] MAX.WCLIMIT=<X>`

Class maximum resource limits	
Description	The maximum node features, node count, processor count, and wallclock limit allowed for a job submitted to the class/queue. If these limits are not satisfied, the job is not accepted and the submit request fails. **MAX.FEATURES** indicates that only the listed features may be requested by a job.
Example	`CLASSCFG[smalljob] MAX.PROC=4 MAX.FEATURES=slow,matlab` or `CLASSCFG[smalljob] MAX.PROC=4` `CLASSCFG[smalljob] MAX.FEATURES=slow,matlab`

Class minimum resource limits	
Format	CLASSCFG[*<CLASSID>*] MIN.FEATURES=*<X>* CLASSCFG[*<CLASSID>*] MIN.NODE=*<X>* CLASSCFG[*<CLASSID>*] MIN.PROC=*<X>* CLASSCFG[*<CLASSID>*] MIN.WCLIMIT=*<X>*
Description	The minimum node features, node count, processor count, and wallclock limit allowed for a job submitted to the class/queue. If these limits are not satisfied, the job is not accepted and the submit request fails. **MIN.FEATURES** indicates that only the listed features may be requested by a job.
Example	`CLASSCFG[bigjob] MIN.PROC=4 MIN.WCLIMIT=1:00:00` or `CLASSCFG[bigjob] MIN.PROC=4` `CLASSCFG[bigjob] MIN.WCLIMIT=1:00:00`

Class access	
Format	CLASSCFG[<CLASSID>] REQUIREDUSERLIST=<USERID>[,<USERID>]...
Description	The list of users who may submit jobs to the queue.
Example	`CLASSCFG[math] REQUIREDUSERLIST=john,steve`

Available resources	
Format	CLASSCFG[*<CLASSID>*] HOSTLIST=*<HOSTID>*[,*<HOSTID>*]...
Description	The list of nodes that jobs in the queue may use.
Example	`CLASSCFG[special] HOSTLIST=node001,node003,node13`

Class mapping between multiple sites is described in the section on Moab grid facilities.

If a job is submitted directly to the resource manager used by the local queue, the class default resource requirements are not applied. Also, if the job violates a local queue limitation, the job is accepted by the resource manager, but placed in the Blocked state.

Job Deadlines

Deadline Overview

Job deadlines may be specified on a per job and per credential basis and are also supported using both absolute and QoS based specifications. A job requesting a deadline is first evaluated to determine if the deadline is acceptable. If so, Moab adds it to the list of deadline jobs and allocates resources to guarantee that all accepted deadline jobs are able to complete on or before their requested deadline. Once the scheduler confirms that all deadlines can be satisfied, it then optimizes resource allocation (in priority order) attempting to execute all jobs at the earliest possible time.

Setting Job Deadlines via QoS

Two types of job deadlines exist in Moab. The priority-based deadline linearly increases a job's priority as its deadline approaches (See **Deadline (DEADLINE) Subcomponent** on page 391 for more information). The QoS method allows you to set a job completion time on job submission if, and only if, it requests and is allowed to access a QoS with the *DEADLINE* QFLAG set. This method is more powerful than the priority method, because Moab will attempt to make a reservation for the job as soon as the job enters the queue in order to meet the deadline, essentially bumping it to the front of the queue.

When a job is submitted to a QoS with the *DEADLINE* flag set, the job's `-l deadline` attribute is honored. If such QoS access is not available, or if resources do not exist at job submission time to allow the deadline to be satisfied, the job's deadline request is ignored.

Two methods exist for setting deadlines with a QoS:

- Submitting a job to a deadline-enabled QoS and specifying a deadline using msub -l.
- Submitting a job to a deadline-enabled QoS with a QTTARGET specified.

Setting Job Deadlines at Job Submission

This method of setting a job deadline allows you to specify a job deadline as you submit the job. You can set the deadline as either an exact date and time or as an amount of time after job submission (i.e. three hours after submission).

To specify a deadline on job submission

1. In `moab.cfg`, reate a QoS with the *DEADLINE* flag enabled.

```
...
QOSCFG[special] QFLAGS=DEADLINE
```

Jobs requesting the QoS `special` may submit jobs with a deadline that Moab will honor.

2. Submit a job to the QoS and set a deadline. This can be either absolute or relative.

   a. For an absolute deadline, use the format `hh:mm:ss_mm/dd/yy`. The following configuration sets a deadline for a job to finish by 8 a.m. on March 15th, 2013.

```
msub -l qos=special deadline=08:00:00_03/15/13 job.sh
```

   The job must finish running by 8 A.M. on March 15, 2013.

   b. For a relative deadline, or the completion deadline of the job relative to its submission time, use the time format `[[[DD:]HH:]MM:]SS`.

```
msub -l qos=special deadline=5:00:00 job.sh
```

   The job's deadline is 5 hours after its submission.

Submitting a Job to a QoS with a Preconfigured Deadline

You may also set a relative job deadline by limiting the job's queue time. This method allows you to pre-configure the deadline rather than giving the power to specify a deadline to the user submitting the job. For jobs requesting these QoSes, Moab identifies and sets job deadlines to satisfy the corresponding response time targets.

To submit a job to a QoS with a preconfigured deadline

1. In `moab.cfg`, create a QoS with both the *DEADLINE* **QFLAG** and a response time target (**QTTARGET**). The **QTTARGET** is the maximum amount of time that Moab should allow the job to be idle in the

queue.

```
...
QOSCFG[special2] QFLAGS=DEADLINE QTTARGET=1:00:00
```

Given this configuration, a job requesting QoS special2 must spend a maximum of one hour in the queue.

2. Submit a job requesting the `special2` quality of service.

```
msub -l qos=special2 walltime=2:00:00 job.sh
```

This two-hour job has a completion time deadline set to three hours after its submission (one hour of target queue time and two hours of run time).

Job Termination Date

In addition to job completion targets, jobs may also be submitted with a TERMTIME attribute. The scheduler attempts to complete the job prior to the termination date, but if it is unsuccessful, it will terminate (cancel) the job once the termination date is reached.

Conflict Policies

The specific policy can be configured using the DEADLINEPOLICY parameter. Moab does not have a default policy for this parameter.

Policy	Description
CANCEL	The job is canceled and the user is notified that the deadline could not be satisfied.
HOLD	The job has a batch hold placed on it indefinitely. The administrator can then decide what action to take.
RETRY	The job continually retries each iteration to meet its deadline; note that when used with `QTTARGET` the job's deadline continues to slide with relative time.
IGNORE	The job has its request ignored and is scheduled as normal.

Deadline scheduling may not function properly with per partition scheduling enabled. Check that **PARALLOCATIONPOLICY** is disabled to ensure **DEADLINEPOLICY** will work correctly.

Related topics

- QoS Facilities
- Job Submission Eligible Start Time constraints

Job Arrays

Job Array Overview

You can submit an array of jobs to Moab via the msub command. Array jobs are an easy way to submit many sub-jobs that perform the same work using the same script, but operate on different sets of data. Sub-jobs are the jobs created by an array job and are identified by the array job ID and an index; for example, if `235[1]` is an identifier, the number `235` is a job array ID, and `1` is the sub-job.

Sub-jobs of an array are executed in sub-job index order.

> Moab job arrays are different from TORQUE job arrays.

Enabling Job Arrays

To enable job arrays, include the ENABLEJOBARRAYS parameter in the Moab configuration file (`moab.cfg`).

Sub-job Definitions

Like a normal job, an array job submits a job script, but it additionally has a start index (`sidx`) and an end index (`eidx`); array jobs also have increment (`incr`) values, which Moab uses to create sub-jobs, all executing the same script. The model for sub-job creation follows the formula of end index minus start index plus increment divided by the increment value: (`eidx` - `sidx` + `incr`) / `incr`.

To illustrate, suppose an array job has a start index of 1, an end index of 100, and an increment of 1. This is an array job that creates (100 - 1 + 1) / 1 = 100 sub-jobs with indexes of 1, 2, 3, ..., 100. An increment of 2 produces (100 - 1 + 2) / 2 = 50 sub-jobs with indexes of 1, 3, 5, ..., 99. An increment of 2 with a start index of 2 produces (100 - 2 + 2) / 2 = 50 sub-jobs with indexes of 2, 4, 6, ..., 100. Again, sub-jobs are jobs in their own right that have a slightly different job naming convention `jobID[subJobIndex]` (e.g. `mycluster.45[37]` or `45[37]`).

Using Environment Variables to Specify Array Index Values

The script can use an environment variable to obtain the array index value to form data file and/or directory names unique to an array job's particular sub-job. The following two environment variables are supplied so job scripts can recognize what index in the array they are in; use the msub command with the -V option to pass the environment parameters to the resource manager, or include the parameters in a job script; for example: `#PBS -V MOAB_JOBARRAYRANGE`.

Environment Parameter	Description
MOAB_ JOBARRAYINDEX	Used to create dataset file names, directory names, and so forth, when splitting up a single problem into multiple jobs. For example, a user may split up a problem into 20 separate jobs, each with its own input and output data files whose names contain the numbers 1-20. To illustrate, assume a user submits the 20 sub-jobs using two msub commands; one to submit the ten even-numbered jobs and one to submit the ten odd-numbered jobs. `msub -t job1.[1-20:2]` `msub -t job2.[2-20:2]` The `MOAB_JOBARRAYINDEX` environment variable value would populate each of the two job arrays' ten sub-jobs as `1`, `3`, `5`, `7`, `9`, `11`, `13`, `15`, `17` and `19` for the first array job's ten sub-jobs, and `2`, `4`, `6`, `8`, `10`, `12`, `14`, `16`, `18`, and `20` for the second array job's ten sub-jobs.
MOAB_ JOBARRAYRANGE	The count of jobs in the array.

Control

Users can control individual sub-jobs in the same manner as normal jobs. In addition, an array job represents its group of sub-jobs and any user or administrator commands performed on an array job apply to its sub-jobs; for example, the command canceljob *<arrayJobId>* cancels all sub-jobs that belong to the array job. For more information about job control, see the documentation for the mjobctl command.

Reporting

In the first example below, the parts unique to array subjobs are in red.

```
$ checkjob -v Moab.1[1]
job Moab.1[1]

AName: Moab
State: Running
Creds:  user:user1  group:usergroup1
WallTime:   00:00:17 of 8:20:00
SubmitTime: Thu Nov  4 11:50:03
(Time Queued  Total: 00:00:00  Eligible:   INFINITY)
StartTime: Thu Nov  4 11:50:03
Total Requested Tasks: 1
Req[0]  TaskCount: 1  Partition: base
Average Utilized Procs: 0.96
NodeCount:  1
Allocated Nodes:
[node010:1]

Job Group:         Moab.1
Parent Array ID:  Moab.1
Array Index:       1
Array Range:       10
SystemID:   Moab
SystemJID:  Moab.1[1]
Task Distribution: node010
IWD:            /home/user1
UMask:          0000
Executable:     /opt/moab/spool/moab.job.3CvNjl
StartCount:     1
Partition List: base
SrcRM:          internal  DstRM: base  DstRMJID: Moab.1[1]
Flags:          ARRAYJOB,GLOBALQUEUE
StartPriority:  1
PE:             1.00
Reservation 'Moab.1[1]' (-00:00:19 -> 8:19:41  Duration: 8:20:00)
```

If the array range is not provided, the output displays all the jobs in the array.

```
$ checkjob -v Moab.1
job Moab.1

AName: Moab
Job Array Info:
  Name: Moab.1
  1 : Moab.1[1] : Running
  2 : Moab.1[2] : Running
  3 : Moab.1[3] : Running
  4 : Moab.1[4] : Running
  5 : Moab.1[5] : Running
  6 : Moab.1[6] : Running
  7 : Moab.1[7] : Running
  8 : Moab.1[8] : Running
  9 : Moab.1[9] : Running
  10 : Moab.1[10] : Running
  11 : Moab.1[11] : Running
  12 : Moab.1[12] : Running
  13 : Moab.1[13] : Running
  14 : Moab.1[14] : Running
  15 : Moab.1[15] : Running
  16 : Moab.1[16] : Running
  17 : Moab.1[17] : Running
  18 : Moab.1[18] : Running
  19 : Moab.1[19] : Running
  20 : Moab.1[20] : Running
  Totals:
    Active:   20
    Idle:     0
    Complete: 0
```

You can also use showq. This displays the array master job with a count of how many sub-jobs are in each queue.

```
$ showq

active jobs------------------------
JOBID              USERNAME      STATE  PROCS   REMAINING            STARTTIME

Moab.1(5)           aesplin    Running     5    00:52:41  Thu Jun 23 17:05:56
Moab.2(1)           aesplin    Running     1    00:53:41  Thu Jun 23 17:06:56

6 active jobs              6 of 6 processors in use by local jobs (100.00%)
1 of 1 nodes active      (100.00%)

eligible jobs----------------------
JOBID              USERNAME      STATE  PROCS     WCLIMIT            QUEUETIME

Moab.2(4)           aesplin       Idle     4     1:00:00  Thu Jun 23 17:06:56

4 eligible jobs

blocked jobs-----------------------
JOBID              USERNAME      STATE  PROCS     WCLIMIT            QUEUETIME

Moab.2(1)           aesplin    Blocked     1     1:00:00  Thu Jun 23 17:06:56

1 blocked job

Total jobs:  11
```

`Moab.1` has five sub-jobs running. `Moab.2` has one sub-job running, four waiting to run, and one that is currently blocked.

Job Array Cancellation Policies

Job arrays can be canceled based on the success or failure of the first sub-job, the first success or failure of any sub-job, or if any sub-job exits with a specified exit code. The job array cancellation policies are:

Cancel Policy	Description	Exclusivity
CancelOnFirstFailure	Cancels the job array if the first sub-job (**JOBARRAYINDEX** = *1*) fails. `> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnFirstFailure`	Mutually exclusive
CancelOnFirstSuccess	Cancels the job array if the first sub-job (**JOBARRAYINDEX** = *1*) succeeds. `> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnFirstSuccess`	
CancelOnAnyFailure	Cancels the job array if any sub-job fails. `> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnAnyFailure`	

Cancel Policy	Description	Exclusivity
CancelOnAnySuccess	Cancels the job array if any sub-job succeeds. `> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnAnySuccess`	
CancelOnExitCode	Cancels the job array if any sub-job returns the specified exit code. `> msub -t myarray[1-1000%50] -l ...,flags=CancelOnExitCode:<error code list>` The syntax for the error code list are ranges specified with a dash and individual codes delimited by a plus (+) sign, such as: 1-4+9+15 Exit codes 1-387 are accepted.	

Up to two cancellation polices can be specified for an array and the two policies must be delimited by a colon (`:`). The two "first sub-job" policies are mutually exclusive, as are the three "any sub-job" policies. You can use either "first sub-job" policy with one of the "any sub-job" policies, as shown in this example:

```
> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnFirstFailure:CancelOnExitCode:3-7+11
```

Examples

Operations can be performed on individual jobs, a selection of jobs in a job array, or on the entire array.

Submitting Job Arrays

The syntax for submitting job arrays is: `msub -t [`*`<jobname>`*`]`*`<indexlist>`*`[%`*`<limit>`*`] arrayscript.sh`

The *<jobname>* and *<limit>* are optional. The jobname does not override the `jobID` Moab assigns to the array. When submitting an array with a jobname, Moab returns the `jobID`, which is the scheduler name followed by a unique ID.

For example, if the scheduler name in `moab.cfg` is *Moab* (`SCHEDCFG[Moab]`), submitting an array with a jobname responds like this:

```
> msub -t myarray[1-10] job.sh

Moab.6
```

To specify that only a certain number of sub-jobs in the array can run at a time, use the percent sign (`%`) delimiter. In this example, only five sub-jobs in the array can run at a time:

```
> msub -t myarray[1-1000]%5
```

To submit a specific set of array sub-jobs, use the comma delimiter in the array index list:

```
> msub -t myarray[1,2,3,4]
> msub -t myarray[1-5,7,10]
```

You can use the checkjob command on either the `jobID` or the `jobname` you specified.

```
> msub -t myarray[1-2] job.sh

Moab.10

$ checkjob -v myarray
  job Moab.10

AName: myarray
Job Array Info:
   Name: Moab.10
   1 : Moab.10[1] : Running
   2 : Moab.10[2] : Running

   Sub-jobs:           2
     Active:           2 ( 100.0% )
     Eligible:         0 ( 0.0% )
     Blocked:          0 ( 0.0% )
     Completed:        0 ( 0.0% )

State: Idle
Creds:  user:tuser1  group:tgroup1
WallTime:   00:00:00 of 99:23:59:59
SubmitTime: Thu Jun  2 16:37:17
   (Time Queued  Total: 00:00:33  Eligible: 00:00:00)

Total Requested Tasks: 1

Req[0]  TaskCount: 1  Partition: ALL
```

To submit a job with a step size, use a colon in the array range and specify how many jobs to step. In the example below, a step size of 2 is requested. The sub-jobs will be numbered according to the step size inside the index limit. The array master job name will be the same as explained above.

```
$ msub -t myarray[2-10:2] job.sh

job Moab.15

$ checkjob -v myarray #or you could use 'checkjob -v Moab.15'
job Moab.15

AName: myarray
Job Array Info:
   Name: Moab.15
   2 : Moab.15[2] : Running
   4 : Moab.15[4] : Running
   6 : Moab.15[6] : Running
   8 : Moab.15[8] : Running
   10 : Moab.15[10] : Running

   Sub-jobs:           5
     Active:           5 ( 100.0% )
     Eligible:         0 ( 0.0% )
     Blocked:          0 ( 0.0% )
     Completed:        0 ( 0.0% )

State: Idle
Creds:  user:tuser1  group:tgroup1
WallTime:   00:00:00 of 99:23:59:59
SubmitTime: Thu Jun  2 16:37:17
   (Time Queued  Total: 00:00:33  Eligible: 00:00:00)

Total Requested Tasks: 1

Req[0]  TaskCount: 1  Partition: ALL
```

Related topics

- Moab Workload Manager for Grids
- Job Dependencies

General Node Administration

Overview

Moab has a very flexible and generalized definition of a node. This flexible definition, together with the fact that Moab must inter-operate with many resource managers of varying capacities, requires that Moab must possess a complete set of mechanisms for managing nodes that in some cases may be redundant with resource manager facilities.

Resource Manager Specified 'Opaque' Attributes

Many resource managers support the concept of opaque node attributes, allowing a site to assign arbitrary strings to a node. These strings are opaque in the sense that the resource manager passes them along to the scheduler without assigning any meaning to them. Nodes possessing these opaque attributes can then be requested by various jobs. Using certain Moab parameters, sites can assign a meaning within Moab to these opaque node attributes and extract specific node information. For example, setting the parameter FEATUREPROCSPEEDHEADER xps causes a node with the opaque string `xps950` to be assigned a processor speed of 950 MHz within Moab.

Scheduler Specified Default Node Attributes

Some default node attributes can be assigned on a rack or partition basis. In addition, many node attributes can be specified globally by configuring the *DEFAULT* node template using the NODECFG parameter (i.e., `NODECFG[DEFAULT] PROCSPEED=3200`). Unless explicitly specified otherwise, nodes inherit node attributes from the associated rack or partition or from the default node template. See the Partition Overview for more information.

Scheduler Specified Node Attributes

The **NODECFG** parameter also allows direct per-node specification of virtually all node attributes supported via other mechanisms and also provides a number of additional attributes not found elsewhere. For example, a site administrator may want to specify something like the following:

```
NODECFG[node031] MAXJOB=2 PROCSPEED=600 PARTITION=small
```

> These approaches may be mixed and matched according to the site's local needs. Precedence for the approaches generally follows the order listed earlier in cases where conflicting node configuration information is specified through one or more mechanisms.

Node Location

Nodes can be assigned three types of location information based on partitions, racks, and queues.

- Partitions
- Racks
- Queues
    - TORQUE/OpenPBS Queue to Node Mapping

- Node Selection/Specification

Partitions

The first form of location assignment, the partition, allows nodes to be grouped according to physical resource constraints or policy needs. By default, jobs are not allowed to span more than one partition so partition boundaries are often valuable if an underlying network topology make certain resource allocations undesirable. Additionally, per-partition policies can be specified to grant control over how scheduling is handled on a partition by partition basis. See the Partition Overview for more information.

Racks

Rack-based location information is orthogonal to the partition based configuration and is mainly an organizational construct. In general rack based location usage, a node is assigned both a rack and a slot number. This approach has descended from the IBM SP2 organizational approach in which a rack can contain any number of slots but typically contains between 1 and 99. Using the rack and slot number combo, individual compute nodes can be grouped and displayed in a more ordered manner in certain Moab commands (i.e., showstate). Currently, rack information can only be specified directly by the system via the SDR interface on SP2/Loadleveler systems. In all other systems, this information must be specified using an information service or specified manually using the RACK, SLOT, and SIZE attributes of the NODECFG parameter.

> Sites may arbitrarily assign nodes to racks and rack slots without impacting scheduling behavior. Neither rack numbers nor rack slot numbers need to be contiguous; their use is simply for convenience purposes in displaying and analyzing compute resources.

Example 3-133:

```
NODECFG[node024] RACK=1 SLOT=1
NODECFG[node025] RACK=1 SLOT=2
NODECFG[node026] RACK=2 SLOT=1 PARTITION=special
...
```

When specifying node and rack information, slot values must be in the range of 1 to 99, and racks must be in the range of 1 to 399.

Queues

Some resource managers allow queues (or classes) to be defined and then associated with a subset of available compute resources. With systems such as Loadleveler or PBSPro these queue to node mappings are automatically detected. On resource managers that do not provide this service, Moab provides alternative mechanisms for enabling this feature.

TORQUE/OpenPBS Queue to Node Mapping

Under TORQUE, queue to node mapping can be accomplished by using the qmgr command to set the queue acl_hosts parameter to the mapping hostlist desired. Further, the `acl_host_enable` parameter should be set to `False`.

> Setting `acl_hosts` and then setting `acl_host_enable` to `True` constrains the list of hosts from which jobs may be submitted to the queue.

The following example highlights this process and maps the queue `debug` to the nodes `host14` through .

```
> qmgr
Max open servers: 4
Qmgr: set queue debug acl_hosts = "host14,host15,host16,host17"
Qmgr: set queue debug acl_host_enable = false
Qmgr: quit
```

> All queues that do not have `acl_hosts` specified are global; that is, they show up on every node. To constrain these queues to a subset of nodes, each queue requires its own `acl_hosts` parameter setting.

Node Selection

When selecting or specifying nodes either via command line tools or via configuration file based lists, Moab offers three types of node expressions that can be based on node lists, exact lists, node ranges, or regular expressions.

Node Lists

Node lists can be specified as one or more comma or whitespace delimited node IDs. Specified node IDs can be based on either short or fully qualified hostnames. Each element will be interpreted as a regular expression.

```
SRCFG[basic]  HOSTLIST=cl37.icluster,ax45,ax46
...
```

Exact Lists

When Moab receives a list of nodes it will, by default, interpret each element as a regular expression. To disable this and have each element interpreted as a string node name, the `l:` can be used as in the following example:

```
> setres l:n00,n01,n02
```

Node Range

Node lists can be specified as one or more comma or whitespace delimited node ranges. Each node range can be based using either *<STARTINDEX>*-*<ENDINDEX>* or *<HEADER>*[*<STARTINDEX>*-*<ENDINDEX>*] format. To explicitly request a range, the node expression must be preceded with the string `r:` as in the following example:

```
> setres r:37-472,513,516-855
```

When you specify a *<HEADER>* for the range, note that it must only contain alphabetical characters. As always, the range must be numeric.

```
CLASSCFG[long] HOSTLIST=r:anc-b[37-472]
```

> Only one expression is allowed with node ranges.

> By default, Moab attempts to extract a node's node index assuming this information is built into the node's naming convention. If needed, this information can be explicitly specified in the Moab configuration file using NODECFG's **NODEINDEX** attribute, or it can be extracted from alternately formatted node IDs by specifying the NODEIDFORMAT parameter.

Node Regular Expression

Node lists may also be specified as one or more comma or whitespace delimited regular expressions. Each node regular expression must be specified in a format acceptable by the standard C regular expression libraries that allow support for wildcard and other special characters such as the following:

- * (asterisk)
- . (period)
- [ ] (left and right bracket)
- ^ (caret)
- $ (dollar)

Node lists are by default interpreted as a regular expression but can also be explicitly requested with the string *x:* as in the following examples:

```
# select nodes cl30 thru cl55
SRCFG[basic]  HOSTLIST=x:cl[34],cl5[0-5]
...
```

```
# select nodes cl30 thru cl55
SRCFG[basic]  HOSTLIST=cl[34],cl5[0-5]
...
```

> To control node selection search ordering, set the **OBJECTELIST** parameter to one of the following options: exact, range, regex, rangere, or rerange.

Node Attributes

Configurable Node Attributes

Nodes can possess a large number of attributes describing their configuration which are specified using the NODECFG parameter. The majority of these attributes such as operating system or configured network interfaces can only be specified by the direct resource manager interface. However, the number and detail of node attributes varies widely from resource manager to resource manager. Sites often have interest in making scheduling decisions based on scheduling attributes not directly supplied by the resource manager. Configurable node attributes are listed in the following table; click an attribute for more detailed information:

Attribute	Description
ACCESS	Specifies the node access policy that can be one of *SHARED*, *SHAREDONLY*, *SINGLEJOB*, *SINGLETASK*, or *SINGLEUSER*. See Node Access Policies for more details. `NODECFG[node013] ACCESS=singlejob`
ARCH	Specifies the node's processor architecture. `NODECFG[node013] ARCH=opteron`
COMMENT	Allows an organization to annotate a node via the configuration file to indicate special information regarding this node to both users and administrators. The **COMMENT** value may be specified as a quote delimited string as shown in the example that follows. Comment information is visible using checknode, mdiag, Moab Cluster Manager, and Moab Access Portal. `NODECFG[node013] COMMENT="Login Node"`
ENABLEPROFILING	Allows an organization to track node state over time. This information is available using showstats -n. `NODECFG[DEFAULT] ENABLEPROFILING=TRUE`

Attribute	Description
FEATURES	Not all resource managers allow specification of opaque node features (also known as node properties). For these systems, the NODECFG parameter can be used to directly assign a list of node features to individual nodes. To append node features, use `FEATURES=<X>`; to overwrite or remove a node's features, you must update them in your Moab configuration file or resource manager. `NODECFG[node013] FEATURES=gpfs,fastio` *Node `node013` now has features gpfs and fastio in addition to any other features configured in this file or the resource manager.* The total number of supported node features is limited as described in the Adjusting Default Limits section. If supported by the resource manager, the resource manager specific manner of requesting node features/properties within a job may be used. (Within TORQUE, use `qsub -l nodes=<NODECOUNT>:<NODEFEATURE>`.) However, if either not supported within the resource manager or if support is limited, the Moab feature resource manager extension may be used.
FLAGS	Specifies various flags that should be set on the given node. Node flags must be set using the mschedctl -m config command. Do not set node flags in the `moab.cfg` file. Flags set in `moab.cfg` may conflict with settings controlled automatically by resource managers, Moab Web Services. • *globalvars* - The node has variables that may be used by triggers. • *novmmigrations* - Excludes this hypervisor from VM auto-migrations. This means that VMs cannot automatically migrate to or from this hypervisor while this flag is set. `NODECFG[node1] FLAGS=NoVMMigrations` *To allow VMs to resume migrating, remove this flag using `mschedctl -m config 'NODECFG[node1] FLAGS-=NoVMMigrations'` or use a resource manager to unset the flag. Because both Moab and the RM report the novmmigration flag and the RM's setting always overrides the Moab setting, you cannot remove the flag via the Moab command when the RM is reporting it.*

Attribute	Description
GRES	Many resource managers do not allow specification of consumable generic node resources. For these systems, the NODECFG parameter can be used to directly assign a list of consumable generic attributes to individual nodes or to the special pseudo-node global, which provides shared cluster (floating) consumable resources. To set/overwrite a node's generic resources, use `GRES=<NAME>[:<COUNT>]`. (See Managing Consumable Generic Resources.) `NODECFG[node013] GRES=quickcalc:20`
MAXIOIN	Maximum input allowed on node before it is marked busy.
MAXJOB	See Node Policies for details.
MAXJOBPERUSER	See Node Policies for details.
MAXPE	See Node Policies for details.
MAXPEPERJOB	Maximum allowed Processor Equivalent per job on this node. A job will not be allowed to run on this node if its PE exceeds this number. `NODECFG[node024] MAXPEPERJOB=10000` `...`
MAXPROC	Maximum dedicated processors allowed on this node. No jobs are scheduled on this node when this number is reached. See Node Policies for more information. `NODECFG[node024] MAXPROC=8` `...`
NETWORK	The ability to specify which networks are available to a given node is limited to only a few resource managers. Using the **NETWORK** attribute, administrators can establish this node to network connection directly through the scheduler. The NODECFG parameter allows this list to be specified in a comma-delimited list. `NODECFG[node024] NETWORK=GigE` `...`
NODEINDEX	The node's index. See Node Location for details.

Attribute	Description
OS	This attribute specifies the node's operating system. `NODECFG[node013] OS=suse10` Because the TORQUE operating system overwrites the Moab operating system, change the operating system with **opsys** on page 2446 instead of **OS** if you are using TORQUE.
OSLIST	This attribute specifies the list of operating systems the node can run. `NODECFG[compute002] OSLIST=linux,windows`
OVERCOMMIT	Specifies the high-water limit for over-allocation of processors or memory on a hypervisor. This setting is used to protect hypervisors from having too many VMs placed on them, regardless of the utilization level of those VMs. Possible attributes include `DISK`, `MEM`, `PROC`, and `SWAP`. Usage is *<attr>:<integer>*. `NODECFG[node012] OVERCOMMIT=PROC:2,MEM:4`
PARTITION	See Node Location for details.
POWERPOLICY	The **POWERPOLICY** can be set to *OnDemand* or *STATIC*. It defaults to *STATIC* if not set. If set to *STATIC*, Moab will never automatically change the power status of a node. If set to *OnDemand*, Moab will turn the machine off and on based on workload and global settings. See Green Computing for further details.
PREEMPTMAXCPULOAD	If the node CPU load exceeds the specified value, any batch jobs running on the node are preempted using the preemption policy specified with the node's PREEMPTPOLICY attribute. If this attribute is not specified, the global default policy specified with PREEMPTPOLICY parameter is used. See Sharing Server Resources for further details. `NODECFG[node024] PRIORITY=-150 COMMENT="NFS Server Node"` `NODECFG[node024] PREEMPTPOLICY=CANCEL PREEMPTMAXCPULOAD=1.2` `...`

Attribute	Description
PREEMPTMINMEMAVAIL	If the available node memory drops below the specified value, any batch jobs running on the node are preempted using the preemption policy specified with the node's PREEMPTPOLICY attribute. If this attribute is not specified, the global default policy specified with PREEMPTPOLICY parameter is used. See Sharing Server Resources for further details. `NODECFG[node024] PRIORITY=-150 COMMENT="NFS Server Node"` `NODECFG[node024] PREEMPTPOLICY=CANCEL PREEMPTMINMEMAVAIL=256` `...`
PREEMPTPOLICY	If any node preemption policies are triggered (such as PREEMPTMAXCPULOAD or PREEMPTMINMEMAVAIL) any batch jobs running on the node are preempted using this preemption policy if specified. If not specified, the global default preemption policy specified with PREEMPTPOLICY parameter is used. See Sharing Server Resources for further details. `NODECFG[node024] PRIORITY=-150 COMMENT="NFS Server Node"` `NODECFG[node024] PREEMPTPOLICY=CANCEL PREEMPTMAXCPULOAD=1.2` `...`
PRIORITY	The **PRIORITY** attribute specifies the fixed node priority relative to other nodes. It is only used if NODEALLOCATIONPOLICY is set to *PRIORITY*. The default node priority is 0. A default cluster-wide node priority may be set by configuring the **PRIORITY** attribute of the *DEFAULT* node. See Priority Node Allocation for more details. `NODEALLOCATIONPOLICY  PRIORITY` `NODECFG[node024] PRIORITY=120` `...`

Attribute	Description
PRIORITYF	The **PRIORITYF** attribute specifies the function to use when calculating a node's allocation priority specific to a particular job. It is only used if NODEALLOCATIONPOLICY is set to *PRIORITY*. The default node priority function sets a node's priority exactly equal to the configured node priority. The priority function allows a site to indicate that various environmental considerations such as node load, reservation affinity, and ownership be taken into account as well using the following format: `<COEFFICIENT> * <ATTRIBUTE> [ + <COEFFICIENT> * <ATTRIBUTE> ]...` *<ATTRIBUTE>* is an attribute from the table found in the Priority Node Allocation section. A default cluster-wide node priority function may be set by configuring the **PRIORITYF** attribute of the *DEFAULT* node. See Priority Node Allocation for more details. `NODEALLOCATIONPOLICY  PRIORITY` `NODECFG[node024] PRIORITYF='APROC + .01 * AMEM - 10 * JOBCOUNT'` `...`
PROCSPEED	Knowing a node's processor speed can help the scheduler improve intra-job efficiencies by allocating nodes of similar speeds together. This helps reduce losses due to poor internal job load balancing. Moab's Node Set scheduling policies allow a site to control processor speed based allocation behavior. Processor speed information is specified in MHz and can be indicated directly using NODECFG or through use of the FEATUREPROCSPEEDHEADER parameter.
PROVRM	Provisioning resource managers can be specified on a per node basis. This allows flexibility in mixed environments. If the node does not have a provisioning resource manager, the default provisioning resource manager will be used. The default is always the first one listed in `moab.cfg`. `RMCFG[prov] TYPE=NATIVE RESOURCETYPE=PROV` `RMCFG[prov] PROVDURATION=10:00` `RMCFG[prov] NODEMODIFYURL=exec://$HOME/tools/os.switch.pl` `...` `NODECFG[node024] PROVRM=prov`
RACK	The rack associated with the node's physical location. Valid values range from 1 to 400. See Node Location for details.
RADISK	Jobs can request a certain amount of disk space through the RM Extension String's **DDISK** parameter. When done this way, Moab can track the amount of disk space available for other jobs. To set the total amount of disk space available the **RADISK** parameter is used.

Attribute	Description
RCDISK	Jobs can request a certain amount of disk space (in MB) through the RM Extension String's **DDISK** parameter. When done this way, Moab can track the amount of disk space available for other jobs. The **RCDISK** attribute constrains the amount of disk reported by a resource manager while the **RADISK** attribute specifies the amount of disk available to jobs. If the resource manager does not report available disk, the **RADISK** attribute should be used.
RCMEM	Jobs can request a certain amount of real memory (RAM) in MB through the RM Extension String's **DMEM** parameter. When done this way, Moab can track the amount of memory available for other jobs. The **RCMEM** attribute constrains the amount of RAM reported by a resource manager while the **RAMEM** attribute specifies the amount of RAM available to jobs. If the resource manager does not report available disk, the **RAMEM** attribute should be used. Please note that memory reported by the resource manager will override the configured value unless a trailing caret (^) is used. `NODECFG[node024] RCMEM=2048` `...` *If the resource manager does not report any memory, then Moab will assign node0242048 MB of memory.* `NODECFG[node024] RCMEM=2048^` `...` *Moab will assign 2048 MB of memory to node024 regardless of what the resource manager reports.*
RCPROC	The **RCPROC** specifies the number of processors available on a compute node. `NODECFG[node024] RCPROC=8` `...`

Attribute	Description
RCSWAP	Jobs can request a certain amount of swap space in MB. **RCSWAP** works similarly to **RCMEM**. Setting **RCSWAP** on a node will set the swap but can be overridden by swap reported by the resource manager. If the trailing caret (^) is used, Moab will ignore the swap reported by the resource manager and use the configured amount. `NODECFG[node024] RCSWAP=2048` `...` *If the resource manager does not report any memory, Moab will assign node0242048 MB of swap.* `NODECFG[node024] RCSWAP=2048^` `...` *Moab will assign 2048 MB of swap to node024 regardless of what the resource manager reports.*
SIZE	The number of slots or size units consumed by the node. This value is used in graphically representing the cluster using showstate or Moab Cluster Manager. See Node Location for details. For display purposes, legal size values include *1*, *2*, *3*, *4*, *6*, *8*, *12*, and *16*. `NODECFG[node024] SIZE=2` `...`
SLOT	The first slot in the rack associated with the node's physical location. Valid values range from 1 to **MMAX_RACKSIZE** (default=*64*). See Node Location for details.
SPEED	Because today's processors have multiple cores and adjustable clock frequency, this feature has no meaning and will be deprecated. The **SPEED** specification must be in the range of 0.01 to 100.0.
TRIGGER	See **About object triggers** on page 724 for information.
VARIABLE	Variables associated with the given node, which can be used in job scheduling. See -l PREF. `NODECFG[node024] VARIABLE=var1` `...`

Attribute	Description
VMOCTHRESHOLD	Specifies the high-water threshold for utilization of resources on a server (i.e. processor and memory). This setting is used to protect hypervisors from becoming too highly utilized and thus negatively impacting the performance of VMs running on the hypervisor. Possible attributes include `PROC` and `MEM`. `NODECFG[node024] VMOCTHRESHOLD=PROC=2,MEM=2`

Node Features/Node Properties

A node feature (or node property) is an opaque string label that is associated with a compute node. Each compute node may have any number of node features assigned to it, and jobs may request allocation of nodes that have specific features assigned. Node features are labels and their association with a compute node is not conditional, meaning they cannot be consumed or exhausted.

Node features may be assigned by the resource manager, and this information may be imported by Moab or node features may be specified within Moab directly. Moab supports hyphens and underscores in node feature names.

As a convenience feature, certain node attributes can be specified via node features using the parameters listed in the following table:

PARAMETER	DESCRIPTION
FEATURENODETYPEHEADER	Set Node Type
FEATUREPARTITIONHEADER	Set Partition
FEATUREPROCSPEEDHEADER	Set Processor Speed
FEATURERACKHEADER	Set Rack
FEATURESLOTHEADER	Set Slot

Example 3-134:

```
FEATUREPARTITIONHEADER  par
FEATUREPROCSPEEDHEADER  cpu
```

Related topics

- Job Preferences
- Configuring **Specifying Node Features (Node Properties)** on page 2224 in TORQUE
- Configuring Node Features in Moab with NODECFG
- Specifying Job Feature Requirements

- Viewing Feature Availability Breakdown with mdiag -t
- Differences between Node Features and Managing Consumable Generic Resources

Node Specific Policies

Node policies within Moab allow specification of not only how the node's load should be managed, but who can use the node, and how the node and jobs should respond to various events. These policies allow a site administrator to specify on a node by node basis what the node will and will not support. Node policies may be applied to specific nodes or applied system-wide using the specification `NODECFG [DEFAULT] ...`.

Node Usage/Throttling Policies

MAXJOB

This policy constrains the number of total independent jobs a given node may run simultaneously. It can only be specified via the NODECFG parameter.

> On Cray XT systems, use the NID (node id) instead of the node name. For more information, see Configuring the moab.cfg file.

MAXJOBPERUSER

Constrains the number of total independent jobs a given node may run simultaneously associated with any single user. It can only be specified via the NODECFG parameter.

MAXJOBPERGROUP

Constrains the number of total independent jobs a given node may run simultaneously associated with any single group. It can only be specified via the NODECFG parameter.

MAXLOAD

MAXLOAD constrains the CPU load the node will support as opposed to the number of jobs. This maximum load policy can also be applied system wide using the parameter NODEMAXLOAD.

MAXPE

This policy constrains the number of total dedicated processor-equivalents a given node may support simultaneously. It can only be specified via the **NODECFG** parameter.

MAXPROC

This policy constrains the number of total dedicated processors a given node may support simultaneously. It can only be specified via the **NODECFG** parameter.

MAXPROCPERUSER

This policy constrains the number of total processors a given node may have dedicated to any single user. It can only be specified via the **NODECFG** parameter.

MAXPROCPERGROUP

This policy constrains the number of total processors a given node may have dedicated to any single group. It can only be specified via the **NODECFG** parameter.

> Node throttling policies are used strictly as constraints. If a node is defined as having a single processor or the NODEACCESSPOLICY is set to *SINGLETASK*, and a **MAXPROC** policy of 4 is specified, Moab will not run more than one task per node. A node's configured processors must be specified so that multiple jobs may run and then the **MAXJOB** policy will be effective. The number of configured processors per node is specified on a resource manager specific basis. PBS, for example, allows this to be adjusted by setting the number of virtual processors with the `np` parameter for each node in the PBS `nodes` file.

Example 3-135:

```
NODECFG[node024] MAXJOB=4 MAXJOBPERUSER=2
NODECFG[node025] MAXJOB=2
NODECFG[node026] MAXJOBPERUSER=1
NODECFG[DEFAULT] MAXLOAD=2.5
...
```

Node Access Policies

While most sites require only a single cluster wide node access policy (commonly set using NODEACCESSPOLICY), it is possible to specify this policy on a node by node basis using the **ACCESS** attributes of the NODECFG parameter. This attribute may be set to any of the valid node access policy values listed in the Node Access Policies section.

Example 3-136:

To set a global policy of *SINGLETASK* on all nodes except nodes 13 and 14, use the following:

```
# by default, enforce dedicated node access on all nodes
NODEACCESSPOLICY  SINGLETASK
# allow nodes 13 and 14 to be shared
NODECFG[node13]   ACCESS=SHARED
NODECFG[node14]   ACCESS=SHARED
```

Related topics

- mnodectl

Managing Shared Cluster Resources (Floating Resources)

This section describes how to configure, request, and reserve cluster file system space and bandwidth, software licenses, and generic cluster resources.

Shared Cluster Resource Overview

Shared cluster resources such as file systems, networks, and licenses can be managed through creating a pseudo-node. You can configure a pseudo-node via the NODECFG parameter much as a normal node would be but additional information is required to allow the scheduler to contact and synchronize state with the resource.

In the following example, a license manager is added as a cluster resource by defining the *GLOBAL* pseudo-node and specifying how the scheduler should query and modify its state.

```
NODECFG[GLOBAL] RMLIST=NATIVE
NODECFG[GLOBAL] QUERYCMD=/usr/local/bin/flquery.sh
NODECFG[GLOBAL] MODIFYCMD=/usr/local/bin/flmodify.sh
```

In some cases, pseudo-node resources may be very comparable to node-locked generic resources however there are a few fundamental differences which determine when one method of describing resources should be used over the other. The following table contrasts the two resource types.

Attribute	Pseudo-Node	Generic Resource
Node-Locked	No - Resources can be encapsulated as an independent node.	Yes - Must be associated with an existing compute node.
Requires exclusive batch system control over resource	No - Resources (such as file systems and licenses) may be consumed both inside and outside of batch system workload.	Yes - Resources must only be consumed by batch workload. Use outside of batch control results in loss of resource synchronization.
Allows scheduler level allocation of resources	Yes - If required, the scheduler can take external administrative action to allocate the resource to the job.	No - The scheduler can only maintain logical allocation information and cannot take any external action to allocate resources to the job.

Configuring Generic Consumable Floating Resources

Consumable floating resources are configured in the same way as node-locked generic resources with the exception of using the *GLOBAL* node instead of a particular node.

```
NODECFG[GLOBAL] GRES=tape:4,matlab:2
...
```

In this setup, four resources of type tape *and 2 of type* matlab *are floating and available across all nodes.*

Requesting Consumable Floating Resources

Floating resources are requested on a per task basis using native resource manager job submission methods or using the GRES resource manager extensions.

Configuring Cluster File Systems

Moab allows both the file space and bandwidth attributes or a cluster file system to be tracked, reserved, and scheduled. With this capability, a job or reservation may request a particular quantity of file space and a required amount of I/O bandwidth to this file system. While file system resources are managed as a cluster generic resource, they are specified using the **FS** attribute of the **NODECFG** parameter as in the following example:

```
NODECFG[GLOBAL] FS=PV1:10000@100,PV2:5000@100
...
```

In this example, PV1 defines a 10 GB file system with a maximum throughput of 100 MB/s while PV2 defines a 5 GB file system also possessing a maximum throughput of 100 MB/s.

A job may request cluster file system resources using the `fs` resource manager extension. For a TORQUE based system, the following could be used:

```
>qsub -l nodes=1,walltime=1:00:00 -W x=fs:10@50
```

Configuring Cluster Licenses

Jobs may request and reserve software licenses using native methods or using the GRES resource manager extension. If the cluster license manager does not support a query interface, license availability may be specified within Moab using the **GRES** attribute of the NODECFG parameter.

Example 3-137: Configure Moab to support four floating quickcalc and two floating matlab licenses.

```
NODECFG[GLOBAL] GRES=quickcalc:4,matlab:2
...
```

Example 3-138: Submit a TORQUE job requesting a node-locked or floating quickcalc license.

```
> qsub -l nodes=1,software=quickcalc,walltime=72000 testjob.cmd
```

Configuring Generic Resources as Features

Moab can be configured to treat generic resources as features in order to provide more control over server access. For instance, if a node is configured with a certain **GRES** and that **GRES** is turned off, jobs requesting the node will not run. To turn a GRES into a feature, set the **FEATUREGRES** attribute of GRESCFG to *TRUE* in the `moab.cfg` file.

```
GRESCFG[gres1] FEATUREGRES=TRUE
```

Moab now treats gres1 as a scheduler-wide feature rather than a normal generic resource.

Note that jobs are submitted normally using the same GRES syntax.

> If you are running a grid, verify that `FEATUREGRES=TRUE` is set on all members of the grid.

> You can safely upgrade an existing cluster to use the feature while jobs are running. If you are in a grid, upgrade all clusters at the same time.

Two methods exist for managing GRES features: via Moab commands and via the resource manager. Using Moab commands means that feature changes are not checkpointed; they do not remain in place when Moab restarts. Using the resource manager causes changes to be reported by the RM, so any changes made before a Moab restart are still present after it.

These methods are mutually exclusive. Use one or the other, but do not mix methods.

Managing Feature GRES via Moab Commands

In the following example, *gres1* and *gres2* are configured in the `moab.cfg` file. *gres1* is not currently functioning correctly, so it is set to 0, turning the feature off. Values above 0 and non-specified values turn the feature on.

```
NODECFG[GLOBAL] GRES=gres1:0
NODECFG[GLOBAL] GRES=gres2:10000
GRESCFG[gres1] FEATUREGRES=TRUE
GRESCFG[gres2] FEATUREGRES=TRUE
```

Moab now treats gres1 and gres2 as features.

To verify that this is set up correctly, run mdiag -S -v. It returns the following:

```
> mdiag -S -v
...
  Scheduler FeatureGres: gres1:off,gres2:on
```

Once Moab has started, use mschedctl -m to modify whether the feature is turned on or off.

```
mschedctl -m sched featuregres:gres1=on

INFO: FeatureGRes 'gres1' turned on
```

You can verify that the feature turned on or off by once again running mdiag -S -v.

If Moab restarts, it will not checkpoint the state of these changed feature general resources. Instead, it will read the `moab.cfg` file to determine whether the feature GRES is on or off.

With feature GRES configured, jobs are submitted normally, requesting GRES type *gres1* and *gres2*. Moab ignores GRES counts and reads the feature simply as on or off.

```
> msub -l nodes=1,walltime=600,gres=gres1

1012
> checkjob 1012
job 1012

AName: STDIN
State: Running
.....
StartTime: Tue Jul 3 15:33:28
Feature GRes: gres1
Total Requested Tasks: 1
```

If you request a feature that is currently turned off, the state is not reported as `Running`, but as `Idle`. A message like the following returns:

```
BLOCK MSG: requested feature gres 'gres2' is off
```

Managing Feature GRES via the Resource Manager

You can automate the process of having a feature GRES turn on and off by setting up an external tool and configuring Moab to query the tool the same way that Moab queries a license manager. For example:

```
RMCFG[myRM] CLUSTERQUERYURL=file:///$HOME/tools/myRM.dat TYPE=NATIVE
RESOURCETYPE=LICENSE

GRESCFG[gres1] FEATUREGRES=TRUE
GRESCFG[gres2] FEATUREGRES=TRUE
```

> *LICENSE means that the RM does not contain any compute resources and that Moab should not attempt to use it to manage any jobs (start, cancel, submit, etc.).*

The `myRM.dat` file should contain something like the following:

```
GLOBAL state=Idle cres=gres1:0,gres2:10
```

External tools can easily update the file based on filesystem availability. Switching any of the feature GRES to 0 turns it off and switching it to a positive value turns it on. If you use this external mechanism, you do not need to use mschedctl -m to turn a feature GRES on or off. You also do not need to worry about whether Moab has checkpointed the information or not, since the information is provided by the RM and not by any external commands.

Related topics

- Managing Resources Directly with the Native Interface

Managing Node State

There are multiple models in which Moab can operate allowing it to either honor the node state set by an external service or locally determine and set the node state. This section covers the following:

- identifying meanings of particular node states
- specifying node states within locally developed services and resource managers
- adjusting node state within Moab based on load, policies, and events

Node State Definitions

State	Definition
Down	Node is either not reporting status, is reporting status but failures are detected, or is reporting status but has been marked down by an administrator.
Idle	Node is reporting status, currently is not executing any workload, and is ready to accept additional workload.

State	Definition
Busy	Node is reporting status, currently is executing workload, and cannot accept additional workload due to load.
Running	Node is reporting status, currently is executing workload, and can accept additional workload.
Drained	Node is reporting status, currently is not executing workload, and cannot accept additional workload due to administrative action.
Draining	Node is reporting status, currently is executing workload, and cannot accept additional workload due to administrative action.

Specifying Node States within Native Resource Managers

Native resource managers can report node state implicitly and explicitly, using **NODESTATE**, **LOAD**, and other attributes. See Managing Resources Directly with the Native Interface for more information.

Moab Based Node State Adjustment

Node state can be adjusted based on reported processor, memory, or other load factors. It can also be adjusted based on reports of one or more resource managers in a multi-resource manager configuration. Also, both generic events and generic metrics can be used to adjust node state.

- TORQUE health scripts (allow compute nodes to detect and report site specific failures).

Adjusting Scheduling Behavior Based on Reported Node State

Based on reported node state, Moab can support various policies to make better use of available resources. For more information, see the **Green computing overview** on page 701.

Down State

- JOBACTIONONNODEFAILURE parameter (cancel/requeue jobs if allocated nodes fail).
- Triggers (take specified action if failure is detected).

Related topics

- Managing Resources Directly with the Native Interface
- License Management
- Adjusting Node Availability
- NODEMAXLOAD parameter
- Green computing overview

Managing Consumable Generic Resources

- Configuring Node-Locked Consumable Generic Resources
  - Requesting Consumable Generic Resources
- Managing Generic Resource Race Conditions

Each time a job is allocated to a compute node, it consumes one or more types of resources. Standard resources such as CPU, memory, disk, network adapter bandwidth, and swap are automatically tracked and consumed by Moab. However, in many cases, additional resources may be provided by nodes and consumed by jobs that must be tracked. The purpose of this tracking may include accounting, billing, or the prevention of resource over-subscription. Generic consumable resources may be used to manage software licenses, I/O usage, bandwidth, application connections, or any other aspect of the larger compute environment; they may be associated with compute nodes, networks, storage systems, or other real or virtual resources.

These additional resources can be managed within Moab by defining one or more generic resources. The first step in defining a generic resource involves naming the resource. Generic resource availability can then be associated with various compute nodes and generic resource usage requirements can be associated with jobs.

Differences Between Node Features and Consumable Resources

A node feature (or node property) is an opaque string label that is associated with a compute node. Each compute node may have any number of node features assigned to it and jobs may request allocation of nodes that have specific features assigned. Node features are labels and their association with a compute node is not conditional, meaning they cannot be consumed or exhausted.

Configuring Node-locked Consumable Generic Resources

Consumable generic resources are supported within Moab using either direct configuration or resource manager auto-detect (as when using TORQUE and accelerator hardware). For direct configuration, node-locked consumable generic resources (or generic resources) are specified using the **NODECFG** parameter's **GRES** attribute. This attribute is specified using the format *<ATTR>*:*<COUNT>* as in the following example:

```
NODECFG[titan001] GRES=tape:4
NODECFG[login32]  GRES=matlab:2,prime:4
NODECFG[login33]  GRES=matlab:2
...
```

By default, Moab supports up to 128 independent generic resource types.

Requesting Consumable Generic Resources

Generic resources can be requested on a per task or per job basis using the **GRES** resource manager extension. If the generic resource is located on a compute node, requests are by default interpreted as a per task request. If the generic resource is located on a shared, cluster-level resource (such as a network or storage system), then the request defaults to a per job interpretation.

Generic resources are specified per task, not per node. When you submit a job, each processor becomes a task. For example, a job asking for `nodes=3:ppn=4,gres=test:5` asks for 60 gres of type test ((3*4 processors)*5).

If using TORQUE, the GRES or software resource can be requested as in the following examples:

Example 3-139: Per Task Requests

```
NODECFG[compute001] GRES=dvd:2 SPEED=2200
NODECFG[compute002] GRES=dvd:2 SPEED=2200
NODECFG[compute003] GRES=dvd:2 SPEED=2200
NODECFG[compute004] GRES=dvd:2 SPEED=2200
NODECFG[compute005] SPEED=2200
NODECFG[compute006] SPEED=2200
NODECFG[compute007] SPEED=2200
NODECFG[compute008] SPEED=2200
```

```
# submit job which will allocate only from nodes 1 through 4 requesting one dvd per
task
> qsub -l nodes=2,walltime=100,gres=dvd job.cmd
```

In this example, Moab determines that compute nodes exist that possess the requested generic resource. A compute node is a node object that possesses processors on which compute jobs actually execute. License server, network, and storage resources are typically represented by non-compute nodes. Because compute nodes exist with the requested generic resource, Moab interprets this job as requesting two compute nodes each of which must also possess a DVD generic resource.

Example 3-140: Per Job Requests

```
NODECFG[network] PARTITION=shared GRES=bandwidth:2000000
```

```
# submit job which will allocate 2 nodes and 10000 units of network bandwidth
> qsub -l nodes=2,walltime=100,gres=bandwidth:10000 job.cmd
```

In this example, Moab determines that there exist no compute nodes that also possess the generic resource bandwidth so this job is translated into a multiple-requirement—multi-req—job. Moab creates a job that has a requirement for two compute nodes and a second requirement for 10000 bandwidth generic resources. Because this is a multi-req job, Moab knows that it can locate these needed resources separately.

Using Generic Resource Requests in Conjunction with other Constraints

Jobs can explicitly specify generic resource constraints. However, if a job also specifies a hostlist, the hostlist constraint overrides the generic resource constraint if the request is for per task allocation. In the Per Task Requests example, if the job also specified a hostlist, the *DVD* request is ignored.

Requesting Resources with No Generic Resources

In some cases, it is valuable to allocate nodes that currently have no generic resources available. This can be done using the special value *none* as in the following example:

```
> qsub -l nodes=2,walltime=100,gres=none job.cmd
```

In this case, the job only allocates compute nodes that have no generic resources associated with them.

Requesting Generic Resources Automatically within a Queue/Class

Generic resource constraints can be assigned to a queue or class and inherited by any jobs that do not have a **gres** request. This allows targeting of specific resources, automation of co-allocation requests, and other uses. To enable this, use the DEFAULT.GRES attribute of the CLASSCFG parameter as in the following example:

```
CLASSCFG[viz] DEFAULT.GRES=graphics:2
```

For each node requested by a *viz* job, also request two graphics cards.

Managing Generic Resource Race Conditions

A software license race condition "window of opportunity" opens when Moab checks a license server for sufficient available licenses and closes when the user's software actually checks out the software licenses. The time between these two events can be seconds to many minutes depending on overhead factors such as node OS provisioning, job startup, licensed software startup, and so forth.

During this window, another Moab-scheduled job or a user or job external to the cluster or cloud can obtain enough software licenses that by the time the job attempts to obtain its software licenses, there are an insufficient quantity of available licenses. In such cases a job will sit and wait for the license, and while it waits it occupies but does not use resources that another job could have used. Use the **STARTDELAY** parameter to prevent such a situation.

```
GRESCFG[<license>] STARTDELAY=<window_of_opportunity>
```

With the **STARTDELAY** parameter enabled (on a per generic resource basis) Moab blocks any idle jobs requesting the same generic resource from starting until the *<window_of_opportunity>* passes. The window is defined by the customer on a per generic resource basis.

Related topics

- GRESCFG parameter
- Generic Metrics
- Generic Events
- General Node Attributes
- Floating Generic Resources
- Per Class Assignment of Generic Resource Consumption
- mnodectl -m command to dynamically modify node resources
- Favoring Jobs Based On Generic Resource Requirements

Enabling Generic Metrics

- Configuring Generic Metrics
- Example Generic Metric Usage

Moab allows organizations to enable generic performance metrics. These metrics allow decisions to be made and reports to be generated based on site specific environmental factors. This increases Moab's awareness of what is occurring within a given cluster environment, and allows arbitrary information to be associated with resources and the workload within the cluster. Uses of these metrics are widespread and can cover anything from tracking node temperature, to memory faults, to application effectiveness.

- Execute triggers when specified thresholds are reached
- Modify node allocation affinity for specific jobs
- Initiate automated notifications when thresholds are reached
- Display current, average, maximum, and minimum metrics values in reports and charts within Moab Cluster Manager

Configuring Generic Metrics

A new generic metric is automatically created and tracked at the server level if it is reported by either a node or a job.

To associate a generic metric with a job or node, a native resource manager must be set up and the GMETRIC attribute must be specified. For example, to associate a generic metric of *temp* with each node in a TORQUE cluster, the following could be reported by a native resource manager:

```
# temperature output
node001 GMETRIC[temp]=113
node002 GMETRIC[temp]=107
node003 GMETRIC[temp]=83
node004 GMETRIC[temp]=85
...
```

> Generic metrics are tracked as floating point values allowing virtually any number to be reported.

In the preceding example, the new metric, *temp*, can now be used to monitor system usage and performance or to allow the scheduler to take action should certain thresholds be reached. Some uses include the following:

- Executing triggers based on generic metric thresholds
- Adjust a node's availability for accepting additional workload
- Adjust a node's allocation priority
- Initiate administrator notification of current, minimum, maximum, or average generic metric values
- Use metrics to report resource and job performance
- Use metrics to report resource and job failures
- Using job profiles to allow Moab to learn which resources best run which applications
- Tracking effective application efficiency to identify resource brown outseven when no node failure is obvious
- Viewing current and historical cluster-wide generic metric values to identify failure, performance, and usage
- Enable charging policies based on consumption of generic metrics patterns

- View changes in generic metrics on nodes, jobs, and cluster wide over time
- Submit jobs with generic metric based node-allocation requirements

Generic metric values can be viewed using checkjob, checknode, mdiag -n,mdiag -j, or Moab Cluster Manager Charting and Reporting Features.

> Historical job and node generic metric statistics can be cleared using the mjobctl and mnodectl commands.

Example Generic Metric Usage

As an example, consider a cluster with two primary purposes for generic metrics. The first purpose is to track and adjust scheduling behavior based on node temperature to mitigate overheating nodes. The second purpose is to track and charge for utilization of a locally developed data staging service.

The first step in enabling a generic metric is to create probes to monitor and report this information. Depending on the environment, this information may be distributed or centralized. In the case of temperature monitoring, this information is often centralized by a hardware monitoring service and available via command line or an API. If monitoring a locally developed data staging service, this information may need to be collected from multiple remote nodes and aggregated to a central location. The following are popular freely available monitoring tools:

Tool	Link
BigBrother	http://www.bb4.org
Ganglia	http://ganglia.sourceforge.net
Monit	http://www.tildeslash.com/monit
Nagios	http://www.nagios.org

Once the needed probes are in place, a native resource manager interface must be created to report this information to Moab. Creating a native resource manager interface should be very simple, and in most cases a script similar to those found in the `$TOOLSDIR`(`$PREFIX/tools`) directory can be used as a template. For this example, we will assume centralized information and will use the RM script that follows:

```
#!/usr/bin/perl
# 'hwctl outputs information in format '<NODEID> <TEMP>'
open(TQUERY,"/usr/sbin/hwctl -q temp |");
while (<TQUERY>)
  {
  my $nodeid,$temp = split /\w+/;
  $dstage=GetDSUsage($nodeid);
  print "$nodeid GMETRIC[temp]=$temp GMETRIC[dstage]=$dstage
";
  }
```

With the script complete, the next step is to integrate this information into Moab. This is accomplished with the following configuration line:

```
RMCFG[local] TYPE=NATIVE CLUSTERQUERYURL=file://$TOOLSDIR/node.query.local.pl
...
```

Moab can now be recycled and temperature and data staging usage information will be integrated into Moab compute node reports.

If the checknode command is run, output similar to the following is reported:

```
> checknode cluster013
...
Generic Metrics:  temp=113.2,dstage=23748
...
```

Moab Cluster Manager reports full current and historical generic metric information in its visual cluster overview screen.

The next step in configuring Moab is to inform Moab to take certain actions based on the new information it is tracking. For this example, there are two purposes. The first purpose is to get jobs to avoid hot nodes when possible. This is accomplished using the **GMETRIC** attribute of the Node Allocation Priority function as in the following example:

```
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT] PRIORITYF=PRIORITY-10*GMETRIC[temp]
...
```

This simple priority function reduces the priority of the hottest nodes making such less likely to be allocated. See Node Allocation Priority Factors for a complete list of available priority factors.

The example cluster is also interested in notifying administrators if the temperature of a given node ever exceeds a critical threshold. This is accomplished using a trigger. The following line will send email to administrators any time the temperature of a node exceeds 120 degrees.

```
NODECFG[DEFAULT] TRIGGER=atype=mail,etype=threshold,threshold=gmetric[temp]
>120,action='warning: node $OID temp high'
...
```

Related topics

- Simulation Overview
- Generic Consumable Resources
- Object Variables
- Generic Event Counters

Enabling Generic Events

- Configuring Generic Events
  - Action Types
  - Named Events

Generic events are used to identify failures and other occurrences that Moab or other systems must be made aware. This information may result in automated resource recovery, notifications, adjustments to statistics, or changes in policy. Generic events also have the ability to carry an arbitrary human readable message that may be attached to associated objects or passed to administrators or external systems. Generic events typically signify the occurrence of a specific event as opposed to generic metrics which indicate a change in a measured value.

Using generic events, Moab can be configured to automatically address many failures and environmental changes improving the overall performance. Some sample events that sites may be interested in monitoring, recording, and taking action on include:

- Machine Room Status
  - Excessive Room Temperature
  - Power Failure or Power Fluctuation
  - Chiller Health
- Network File Server Status
  - Failed Network Connectivity
  - Server Hardware Failure
  - Full Network File System
- Compute Node Status
  - Machine Check Event (MCE)
  - Network Card (NIC) Failure
  - Excessive Motherboard/CPU Temperature
  - Hard Drive Failures

Configuring Generic Events

Generic events are defined in the `moab.cfg` file and have several different configuration options. The only required option is **action**.

The full list of configurable options for generic events is contained in the following table:

Attribute	Description
ACTION	Comma-delimited list of actions to be processed when a new event is received.
ECOUNT	Number of events that must occur before launching action. Action will be launched each *<ECOUNT>* event if rearm is set.
REARM	Minimum time between events specified in `[[[DD:]HH:]MM:]SS` format.
SEVERITY	An arbitrary severity level from 1 through 4, inclusive. SEVERITY appears in the output of `mdiag -n -v -v --xml`. The severity level will not be used for any other purpose.

Action Types

The impact of the event is controlled using the **ACTION** attribute of the **GEVENTCFG** parameter. The **ACTION** attribute is comma-delimited and may include any combination of the actions in the following table:

Value	Description
DISABLE [:<OTYPE>:<OID>]	Marks event object (or specified object) down until event report is cleared.
EXECUTE	Executes a script at the provided path. The value of *EXECUTE* is not contained in quotation marks. Arguments are allowed at the end of the path and are separated by question marks (`?`). Trigger variables (such as *$OID*) are allowed.
NOTIFY	Notifies administrators of the event occurrence.
OBJECTXMLSTDIN	If the *EXECUTE* action type is also specified, this flag passes an XML description of the firing gevent to the script.
OFF	Powers off node or resource.
ON	Powers on node or resource.
PREEMPT [:<POLICY>]	Preempts workload associated with object (valid for node, job, reservation, partition, resource manager, user, group, account, class, QoS, and cluster objects).

Value	Description
RECORD	Records events to the event log. The record action causes a line to be added to the event log regardless of whether or not RECORDEVENTLIST includes GEVENT.
RESERVE [:<DURATION>]	Reserves node for specified duration (default: 24 hours).
RESET	Resets object (valid for nodes - causes reboot).
SIGNAL[:<SIGNO>]	Sends signal to associated jobs or services (valid for node, job, reservation, partition, resource manager, user, group, account, class, QoS, and cluster objects).

This is an example of using `objectxmlstdin` with a gevent:

```
<gevent name="bob" statuscode="0" time="1320334763">Testing</gevent>
```

Named Events

In general, generic events are named, with the exception of those based on generic metrics. Names are used primarily to differentiate between different events and do not have any intrinsic meaning to Moab. It is suggested that the administrator choose names that denote specific meanings within the organization.

Example 3-141:

```
# Note: cpu failures require admin attention, create maintenance reservation
GEVENTCFG[cpufail] action=notify,record,disable,reserve rearm=01:00:00# Note: power
failures are transient, minimize future use
GEVENTCFG[powerfail] action=notify,record, rearm=00:05:00
# Note: fs full can be automatically fixed
GEVENTCFG[fsfull] action=notify,execute:/home/jason/MyPython/cleartmp.py?$OID?nodefix
# Note: memory errors can cause invalid job results, clear node immediately
GEVENTCFG[badmem] action=notify,record,preempt,disable,reserve
```

Generic Metric (GMetric) Events

GMetric events are generic events based on generic metrics. They are used for executing an action when a generic metric passes a defined threshold. Unlike named events, GMetric events are not named and use the following format:
`GEVENTCFG[GMETRIC<COMPARISON>VALUE] ACTION=...`

Example 3-142:

```
GEVENTCFG[cputemp>150] action=off
```

This form of generic events uses the GMetric name, as returned by a **GMETRIC** attribute in a native Resource Manager interface.

Only one generic event may be specified for any given generic metric.

Valid comparative operators are shows in the following table:

Type	Comparison	Notes
>	greater than	Numeric values only
> =	greater than or equal to	Numeric values only
= =	equal to	Numeric values only
<	less than	Numeric values only
< =	less than or equal to	Numeric values only
< >	not equal	Numeric values only

Reporting Generic Events

Unlike generic metrics, generic events can be optionally configured at the global level to adjust rearm policies, and other behaviors. In all cases, this is accomplished using the GEVENTCFG parameter.

To report an event associated with a job or node, use the native Resource Manager interface or the mjobctl or mnodectl commands. You can report generic events on the scheduler with the mschedctl command.

If using the native Resource Manager interface, use the GEVENT attribute as in the following example:

```
node001 GEVENT[hitemp]='temperature exceeds 150 degrees'
node017 GEVENT[fullfs]='/var/tmp is full'
```

The time at which the event occurred can be passed to Moab to prevent multiple processing of the same event. This is accomplished by specifying the event type in the format *<GEVENTID>*[:*<EVENTTIME>*] as in what follows:

```
node001 GEVENT[hitemp:1130325993]='temperature exceeds 150 degrees'
node017 GEVENT[fullfs:1130325142]='/var/tmp is full'
```

Using Generic Events for VM Detection

To enable Moab to detect a virtual machine (VM) reported by a generic event, do the following:

1. Set up your resource manager to detect virtual machine creation and to submit a generic event to Moab.
2. Configure `moab.cfg` to recognize a generic event.

```
GEVENTCFG[NewVM] ACTION=execute:/opt/moab/AddVM.py,OBJECTXMLSTDIN
```

3. Report the event.

```
> mschedctl -c gevent -n NewVM -m "VM=newVMName"
```

> *With the ObjectXMLStdin action set, Moab sends an XML description of the generic event to the script, so the message passes through.*

The following sample Perl script submits a VMTracking job for the new VM:

```
#!/usr/bin/perl

# in moab.cfg: GEVENTCFG[NewVM] ACTION=execute:$TOOLSDIR/newvm_event.pl,OBJECTXMLSTDIN
# trigger gevent with: mschedctl -c gevent -n NewVM -m "VM=TestVM1"
# input to this script: <gevent name="NewVM" statuscode="0"
time="1318500261">VM=TestVM1</gevent>

use strict;

my $vmidVarName = "preVMID";
my $vmTemplate = "existingVM";
my $vmOwner = "operator";

$ENV{MOABHOMEDIR} = '/opt/moab';

my $xml = join "", <STDIN>;
my ($vmid) = ($xml =~ m/VM=([^\<]+)\</);
if ( defined $vmid )
{
       my $cmd = qq| $ENV{MOABHOMEDIR}/bin/mvmctl -q $vmid --xml |;
       my $vmxml = `$cmd`;
       my ($hv, $os, $proc, $disk, $mem) = (undef, undef, undef, undef, undef);
       ($hv) = ($vmxml =~ m/CONTAINERNODE="([^"]+)"/);
       ($os) = ($vmxml =~ m/OS="([^"]+)"/);
       ($proc) = ($vmxml =~ m/RCPROC="([^"]+)"/);
       ($mem) = ($vmxml =~ m/RCMEM="([^"]+)"/);
       ($disk) = ($vmxml =~ m/RCDISK="([^"]+)"/);
       die "Error parsing VM XML. Invalid VMID $vmid or $hv || $os || $proc || $mem ||
$disk?
"
               if ( ! defined $hv || !defined $os || !defined $proc || !defined $mem || !defined
$disk );

       $cmd = qq| $ENV{MOABHOMEDIR}/bin/msub -l
hostlist=$hv,os=$os,nodes=1:ppn=$proc,mem=$mem,file=$disk,template=$vmTemplate,VAR=$vm
idVarName=$vmid --proxy=$vmOwner /dev/null |;
       my $msubout = `$cmd`;
       die "Error executing msub. Output is:
$msubout
" if ( $? );
} else {
       die "Error parsing VMID from GEVENT message
";
}
```

Generic Events Attributes

Each node will record the following about reported generic events:

- status - is event active
- message - human readable message associated with event
- count - number of event incidences reported since statistics were cleared
- time - time of most recent event

Each event can be individually cleared, annotated, or deleted by cluster administrators using a mnodectl command.

> Generic events are only available in Moab 4.5.0 and later.

Manually Creating Generic Events

Generic events may be manually created on a physical node or VM.

To add GEVENT `event` with message `"hello"` to `node02`, do the following:

```
> mnodectl -m gevent=event:"hello" node02
```

To add GEVENT `event` with message `"hello"` to `myvm`, do the following:

```
> mvmctl -m gevent=event:"hello" myvm
```

Related topics

- Simulation Overview
- Generic Consumable Resources
- Object Variables
- Generic Event Counters

Resource Managers and Interfaces

- **Resource Provisioning** on page 665
- **Resource Manager Translation** on page 672

Moab provides a powerful resource management interface that enables significant flexibility in how resources and workloads are managed. Highlights of this interface are listed in what follows:

Highlight	Description
Support for Multiple Standard Resource Manager Interface Protocols	Manage cluster resources and workloads via PBS, Loadleveler, SGE, LSF, or BProc based resource managers.
Support for Generic Resource Manager Interfaces	Manage cluster resources securely via locally developed or open source projects using simple flat text interfaces or XML over HTTP.
Support for Multiple Simultaneous Resource Managers	Integrate resource and workload streams from multiple independent sources reporting disjoint sets of resources.
Independent Workload and Resource Management	Allow one system to manage your workload (queue manager) and another to manage your resources.
Support for Rapid Development Interfaces	Load resource and workload information directly from a file, a URL, or from the output of a configurable script or other executable.
Resource Extension Information	Integrate information from multiple sources to obtain a cohesive view of a compute resource. (That is, mix information from NIM, OpenPBS, FLEXlm, and a cluster performance monitor to obtain a single node image with a coordinated state and a more extensive list of node configuration and utilization attributes.)

Resource Manager Overview

For most installations, the Moab Workload Manager uses the services of a resource manager to obtain information about the state of compute resources (nodes) and workload (jobs). Moab also uses the resource manager to manage jobs, passing instructions regarding when, where, and how to start or otherwise manipulate jobs.

Moab can be configured to manage more than one resource manager simultaneously, even resource managers of different types. Using a local queue, jobs may even be migrated from one resource manager to another. However, there are currently limitations regarding jobs submitted directly to a resource manager (not to the local queue.) In such cases, the job is constrained to only run within the bound of the resource manager to which it was submitted.

Scheduler/Resource Manager Interactions

Moab interacts with all resource managers using a common set of commands and objects. Each resource manager interfaces, obtains, and translates Moab concepts regarding workload and resources into native resource manager objects, attributes, and commands.

Information on creating a new scheduler resource manager interface can be found in the Adding New Resource Manager Interfaces section.

Resource Manager Commands

For many environments, Moab interaction with the resource manager is limited to the following objects and functions:

Object	Function	Details
Job	Query	Collect detailed state, requirement, and utilization information about jobs
	Modify	Change job state and/or attributes
	Start	Execute a job on a specified set of resources
	Cancel	Cancel an existing job
	Preempt/Resume	Suspend, resume, checkpoint, restart, or requeue a job

Object	Function	Details
Node	Query	Collect detailed state, configuration, and utilization information about compute resources
	Modify	Change node state and/or attributes
Queue	Query	Collect detailed policy and configuration information from the resource manager

Using these functions, Moab is able to fully manage workload, resources, and cluster policies. More detailed information about resource manager specific capabilities and limitations for each of these functions can be found in the individual resource manager overviews. (LL, PBS, LSF, SGE, BProc, or WIKI).

Beyond these base functions, other commands exist to support advanced features such as provisioning and cluster level resource management.

Resource Manager Flow

In general, Moab interacts with resource managers in a sequence of steps each scheduling iteration. These steps are outlined in what follows:

1. load global resource information
2. load node specific information (optional)
3. load job information
4. load queue/policy information (optional)
5. cancel/preempt/modify jobs according to cluster policies
6. start jobs in accordance with available resources and policy constraints
7. handle user commands

Typically, each step completes before the next step is started. However, with current systems, size and complexity mandate a more advanced parallel approach providing benefits in the areas of reliability, concurrency, and responsiveness.

Resource Manager Specific Details (Limitations/Special Features)

- TORQUE
  - TORQUE Homepage
- SLURM/Wiki
  - SLURM Integration Guide
  - Wiki Overview

Synchronizing Conflicting Information

Moab does not trust resource manager information. Node, job, and policy information is reloaded on each iteration and discrepancies are detected. Synchronization issues and allocation conflicts are logged and handled where possible. To assist sites in minimizing stale information and conflicts, a number of policies and parameters are available.

- Node State Synchronization Policies (see **NODESYNCTIME** on page 989)
- Stale Data Purging (see **JOBPURGETIME** on page 964)
- Thread Management (preventing resource manager failures from affecting scheduler operation)
- Resource Manager Poll Interval (see **RMPOLLINTERVAL** on page 1015)
- Node Query Refresh Rate (see **NODEPOLLFREQUENCY** on page 985)

Evaluating Resource Manager Availability and Performance

Each resource manager is individually tracked and evaluated by Moab. Using the mdiag -R command, a site can determine how a resource manager is configured, how heavily it is loaded, what failures, if any, have occurred in the recent past, and how responsive it is to requests.

Related topics

- Resource Manager Configuration
- Resource Manager Extensions

Resource Manager Configuration

- Defining and Configuring Resource Manager Interfaces
  - Resource Manager Attributes
- Resource Manager Configuration Details
  - Resource Manager Types
  - Resource Manager Name
  - Resource Manager Location
  - Resource Manager Flags
  - Other Attributes
- Scheduler/Resource Manager Interactions

Defining and Configuring Resource Manager Interfaces

Moab resource manager interfaces are defined using the **RMCFG** on page 1014 parameter. This parameter allows specification of key aspects of the interface. In most cases, only the **TYPE** attribute needs to be specified and Moab determines the needed defaults required to activate and use the selected interface. In the following example, an interface to a Loadleveler resource manager is defined.

```
RMCFG[orion] TYPE=LL...
```

Note that the resource manager is given a label of *orion*. This label can be any arbitrary site-selected string and is for local usage only. For sites with multiple active resource managers, the labels can be used to distinguish between them for resource manager specific queries and commands.

Resource Manager Attributes

The following table lists the possible resource manager attributes that can be configured.

ADMINEXEC	
Format	"jobsubmit"
Default	*NONE*

ADMINEXEC

Description	Normally, when the **JOBSUBMITURL** is executed, Moab will drop to the UID and GID of the user submitting the job. Specifying an **ADMINEXEC** of *jobsubmit* causes Moab to use its own UID and GID instead (usually root). This is useful for some native resource managers where the **JOBSUBMITURL** is not a user command (such as qsub) but a script that interfaces directly with the resource manager.
Example	`RMCFG[base] ADMINEXEC=jobsubmit` *Moab will not use the user's UID and GID for executing the **JOBSUBMITURL**.*

AUTHTYPE

Format	One of *CHECKSUM*, *OTHER*, *PKI*, *SECUREPORT*, or *NONE*.
Default	*CHECKSUM*
Description	Specifies the security protocol to be used in scheduler-resource manager communication. Only valid with WIKI based interfaces.
Example	`RMCFG[base] AUTHTYPE=CHECKSUM` *Moab requires a secret key-based checksum associated with each resource manager message.*

BANDWIDTH

Format:	`<FLOAT>[{M|G|T}]`
Default:	*-1* (unlimited)
Description:	Specifies the maximum deliverable bandwidth between the Moab server and the resource manager for staging jobs and data. Bandwidth is specified in units per second and defaults to a unit of MB/s. If a unit modifier is specified, the value is interpreted accordingly (M - megabytes/sec, G - gigabytes/sec, T - terabytes/sec).

BANDWIDTH	
Example:	`RMCFG[base] BANDWIDTH=340G` *Moab will reserve up to 340 GB of network bandwidth when scheduling job and data staging operations to and from this resource manager.*

CHECKPOINTSIG	
Format	One of *suspend*, *<INTEGER>*, or *SIG<X>*
Description	Specifies what signal to send the resource manager when a job is checkpointed (See Checkpoint Overview.).
Example	`RMCFG[base] CHECKPOINTSIG=SIGKILL` *Moab routes the signal SIGKILL through the resource manager to the job when a job is checkpointed.*

CHECKPOINTTIMEOUT	
Format	`[[[DD:]HH:]MM:]SS`
Default	*0* (no timeout)
Description	Specifies how long Moab waits for a job to checkpoint before canceling it. If set to *0*, Moab does not cancel the job if it fails to checkpoint (See Checkpoint Overview.).
Example	`RMCFG[base] CHECKPOINTTIMEOUT=5:00` *Moab cancels any job that has not exited 5 minutes after receiving a checkpoint request.*

CLIENT	
Format	*<PEER>*
Default	Use name of resource manager for peer client lookup

CLIENT

Description	If specified, the resource manager will use the peer value to authenticate remote connections. (See configuring peers). If not specified, the resource manager will search for a **CLIENTCFG[<X>]** on page 916 entry of `RM:`*<RMNAME>*in the `moab-private.cfg` file.
Example	`RMCFG[clusterBI] CLIENT=clusterB` Moab will look up and use information for peer *clusterB* when authenticating the *clusterBI* resource manager.

CLUSTERQUERYURL

Format	[*file://<path>* | *http://<address>* | *<path>*] If *file://* is specified, Moab treats the destination as a flat text file. If *http://* is specified, Moab treats the destination as a hypertext transfer protocol file. If just a path is specified, Moab treats the destination as an executable.
Description	Specifies how Moab queries the resource manager (See Native RM, URL Notes, and interface details.).
Example	`RMCFG[base] CLUSTERQUERYURL=file:///tmp/cluster.config` *Moab reads* `/tmp/cluster.config` *when it queries base resource manager.*

CONFIGFILE

Format	*<STRING>*
Description	Specifies the resource manager specific configuration file that must be used to enable correct API communication. Only valid with LL- and SLURM-based interfaces.
Example	`RMCFG[base] TYPE=LL CONFIGFILE=/home/loadl/loadl_config` *The scheduler uses the specified file when establishing the resource manager/scheduler interface connection.*

DATARM

Format	*<RM NAME>*
Description	If specified, the resource manager uses the given storage resource manager to handle staging data in and out.
Example	`RMCFG[clusterB] DATARM=clusterB_storage` *When data staging is required by jobs starting/completing on clusterB, Moab uses the storage interface defined by clusterB_storage to stage and monitor the data.*

DEFAULTCLASS

Format	*<STRING>*
Description	Specifies the class to use if jobs submitted via this resource manager interface do not have an associated class.
Example	`RMCFG[internal] DEFAULTCLASS=batch` *Moab assigns the class batch to all jobs from the resource manager internal that do not have a class assigned.* If you are using PBS as the resource manager, a job will never come from PBS without a class, and the default will never apply.

DEFAULTHIGHSPEEDADAPTER

Format:	*<STRING>*
Default:	*sn0*
Description:	Specifies the default high speed switch adapter to use when starting LoadLeveler jobs (supported in version 4.2.2 and higher of Moab and 3.2 of LoadLeveler).
Example:	`RMCFG[base]     DEFAULTHIGHSPEEDADAPTER=sn1` *The scheduler will start jobs requesting a high speed adapter on sn1.*

DESCRIPTION	
Format	*<STRING>*
Description	Specifies the human-readable description for the resource manager interface. If white space is used, the description should be quoted.
Example	`RMCFG[torque] DESCRIPTION='TORQUE RM for launching jobs'` *Moab annotates the TORQUE resource manager accordingly.*

ENV	
Format	Semi-colon-delimited (;) list of *<KEY>=<VALUE>* pairs
Default	*MOABHOMEDIR=<MOABHOMEDIR>*
Description	Specifies a list of environment variables that will be passed to URLs of type *exec://* for that resource manager.
Example	`RMCFG[base] ENV=HOST=node001;RETRYTIME=50` `RMCFG[base] CLUSTERQUERYURL=exec:///opt/moab/tools/cluster.query.pl` `RMCFG[base] WORKLOADQUERYURL=exec:///opt/moab/tools/` `workload.query.pl` `RMCFG[base] ENV=HOST=node001;RETRYTIME=50` `RMCFG[base] CLUSTERQUERYURL=exec:///opt/moab/tools/cluster.query.pl` `RMCFG[base] WORKLOADQUERYURL=exec:///opt/moab/tools/workload.query.pl` *The environment variables HOST and RETRYTIME (with values `node001` and 50 respectively) are passed to the `/opt/moab/tools/cluster.query.pl` and `/opt/moab/tools/workload.query.pl` when they are executed.*

EPORT	
Format:	*<INTEGER>*
Description:	Specifies the event port to use to receive resource manager based scheduling events.
Example:	`RMCFG[base] EPORT=15017` *The scheduler will look for scheduling events from the resource manager host at port 15017.*

FAILTIME

Format:	`[[[DD:]HH:]MM:]SS`
Description:	Specifies how long a resource manager must be down before any failure triggers associated with the resource manager fire.
Example:	`RMCFG[base] FAILTIME=3:00` *If the base resource manager is down for three minutes, any resource manager failure triggers fire.*

FLAGS

Format	Comma-delimited list of zero or more of the following: asyncdelete, asyncstart, *autostart*, *autosync*, *client*, *fullcp*, *executionServer*, *grid*, *hostingCenter*, *ignqueuestate*, *private*, *pushslavejobupdates*, *report*, *shared*, or *static*
Description	Specifies various attributes of the resource manager. See Flag Details for more information.
Example	`RMCFG[base] FLAGS=static` *Moab uses this resource manager to perform a single update of node and job objects reported elsewhere.*

FNLIST

Format	Comma-delimited list of zero or more of the following: *clusterquery*, *jobcancel*, *jobrequeue*, *jobresume*, *jobstart*, *jobsuspend*, *queuequery*, *resourcequery* or *workloadquery*
Description	By default, a resource manager utilizes all functions supported to query and control batch objects. If this parameter is specified, only the listed functions are used.
Example	`RMCFG[base] FNLIST=queuequery` *Moab only uses this resource manager interface to load queue configuration information.*

HOST	
Format	*<STRING>*
Default	*localhost*
Description	The host name of the machine on which the resource manager server is running.
Example	`RMCFG[base] host=server1`

IGNHNODES	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether to read in the PBSPro host nodes. This parameter is used in conjunction with **USEVNODES** on page 611. When both are set to *TRUE*, the host nodes are not queried.
Example	`RMCFG[pbs] IGNHNODES=TRUE`

JOBCANCELURL	
Format	`<protocol>://[<host>[:<port>]][<path>]`
Default	---
Description	Specifies how Moab cancels jobs via the resource manager. (See URL Notes below.)
Example	`RMCFG[base] JOBCANCELURL=exec:///opt/moab/job.cancel.lsf.pl` *Moab executes `/opt/moab/job.cancel.lsf.pl` to cancel specific jobs.*

JOBEXTENDDURATION	
Format	`[[[DD:]HH:]MM:]SS[,[[[DD:]HH:]MM:]SS][!][<]` (or *<MIN TIME>*`[,`*<MAX TIME>*`][!]`)

JOBEXTENDDURATION	
Default	---
Description	Specifies the minimum and maximum amount of time that can be added to a job's walltime if it is possible for the job to be extended. (See MINWCLIMIT.) As the job runs longer than its current specified minimum wallclock limit (-l minwclimit, for example), Moab attempts to extend the job's limit by the minimum **JOBEXTENDDURATION**. This continues until either the extension can no longer occur (it is blocked by a reservation or job), the maximum **JOBEXTENDDURATION** is reached, or the user's specified wallclock limit (-l walltime) is reached. When a job is extended, it is marked as PREEMPTIBLE, unless the `!` is appended to the end of the configuration string. If the < is at the end of the string, however, the job is extended the maximum amount possible. JOBEXTENDDURATION and JOBEXTENDSTARTWALLTIME TRUE cannot be configured together. If they are in the same moab.cfg or are both active, then the JOBEXTENDDURATION will not be honored. For example, comment out the JOBEXTENDSTARTWALLTIME. `RMCFG[base] JOBEXTENDDURATION=30,1:00:00` `#JOBEXTENDSTARTWALLTIME TRUE`
Example	`RMCFG[base] JOBEXTENDDURATION=30,1:00:00` *Moab extends a job's walltime by 30 seconds each time the job is about to run out of walltime until it is bound by one hour, a reservation/job, or the job's original "maximum" wallclock limit.*

JOBIDFORMAT	
Format	*INTEGER*
Default	---
Description	Specifies that Moab should use numbers to create job IDs. This eliminates multiple job IDs associated with a single job.
Example	`RMCFG[base] JOBIDFORMAT=INTEGER` *Job IDs are generated as numbers.*

JOBMODIFYURL

Format	*<protocol>*://[*<host>*[:*<port>*]][*<path>*]
Default	---
Description	Specifies how Moab modifies jobs via the resource manager. (See URL Notes, and interface details.)
Example	`RMCFG[base] JOBMODIFYURL=exec://$TOOLSDIR/job.modify.dyn.pl` *Moab executes* `/opt/moab/job.modify.dyn.pl` *to modify specific jobs.*

JOBRSVRECREATE

Format	Boolean
Default	*TRUE*
Description	Specifies whether Moab will re-create a job reservation each time job information is updated by a resource manager (See Considerations for Large Clusters for more information.).
Example	`RMCFG[base] JOBRSVRECREATE=FALSE` *Moab only creates a job reservation once when the job first starts.*

JOBSTARTURL

Format	*<protocol>*://[*<host>*[:*<port>*]][*<path>*]
Default	*TRUE*
Description	Specifies how Moab starts jobs via the resource manager. (See URL Notes below.)
Example	`RMCFG[base] JOBSTARTURL=http://orion.bsu.edu:1322/moab/jobstart.cgi` *Moab triggers the* `jobstart.cgi` *script via http to start specific jobs.*

JOBSUBMITURL

Format	*<protocol>*://[*<host>*[:*<port>*]][*<path>*]
Description	Specifies how Moab submits jobs to the resource manager (See URL Notes below.).
Example	`RMCFG[base] JOBSUBMITURL=exec://$TOOLSDIR/job.submit.dyn.pl` *Moab submits jobs directly to the database located on host* `dbserver.flc.com`.

JOBSUSPENDURL

Format	*<protocol>*://[*<host>*[:*<port>*]][*<path>*]
Description	Specifies how Moab suspends jobs via the resource manager. (See URL Notes below.)
Example	`RMCFG[base] JOBSUSPENDURL=EXEC://$HOME/scripts/job.suspend` *Moab executes the* `job.suspend` *script when jobs are suspended.*

JOBVALIDATEURL

Format	*<protocol>*://[*<host>*[:*<port>*]][*<path>*]
Description	Specifies how Moab validates newly submitted jobs (See URL Notes below.). If the script returns with a non-zero exit code, the job is rejected. (See User Proxying/Alternate Credentials.)
Example	`RMCFG[base] JOBVALIDATEURL=exec://$TOOLS/job.validate.pl` *Moab executes the '*`job.validate.pl`*' script when jobs are submitted to verify they are acceptable.*

MAXDSOP

Format	*<INTEGER>*
Default	*-1* (unlimited)

MAXDSOP	
Description	Specifies the maximum number of data staging operations that may be simultaneously active.
Example	`RMCFG[ds] MAXDSOP=16`

MAXITERATIONFAILURECOUNT	
Format	*<INTEGER>*
Default	*80*
Description	Specifies the number of times the RM must fail within a certain iteration before Moab considers it down or corrupt. When an RM is down or corrupt, Moab will not attempt to interact with it.
Example	`RMCFG[base] MAXITERATIONFAILURECOUNT=25` *The RM `base` must fail 25 times in a single iteration for Moab to consider it down and cease interacting with it.*

MAXJOBPERMINUTE	
Format	*<INTEGER>*
Default	*-1* (unlimited)
Description	Specifies the maximum number of jobs allowed to start per minute via the resource manager.
Example	`RMCFG[base] MAXJOBPERMINUTE=5` *The scheduler only allows five jobs per minute to launch via the resource manager base.*

MAXJOBS	
Format	*<INTEGER>*

MAXJOBS	
Default	*0* (limited only by the Moab **MAXJOB** setting)
Description	Specifies the maximum number of active jobs that this interface is allowed to load from the resource manager. Only works with Moab peer resource managers at this time.
Example	`RMCFG[cluster1] SERVER=moab://cluster1 MAXJOBS=200` *The scheduler loads up to 200 active jobs from the remote Moab peer cluster1.*

MINETIME	
Format	*<INTEGER>*
Default	*1*
Description	Specifies the minimum time in seconds between processing subsequent scheduling events.
Example	`RMCFG[base] MINETIME=5` *The scheduler batch-processes scheduling events that occur less than five seconds apart.*

NMPORT	
Format	*<INTEGER>*
Default	(any valid port number)
Description	Allows specification of the resource manager's node manager port and is only required when this port has been set to a non-default value.
Example	`RMCFG[base] NMPORT=13001` *The scheduler contacts the node manager located on each compute node at port 13001.*

NODEFAILURERSVPROFILE	
Format	*<STRING>*
Description	Specifies the rsv template to use when placing a reservation onto failed nodes (See also **NODEFAILURERESERVETIME** on page 983.).
Example	`# moab.cfg` `RMCFG[base] NODEFAILURERSVPROFILE=long` `RSVPROFILE[long]        DURATION=25:00RSVPROFILE[long]        USERLIST=john` *The scheduler will use the long rsv profile when creating reservations over failed nodes belonging to base.*

NODESTATEPOLICY	
Format	One of *OPTIMISTIC* or *PESSIMISTIC*
Default	*PESSIMISTIC*
Description	Specifies how Moab should determine the state of a node when multiple resource managers are reporting state. *OPTIMISTIC* specifies that if any resource manager reports a state of up, that state will be used. *PESSIMISTIC* specifies that if any resource manager reports a state of down, that state will be used.
Example	`# moab.cfg` `RMCFG[native] TYPE=NATIVE NODESTATEPOLICY=OPTIMISTIC`

OMAP	
Format	*<protocol>*://[*<host>*[:*<port>*]][*<path>*]
Description	Specifies an object map file that is used to map credentials and other objects when using this resource manager peer (See Grid Credential Management for full details.).
Example	`moab.cfg` `RMCFG[peer1] OMAP=file:///opt/moab/omap.dat` *When communicating with the resource manager peer1, objects are mapped according to the rules defined in the* `/opt/moab/omap.dat` *file.*

PORT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the port on which the scheduler should contact the associated resource manager. The value *0* specifies that the resource manager default port should be used.
Example	`RMCFG[base] TYPE=PBS HOST=cws PORT=20001` *Moab attempts to contact the PBS server daemon on host cws, port 20001.*

PROVDURATION	
Format	`[[[DD:]HH:]MM:]SS`
Default	*2:30*
Description	Specifies the upper bound (walltime) of a provisioning request. After this duration, Moab will consider the provisioning attempt failed.
Example	`RMCFG[base] PROVDURATION=5:00` *When RM `base` provisions a node for more than 5 minutes, Moab considers the provisioning as having failed.*

PTYSTRING	
Format	*<STRING>*
Default	*srun -n1 -N1 --pty*

PTYSTRING

Description	When a SLURM interactive job is submitted, it builds an salloc command that gets the requested resources and an srun command that creates a terminal session on one of the nodes. The srun command is called the PTYString. PTYString is configured in `moab.cfg`. There are two special things you can do with PTYString: 1. You can have `PTYSTRING=$salloc` which says to use the default salloc command (SallocDefaultCommand, look in the `slurm.conf` man page) defined in `slurm.conf`. Internally, Moab won't add a PTYString because SLURM will call the `SallocDefaultCommand`. 2. As in the example below, you can add `$SHELL`. `$SHELL` will be expanded to either what you request on the command line (such as msub -S /bin/tcsh -l) or to the value of `$SHELL` in your current session. PTYString works only with SLURM.
Example	`RMCFG[slurm] PTYSTRING="srun -n1 -N1 --pty --preserve-env $SHELL"`

RESOURCECREATEURL

Format	*<STRING>*
Default	[*exec://<path>* | *http://<address>* | *<path>*] If *exec://* is specified, Moab treats the destination as an executable file; if *http://* is specified, Moab treats the destination as a hypertext transfer protocol file.
Description	Specifies a script or method that can be used by Moab to create resources dynamically, such as creating a virtual machine on a hypervisor.
Example	`RMCFG[base] RESOURCECREATEURL=exec:///opt/script/vm.provision.py` *Moab invokes the `vm.provision.py` script, passing in data as command line arguments, to request a creation of new resources.*

RESOURCETYPE

Format	{*COMPUTE*|*FS*|*LICENSE*|*NETWORK*|*PROV*}
Description	Specifies which type of resource this resource manager is configured to control. See Native Resource Managers for more information.

RESOURCETYPE	
Example	`RMCFG[base] TYPE=NATIVE RESOURCETYPE=FS` *Resource manager base will function as a NATIVE resource manager and control file systems.*

RMSTARTURL	
Format	[*exec://<path>* | *http://<address>* | *<path>*] If *exec://* is specified, Moab treats the destination as an executable file; if *http://* is specified, Moab treats the destination as a hypertext transfer protocol file.
Description	Specifies how Moab starts the resource manager.
Example	`RMCFG[base] RMSTARTURL=exec:///tmp/nat.start.pl` *Moab executes `/tmp/nat.start.pl` to start the resource manager base.*

RMSTOPURL	
Format	[*exec://<path>* | *http://<address>* | *<path>*] If *exec://* is specified, Moab treats the destination as an executable file; if *http://* is specified, Moab treats the destination as a hypertext transfer protocol file.
Description	Specifies how Moab stops the resource manager.
Example	`RMCFG[base] RMSTOPURL=exec:///tmp/nat.stop.pl` *Moab executes `/tmp/nat.stop.pl` to stop the resource manager base.*

SBINDIR	
Format	*<PATH>*
Description	For use with TORQUE; specifies the location of the TORQUE system binaries (supported in TORQUE 1.2.0p4 and higher).

SBINDIR	
Example	`RMCFG[base] TYPE=pbs  SBINDIR=/usr/local/torque/sbin` *Moab tells TORQUE that its system binaries are located in `/usr/local/torque/sbin`.*

SERVER	
Format	*<URL>*
Description	Specifies the resource management service to use. If not specified, the scheduler locates the resource manager via built-in defaults or, if available, with an information service.
Example	`RMCFG[base] server=ll://supercluster.org:9705` *Moab attempts to use the Loadleveler scheduling API at the specified location.*

SLURMFLAGS	
Format	*<STRING>*
Description	Specifies characteristics of the SLURM resource manager interface. The *COMPRESSOUTPUT* flag instructs Moab to use the compact hostlist format for job submissions to SLURM. The flag *NODEDELTAQUERY* instructs Moab to request delta node updates when it queries SLURM for node configuration.
Example	`RMCFG[slurm] SLURMFLAGS=COMPRESSOUTPUT` *Moab uses the COMPRESSOUTPUT flag to determine interface characteristics with SLURM.*

SOFTTERMSIG	
Format	*<INTEGER>* or SIG*<X>*
Description	Specifies what signal to send the resource manager when a job reaches its soft wallclock limit. (See JOBMAXOVERRUN.)

SOFTTERMSIG	
Example	`RMCFG[base] SOFTTERMSIG=SIGUSR1` *Moab routes the signal SIGUSR1 through the resource manager to the job when a job reaches its soft wallclock limit.*

STAGETHRESHOLD	
Format	`[[[DD:]HH:]MM:]SS`
Description	Specifies the maximum time a job waits to start locally before considering being migrated to a remote peer. In other words, if a job's start time on a remote cluster is less than the start time on the local cluster, but the difference between the two is less than **STAGETHRESHOLD**, then the job is scheduled locally. The aim is to avoid job/data staging overhead if the difference in start times is minimal. If this attribute is used, backfill is disabled for the associated resource manager.
Example	`RMCFG[remote_cluster] STAGETHRESHOLD=00:05:00` *Moab only migrates jobs to remote_cluster if the jobs can start five minutes sooner on the remote cluster than they could on the local cluster.*

STARTCMD	
Format	*<STRING>*
Description	Specifies the full path to the resource manager job start client. If the resource manager API fails, Moab executes the specified start command in a second attempt to start the job. Moab calls the start command with the format *<CMD><JOBID>* `-H` *<HOSTLIST>* unless the environment variable MOABNOHOSTLIST is set in which case Moab will only pass the job ID.
Example	`RMCFG[base] STARTCMD=/usr/local/bin/qrun` *Moab uses the specified start command if API failures occur when launching jobs.*

SUBMITCMD	
Format	*<STRING>*
Description	Specifies the full path to the resource manager job submission client.
Example	`RMCFG[base] SUBMITCMD=/usr/local/bin/qsub` *Moab uses the specified submit command when migrating jobs.*

SUBMITPOLICY	
Format	One of *NODECENTRIC* or *PROCCENTRIC*
Default	*PROCCENTRIC*
Description	If set to *NODECENTRIC*, each specified node requested by the job is interpreted as a true compute host, not as a task or processor.
Example	`RMCFG[base] SUBMITPOLICY=NODECENTRIC` *Moab uses the specified submit policy when migrating jobs.*

SUSPENDSIG	
Format	*<INTEGER>* (valid UNIX signal between 1 and 64)
Default	RM-specific default
Description	If set, Moab sends the specified signal to a job when a job suspend request is issued.
Example	`RMCFG[base] SUSPENDSIG=19` *Moab uses the specified suspend signal when suspending jobs within the base resource manager.* **SUSPENDSIG** should not be used with TORQUE or other PBS-based resource managers.

SYNCJOBID	
Format	*<BOOLEAN>*
Description	Specifies that Moab should migrate jobs to the local resource manager with the job's Moab-assigned job ID. In a grid, the grid-head will only pass dependencies to the underlying Moab if **SYNCJOBID** is set. This attribute can be used with the **JOBIDFORMAT** on page 597 attribute and **PROXYJOBSUBMISSION** on page 617 flag in order to synchronize job IDs between Moab and the resource manager. For more information about all steps necessary to synchronize job IDs between Moab and TORQUE, see **Synchronizing Job IDs in TORQUE and Moab** on page 613.
Example	`RMCFG[slurm] TYPE=wiki:slurm SYNCJOBID=TRUE`

SYSTEMMODIFYURL	
Format	[*exec://<path>* | *http://<address>* | *<path>*] If *exec://* is specified, Moab treats the destination as an executable file; if *http://* is specified, Moab treats the destination as a hypertext transfer protocol file.
Description	Specifies how Moab modifies attributes of the system. This interface is used in data staging.
Example	`RMCFG[base] SYSTEMMODIFYURL=exec:///tmp/system.modify.pl` *Moab executes `/tmp/system.modify.pl` when it modifies system attributes in conjunction with the resource manager base.*

SYSTEMQUERYURL	
Format	[*exec://<path>* | *http://<address>* | *<path>*] If *file://* is specified, Moab treats the destination as a flat text file; if *http://* is specified, Moab treats the destination as a hypertext transfer protocol file; if just a path is specified, Moab treats the destination as an executable.
Description	Specifies how Moab queries attributes of the system. This interface is used in data staging.
Example	`RMCFG[base] SYSTEMQUERYURL=file:///tmp/system.query` *Moab reads `/tmp/system.query` when it queries the system in conjunction with base resource manager.*

TARGETUSAGE	
Format	*<INTEGER>*[%]
Default	90%
Description	Amount of resource manager resources to explicitly use. In the case of a storage resource manager, indicates the target usage of data storage resources to dedicate to active data migration requests. If the specified value contains a percent sign (%), the target value is a percent of the configured value. Otherwise, the target value is considered to be an absolute value measured in megabytes (MB).
Example	`RMCFG[storage] TYPE=NATIVE RESOURCETYPE=storage` `RMCFG[storage] TARGETUSAGE=80%` *Moab schedules data migration requests to never exceed 80% usage of the storage resource manager's disk cache and network resources.*

TIMEOUT	
Format	*<INTEGER>*
Default	*30*
Description	Time (in seconds) the scheduler waits for a response from the resource manager.
Example	`RMCFG[base] TIMEOUT=40` *Moab waits 40 seconds to receive a response from the resource manager before timing out and giving up. Moab tries again on the next iteration.*

TRIGGER	
Format	*<TRIG_SPEC>*
Description	A trigger specification indicating behaviors to enforce in the event of certain events associated with the resource manager, including resource manager start, stop, and failure.
Example	`RMCFG[base] TRIGGER=<X>`

TYPE	
Format	*<RMTYPE>*[:*<RMSUBTYPE>*] where *<RMTYPE>* is one of the following: TORQUE, NATIVE, PBS, RMS, SSS, or WIKI and the optional *<RMSUBTYPE>* value is one of RMS.
Default	PBS
Description	Specifies type of resource manager to be contacted by the scheduler. For TYPE WIKI, AUTHTYPE must be set to CHECKSUM. The *<RMSUBTYPE>* option is currently only used to support Compaq's RMS resource manager in conjunction with PBS. In this case, the value `PBS:RMS` should be specified.
Example	`RMCFG[clusterA] TYPE=PBS HOST=clusterA PORT=15003` `RMCFG[clusterB] TYPE=PBS HOST=clusterB PORT=15005` *Moab interfaces to two different PBS resource managers, one located on server clusterA at port 15003 and one located on server clusterB at port 15005.*

USEVNODES	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether to schedule on PBS virtual nodes. When set to *TRUE*, Moab queries PBSPro for vnodes and puts jobs on vnodes rather than hosts. In some systems, such as PBS + Altix, it may not be desirable to read in the host nodes; for such situations refer to the IGNHNODES attribute.
Example	`RMCFG[pbs] USEVNODES=TRUE`

VARIABLES	
Format	*<VAR>*=*<VAL>*[,*<VAR>*=*<VAL>*]
Description	Opaque resource manager variables.
Example	`RMCFG[base] VARIABLES=SCHEDDHOST=head1` *Moab associates the variable SCHEDDHOST with the value head1 on resource manager base.*

VERSION	
Format	<STRING>
Default	*SLURM: 10200* (i.e., 1.2.0)
Description	Resource manager-specific version string.
Example	`RMCFG[base] VERSION=10124` *Moab assumes that resource manager base has a version number of 1.1.24.*

VMOWNERRM	
Format	*<STRING>*
Description	Used with provisioning resource managers that can create VMs. It specifies the resource manager that will own any VMs created by the resource manager.
Example	`RMCFG[torque]` `RMCFG[prov] RESOURCETYPE=PROV VMOWNERRM=torque`

WORKLOADQUERYURL	
Format	[*file://<path>* | *http://<address>* | *<path>*] If *file://* is specified, Moab treats the destination as a flat text file; if *http://* is specified, Moab treats the destination as a hypertext transfer protocol file; if just a path is specified, Moab treats the destination as an executable.
Description	Specifies how Moab queries the resource manager for workload information. (See Native RM, URL Notes, and interface details.)
Example	`RMCFG[TORQUE] WORKLOADQUERYURL=exec://$TOOLSDIR/job.query.dyn.pl` *Moab executes* `/opt/moab/tools/job.query.dyn.pl` *to obtain updated workload information from resource manager TORQUE.*

URL notes

URL parameters can load files by using the *file*, *exec*, and *http* protocols.

For the protocol *file*, Moab loads the data directly from the text file pointed to by path.

```
RMCFG[base] SYSTEMQUERYURL=file:///tmp/system.query
```

For the protocol *exec*, Moab executes the file pointed to by path and loads the output written to STDOUT. If the script requires arguments, you can use a question mark (?) between the script name and the arguments, and an ampersand (&) for each space.

```
RMCFG[base] JOBVALIDATEURL=exec://$TOOLS/job.validate.pl
RMCFG[native] CLUSTERQUERYURL=exec://opt/moab/tools/cluster.query.pl?-group=group1&-
arch=x86
```

Synchronizing Job IDs in TORQUE and Moab

> Unless you use an **msub** on page 290 submit filter or you're in a grid, it is recommended that you use your RM-specific job submission command (for instance, qsub).

In order to synchronize your job IDs between TORQUE and Moab you must perform the following steps:

1. Verify that you are using TORQUE version 2.5.6 or later.
2. Set **SYNCJOBID** on page 609 to *TRUE* in all resource managers.

   ```
   RMCFG[torque] TYPE=PBS SYNCJOBID=TRUE
   ```

3. Set the **PROXYJOBSUBMISSION** on page 617 flag. With *PROXYJOBSUBMISSION* enabled, you must run Moab as a TORQUE manager or operator. Verify that other users can submit jobs using **msub** on page 290. Moab, as a non-root user, should still be able to submit jobs to TORQUE and synchronize job IDs.

   ```
   RMCFG[torque] TYPE=PBS SYNCJOBID=TRUE
   RMCFG[torque] FLAGS=PROXYJOBSUBMISSION
   ```

4. Add **JOBIDFORMAT** on page 597=*INTEGER* to the internal RM. Adding this parameter forces Moab to only use numbers as job IDs and those numbers to synchronize across Moab, TORQUE, and the entire grid. This enhances the end-user experience as it eliminates multiple job IDs associated with a single job.

   ```
   RMCFG[torque] TYPE=PBS SYNCJOBID=TRUE
   RMCFG[torque] FLAGS=PROXYJOBSUBMISSION

   RMCFG[internal] JOBIDFORMAT=INTEGER
   ```

Resource Manager Configuration Details

As with all scheduler parameters, follows the syntax described within the Parameters Overview.

Resource Manager Types

The **RMCFG** parameter allows the scheduler to interface to multiple types of resource managers using the **TYPE** or **SERVER** attributes. Specifying these attributes, any of the following listed resource managers may be supported.

Type	Resource managers	Details
Moab	Moab Workload Manager	Use the Moab peer-to-peer (grid) capabilities to enable grids and other configurations. (See Grid Configuration.)
MWS	Moab Web Services	The MWS resource manager type is a native integration between Moab and MWS. Resource manager data is passed directly between Moab and MWS using JSON (rather than Moab's native WIKI syntax). This simplifies RM configuration for systems where one or more MWS plugins are acting as resource managers. See the "Moab Workload Manager resource manager integration" section of the MWS plugins chapter in the MWS documentation for more information.
Native	Moab *Native* Interface	Used for connecting directly to scripts, files, and databases. (See Managing Resources Directly with the Native Interface.)
PBS	TORQUE (all versions)	N/A
SSS	Scalable Systems Software Project version 2.0 and higher	N/A
WIKI	Wiki interface specification version 1.0 and higher	Used for LRM, YRM, ClubMASK, BProc, SLURM, and others.

Resource Manager Name

Moab can support more than one resource manager simultaneously. Consequently, the **RMCFG** parameter takes an index value such as `RMCFG[clusterA]`. This index value essentially names the resource manager (as done by the deprecated parameter **RMNAME**). The resource manager name is used by the scheduler in diagnostic displays, logging, and in reporting resource consumption to the allocation manager. For most environments, the selection of the resource manager name can be arbitrary.

Resource Manager Location

The **HOST**, **PORT**, and **SERVER** attributes can be used to specify how the resource manager should be contacted. For many resource managers the interface correctly establishes contact using default values. These parameters need only to be specified for resource managers such as the WIKI interface (that do not include defaults) or with resources managers that can be configured to run at non-standard locations (such as PBS). In all other cases, the resource manager is automatically located.

Resource Manager Flags

The **FLAGS** attribute can be used to modify many aspects of a resources manager's behavior.

Flag	Description
ASYNCSTART	Jobs started on this resource manager start asynchronously. In this case, the scheduler does not wait for confirmation that the job correctly starts before proceeding. (See Large Cluster Tuning for more information.)
AUTOSTART	Jobs staged to this resource manager do not need to be explicitly started by the scheduler. The resource manager itself handles job launch.
AUTOSYNC	Resource manager starts and stops together with Moab. This requires that the resource manager support a resource manager start and stop API or the RMSTARTURL and RMSTOPURL attributes are set.
BECOMEMASTER	Nodes reported by this resource manager will transfer ownership to this resource manager if they are currently owned by another resource manager that does not have this flag set.
CLIENT	A client resource manager object is created for diagnostic/statistical purposes or to configure Moab's interaction with this resource manager. It represents an external entity that consumes server resources or services, allows a local administrator to track this usage, and configures specific policies related to that resource manager. A client resource manager object loads no data and provides no services.
CLOCKSKEWCHECKING	Setting *CLOCKSKEWCHECKING* allows you to configure clock skew adjustments. Most of the time it is sufficient to use an NTP server to keep the clocks in your system synchronized.
COLLAPSEDVIEW	Does not work — not supported The resource manager masks details about local workload and resources and presents only information relevant to the remote server.
DYNAMICCRED	The resource manager creates credentials within the cluster as needed to support workload. (See Identity Manager Overview.)
EXECUTIONSERVER	The resource manager is capable of launching and executing batch workload.

Flag	Description
FSISREMOTE	Add this flag if the working file system doesn't exist on the server to prevent Moab from validating files and directories at migration.
FULLCP	Always checkpoint full job information (useful with Native resource managers).
HOSTINGCENTER	The resource manager interface is used to negotiate an adjustment in dynamic resource access.
IGNQUEUESTATE	The queue state reported by the resource manager should be ignored. May be used if queues must be disabled inside of a particular resource manager to allow an external scheduler to properly operate.
IGNWORKLOADSTATE	When this flag is applied to a native resource manager, any jobs that are reported via that resource manager's "workload query URL" have their reported state ignored. For example, if an RM has the *IgnWorkloadState* flag and it reports that a set of jobs have a state of "Running," this state is ignored and the jobs will either have a default state set or will inherit the state from another RM reporting on that same set of jobs. This flag only changes the behavior of RMs of type *NATIVE*.
LOCALWORKLOADEXPORT	When set, destination peers share information about local and remote jobs, allowing job management of different clusters at a single peer. For more information, see Workload Submission and Control.
MIGRATEALLJOBATTRIBUTES	When set, this flag causes additional job information to be migrated to the resource manager; additional job information includes things such as node features applied via `CLASSCFG[name] DEFAULT.FEATURES`, the account to which the job was submitted, and job walltime limit.
NOAUTORES	If the resource manager does not report CPU usage to Moab because CPU usage is at 0%, Moab assumes full CPU usage. When set, Moab recognizes the resource manager report as 0% usage. This is only valid for PBS.
NOCREATERESOURCE	To use resources discovered from this resource manager, they must be created by another resource manager first. For example, if you set *NOCREATERESOURCE* on RM `A`, which reports nodes 1 and 2, and RM `B` only reports node 1, then node 2 will not be created because RM `B` did not report it.
PRIVATE	The resources and workload reported by the resource manager are not reported to non-administrator users.

Flag	Description
PROXYJOBSUBMISSION	Enables Admin proxy job submission, which means administrators may submit jobs in behalf of other users.
PUSHSLAVEJOBUPDATES	Enables job changes made on a grid slave to be pushed to the grid head or master. Without this flag, jobs being reported to the grid head do not show any changes made on the remote Moab server (via mjobctl and so forth).
RECORDGPUMETRICS	Enables the recording of GPU metrics for nodes.
RECORDMICMETRICS	Enables the recording of MIC metrics for nodes.
REPORT	N/A
SHARED	Resources of this resource manager may be scheduled by multiple independent sources and may not be assumed to be owned by any single source.
STATIC	This resource manager only provides partial object information and this information does not change over time. Consequently, this resource manager may only be called once per object to modify job and node information.
USERSPACEISSEPARATE	This tells Moab to ignore validating the user's uid and gid in the case that information doesn't exist on the Moab server.

Example

```
# resource manager 'torque' should use asynchronous job start
# and report resources in 'grid' mode
RMCFG[torque] FLAGS=asyncstart,grid
```

Scheduler/Resource Manager Interactions

In the simplest configuration, Moab interacts with the resource manager using the following four primary functions:

Function	Description
GETJOBINFO	Collect detailed state and requirement information about idle, running, and recently completed jobs.
GETNODEINFO	Collect detailed state information about idle, busy, and defined nodes.

Function	Description
STARTJOB	Immediately start a specific job on a particular set of nodes.
CANCELJOB	Immediately cancel a specific job regardless of job state.

Using these four simple commands, Moab enables nearly its entire suite of scheduling functions. More detailed information about resource manager specific requirements and semantics for each of these commands can be found in the specific resource manager (such as WIKI) overviews.

In addition to these base commands, other commands are required to support advanced features such as suspend/resume, gang scheduling, and scheduler initiated checkpoint restart.

Information on creating a new scheduler resource manager interface can be found in the Adding New Resource Manager Interfaces section.

Resource Manager Extensions

- Resource Manager Extension Specification
- Resource Manager Extension Values
- Resource Manager Extension Examples

All resource managers are not created equal. There is a wide range in what capabilities are available from system to system. Additionally, there is a large body of functionality that many, if not all, resource managers have no concept of. A good example of this is job QoS. Since most resource managers do not have a concept of quality of service, they do not provide a mechanism for users to specify this information. In many cases, Moab is able to add capabilities at a global level. However, a number of features require a *per job* specification. Resource manager extensions allow this information to be associated with the job.

Resource Manager Extension Specification

Specifying resource manager extensions varies by resource manager. TORQUE, OpenPBS, PBSPro, Loadleveler, LSF, S3, and Wiki each allow the specification of an *extension* field as described in the following table:

Resource manager	Specification method
TORQUE 2.0+	-l `> qsub -l nodes=3,qos=high sleepy.cmd`

<table>
<tr><th>Resource manager</th><th>Specification method</th></tr>
<tr><td>TORQUE 1.x/OpenPBS</td><td>-W x=<pre>> qsub -l nodes=3 -W x=qos:high sleepy.cmd</pre>OpenPBS does not support this ability by default but can be patched as described in the PBS Resource Manager Extension Overview.</td></tr>
<tr><td>Loadleveler</td><td>#@comment<pre>#@nodes = 3
#@comment = qos:high</pre></td></tr>
<tr><td>LSF</td><td>-ext<pre>> bsub -ext advres:system.2</pre></td></tr>
<tr><td>PBSPro</td><td>-l<pre>> qsub -l advres=system.2</pre>Use of PBSPro resources requires configuring the <code>server_priv/resourcedef</code> file to define the needed extensions as in the following example:<pre>advres type=string
qos   type=string
sid   type=string
sjid   type=string</pre></td></tr>
<tr><td>Wiki</td><td>comment<pre>comment=qos:high</pre></td></tr>
</table>

Resource Manager Extension Values

Using the resource manager specific method, the following job extensions are currently available:

ADVRES

Format	`[!]`*`<RSVID>`*
Description	Specifies that reserved resources are required to run the job. If *<RSVID>* is specified, then only resources within the specified reservation may be allocated (see Job to Reservation Binding). You can request to not use a specific reservation by using `advres=!`*`<reservationname>`*.
Example	`> qsub -l advres=grid.3` *Resources for the job must come from grid.3.* `> qsub -l advres=!grid.5` *Resources for the job must not come from grid.5*

BANDWIDTH

Format	*`<DOUBLE>`* (in MB/s)
Description	Minimum available network bandwidth across allocated resources (See Network Management).

BANDWIDTH	
Example	`> bsub -ext bandwidth=120 chemjob.txt`

CPUCLOCK	
Format	*<STRING>*

CPUCLOCK	
Description	Specify the CPU clock frequency for each node requested for this job. A **cpuclock** request applies to every processor on every node in the request. Specifying varying CPU frequencies for different nodes or different processors on nodes in a single job request is not supported. Not all CPUs support all possible frequencies or ACPI states. If the requested frequency is not supported by the CPU, the nearest frequency is used. Using **cpuclock** sets **NODEACCESSPOLICY** to *SINGLEJOB*. ALPS 1.4 or later is required when using **cpuclock** on Cray. The clock frequency can be specified via: • a number that indicates the clock frequency (with or without the SI unit suffix). • a Linux power governor policy name. The governor names are: ◦ *performance*: This governor instructs Linux to operate each logical processor at its maximum clock frequency. This setting consumes the most power and workload executes at the fastest possible speed. ◦ *powersave*: This governor instructs Linux to operate each logical processor at its minimum clock frequency. This setting executes workload at the slowest possible speed. This setting does not necessarily consume the least amount of power since applications execute slower, and may actually consume more energy because of the additional time needed to complete the workload's execution. ◦ *ondemand*: This governor dynamically switches the logical processor's clock frequency to the maximum value when system load is high and to the minimum value when the system load is low. This setting causes workload to execute at the fastest possible speed or the slowest possible speed, depending on OS load. The system switches between consuming the most power and the least power. The power saving benefits of *ondemand* might be non-existent due to frequency switching latency if the system load causes clock frequency changes too often. This has been true for older processors since changing the clock frequency required putting the processor into the C3 "sleep" state, changing its clock frequency, and then waking it up, all of which required a significant amount of time. Newer processors, such as the Intel Xeon E5-2600 Sandy Bridge processors, can change clock frequency dynamically and much faster. ◦ *conservative*: This governor operates like the *ondemand* governor but is more conservative in switching between frequencies. It switches more gradually and uses all possible clock frequencies. This governor can switch to an intermediate clock frequency if it seems appropriate to the system load and usage, which the *ondemand* governor does not do. • an ACPI performance state (or P-state) with or without the P prefix. P-states are a special

CPUCLOCK	
	range of values (0-15) that map to specific frequencies. Not all processors support all 16 states, however, they all start at `P0`. `P0` sets the CPU clock frequency to the highest performance state which runs at the maximum frequency. `P15` sets the CPU clock frequency to the lowest performance state which runs at the lowest frequency. When reviewing job or node properties when **cpuclock** was used, be mindful of unit conversion. The OS reports frequency in Hz, not MHz or GHz.
Example	`msub -l cpuclock=1800,nodes=2 script.sh` `msub -l cpuclock=1800mhz,nodes=2 script.sh` *This job requests 2 nodes and specifies their CPU frequencies should be set to 1800 MHz.* `msub -l cpuclock=performance,nodes=2 script.sh` *This job requests 2 nodes and specifies their CPU frequencies should be set to the performance power governor policy.* `msub -l cpuclock=3,nodes=2 script.sh` `msub -l cpuclock=p3,nodes=2 script.sh` *This job requests 2 nodes and specifies their CPU frequencies should be set to a performance state of 3.*

DDISK	
Format	*<INTEGER>*
Default	*0*
Description	Dedicated disk per task in MB.
Example	`> qsub -l ddisk=2000`

DEADLINE	
Format	Relative time: `[[[DD:]HH:]MM:]SS` Absolute time: `hh:mm:ss_mm/dd/yy`
Description	Either the relative completion deadline of job (from job submission time) or an absolute deadline in which you specify the date and time the job will finish.

DEADLINE	
Example:	`> qsub -l deadline=2:00:00,nodes=4 /tmp/bio3.cmd` *The job's deadline is 2 hours after its submission.*

DEPEND	
Format	[*<DEPENDTYPE>*:][{jobname|jobid}.]*<ID>*[:[{jobname|jobid}.]*<ID>*]...
Description	Allows specification of job dependencies for compute or system jobs. If no ID prefix (jobname or jobid) is specified, the ID value is interpreted as a job ID.
Example	`# submit job which will run after job 1301 and 1304 complete` `> msub -l depend=orion.1301:orion.1304 test.cmd` `orion.1322` `# submit jobname-based dependency job` `> msub -l depend=jobname.data1005 dataetl.cmd` `orion.1428`

DMEM	
Format	*<INTEGER>*
Default	*0*
Description	Dedicated memory per task in bytes.
Example	`> msub -l dmem=20480` *Moab will dedicate 20 MB of memory to the task.*

EPILOGUE	
Format	*<STRING>*

EPILOGUE	
Description	Specifies a user owned epilogue script which is run before the system epilogue and `epilogue.user` scripts at the completion of a job. The syntax is `epilogue=<file>`. The file can be designated with an absolute or relative path. This parameter works only with TORQUE.
Example	`> msub -l epilogue=epilogue_script.sh job.sh`

EXCLUDENODES	
Format	`{<nodeid>|<node_range>}[:...]`
Description	Specifies nodes that should not be considered for the given job.
Example	`> msub -l excludenodes=k1:k2:k[5-8]` `# Comma separated ranges work only with SLURM` `> msub -l excludenodes=k[1-2,5-8]`

FEATURE	
Format	`<FEATURE>[{:|}<FEATURE>]...`
Description	Required list of node attribute/node features. If the *pipe* (|) character is used as a delimiter, the features are logically OR'd together and the associated job may use resources that match any of the specified features.
Example	`> qsub -l feature='fastos:bigio' testjob.cmd`

GATTR	
Format	`<STRING>`
Description	Generic job attribute associated with job. The maximum size for an attribute is 63 bytes (the core Moab size limit of 64, including a null byte)
Example	`> qsub -l gattr=bigjob`

GEOMETRY	
Format:	`{(<TASKID>[,<TASKID>[,...]])[(<TASKID>[,...])...]}`
Description:	Explicitly specified task geometry.
Example:	`> qsub -l nodes=2:ppn=4 -W x=geometry:'{(0,1,4,5)(2,3,6,7)}' quanta2.cmd` *The job quanta2.cmd runs tasks 0, 1, 4, and 5 on one node, while tasks 2, 3, 6, and 7 run on another node.*

GMETRIC	
Format	Generic metric requirement for allocated nodes where the requirement is specified using the format `<GMNAME>[:{lt:,le:,eq:,ge:,gt:,ne:}<VALUE>]`
Description	Indicates generic constraints that must be found on all allocated nodes. If a *<VALUE>* is not specified, the node must simply possess the generic metric (See Generic Metrics for more information.).
Example	`> qsub -l gmetric=bioversion:ge:133244 testj.txt`

GPUs	
Format	`msub -l nodes=<VALUE>:ppn=<VALUE>:gpus=<VALUE>[:mode][:reseterr]` Where `mode` is one of: *exclusive* - The default setting. The GPU is used exclusively by one process thread. *exclusive_thread* - The GPU is used exclusively by one process thread. *exclusive_process* - The GPU is used exclusively by one process regardless of process thread. If present, `reseterr` resets the ECC memory bit error counters. This only resets the volatile error counts, or errors since the last reboot. The permanent error counts are not affected. Moab passes the `mode` and `reseterr` portion of the request to TORQUE for processing. Moab does not support requesting GPUs as a GRES. Submitting `msub -l gres=gpus:x` does not work.
Description	Moab schedules GPUs as a special type of node-locked generic resources. When TORQUE reports GPUs to Moab, Moab can schedule jobs and correctly assign GPUs to ensure that jobs are scheduled efficiently. To have Moab schedule GPUs, configure them in TORQUE then submit jobs using the "GPU" attribute. Moab automatically parses the "GPU" attribute and assigns them in the correct manner. For information about GPU metrics, see GPGPUMetrics.

GPUs	
Examples	`> msub -l nodes=2:ppn=2:gpus=1:exclusive_process:reseterr` *Submits a job that requests 2 tasks, 2 processors and 1 GPU per task (2 GPUs total). Each GPU runs only threads related to the task and resets the volatile ECC memory big error counts at job start time.* `> msub -l nodes=4:gpus=1,tpn=2` *Submits a job that requests 4 tasks, 1 GPU per node (4 GPUs total), and 2 tasks per node. Each GPU is dedicated exclusively to one task process and the ECC memory bit error counters are not reset.* `> msub -l nodes=4:gpus=1:reseterr` *Submits a job that requests 4 tasks, 1 processor and 1 GPU per task (4 GPUs total). Each GPU is dedicated exclusively to one task process and resets the volatile ECC memory bit error counts at job start time.* `> msub -l nodes=4:gpus=2+1:ppn=2,walltime=600` *Submits a job that requests two different types of tasks, the first is 4 tasks, each with 1 processor and 2 gpus, and the second is 1 task with 2 processors. Each GPU is dedicated exclusively to one task process and the ECC memory bit error counters are not reset.*

GRES and SOFTWARE	
Format	Percent sign (`%`) delimited list of generic resources where each resource is specified using the format `<RESTYPE>[{+|:}<COUNT>]`
Description	Indicates generic resources required by the job. If the generic resource is node-locked, it is a per-task count. If a *<COUNT>* is not specified, the resource count defaults to 1.
Example	`> qsub -W x=GRES:tape+2%matlab+3 testj.txt` When specifying more than one generic resource with -l, use the percent (%) character to delimit them. `> qsub -l gres=tape+2%matlab+3 testj.txt` `> qsub -l software=matlab:2 testj.txt`

HOSTLIST	
Format	Comma (,) or plus (+) delimited list of hostnames. Ranges and regular expressions are supported in msub only.
Description	Indicates an *exact set*, *superset*, or *subset* of nodes on which the job must run. Use the caret (^) or asterisk (*) characters to specify a host list as *superset* or *subset* respectively. An exact set is defined without a caret or asterisk. An exact set means *all* the hosts in the specified hostlist must be selected for the job. A subset means the specified hostlist is used first to select hosts for the job. If the job requires more hosts than are in the subset hostlist, they will be obtained from elsewhere if possible. If the job does not require all of the nodes in the subset hostlist, it will use only the ones it needs. A superset means the hostlist is the *only* source of hosts that should be considered for running the job. If the job can't find the necessary resources in the superset hostlist it should *not* run. No other hosts should be considered in allocating the job.

HOSTLIST	
Examples	`> msub -l hostlist=nodeA+nodeB+nodeE` `hostlist=foo[1-5]` *This is an exact set of (foo1,foo2,...,foo5). The job must run on all these nodes.* `hostlist=foo1+foo[3-9]` *This is an exact set of (foo1,foo3,foo4,...,foo9). The job must run on all these nodes.* `hostlist=foo[1,3-9]` *This is an exact set of the same nodes as the previous example.* `hostlist=foo[1-3]+bar[72-79]` *This is an exact set of (foo1,foo2,foo3,bar72,bar73,...,bar79). The job must run on all these nodes.* `hostlist=^node[1-50]` *This is a superset of (node1,node2,...,node50). These are the only nodes that can be considered for the job. If the necessary resources for the job are not in this hostlist, the job is not run. If the job does not require all the nodes in this hostlist, it will use only the ones that it needs.* `hostlist=*node[15-25]` *This is a subset of (node15,node16,...,node25). The nodes in this hostlist are considered first for the job. If the necessary resources for the job are not in this hostlist, Moab tries to obtain the necessary resources from elsewhere. If the job does not require all the nodes in this hostlist, it will use only the ones that it needs.*

JGROUP	
Format	*<JOBGROUPID>*
Description	ID of job group to which this job belongs (different from the GID of the user running the job).
Example	`> msub -l JGROUP=bluegroup`

JOBFLAGS (aka FLAGS)	
Format	One or more of the following colon delimited job flags including ADVRES[:RSVID], NOQUEUE, NORMSTART, PREEMPTEE, PREEMPTOR, RESTARTABLE, or SUSPENDABLE (see job flag overview for a complete listing).
Description	Associates various flags with the job.
Example	`> qsub -l nodes=1,walltime=3600,jobflags=advres myjob.py`

JOBREJECTPOLICY	
Format:	One or more of *CANCEL*, *HOLD*, *IGNORE* (beta), *MAIL*, or *RETRY*
Default:	*HOLD*
Details:	Specifies the action to take when the scheduler determines that a job can never run. *CANCEL* issues a call to the resource manager to cancel the job. *HOLD* places a batch hold on the job preventing the job from being further evaluated until released by an administrator. Administrators can dynamically alter job attributes and possibly *fix* the job with mjobctl -m. With *IGNORE* (currently in beta), the scheduler will allow the job to exist within the resource manager queue but will neither process it nor report it. MAIL will send email to both the admin and the user when rejected jobs are detected. If *RETRY* is set, then Moab will allow the job to remain idle and will only attempt to start the job when the policy violation is resolved. Any combination of attributes may be specified. See QOSREJECTPOLICY. This is a per-job policy specified with msub -l. JOBREJECTPOLICY also exists as a global parameter.
Example:	`> msub -l jobrejectpolicy=cancel:mail`

MAXMEM	
Forma:	*<INTEGER>* (in megabytes)
Description	Maximum amount of memory the job may consume across all tasks before the JOBMEM action is taken.
Example	`> qsub -W x=MAXMEM:1000mb bw.cmd` *If a RESOURCELIMITPOLICY is set for per-job memory utilization, its action will be taken when this value is reached.*

MAXPROC	
Format	*<INTEGER>*
Description	Maximum CPU load the job may consume across all tasks before the JOBPROC action is taken.
Example	`> qsub -W x=MAXPROC:4 bw.cmd` *If a RESOURCELIMITPOLICY is set for per-job processor utilization, its action will be taken when this value is reached.*

MEM	
Format	*<INTEGER>*
Description	Specify the maximum amount of physical memory used by the job. If you do not specify MB or GB, Moab uses bytes if your resource manger is TORQUE and MB if your resource manager is Native.
Example	`> msub -l nodes=4:ppn=2,mem=1024mb` *The job must have 4 compute nodes with 2 processors per node. The job is limited to 1024 MB of memory.*

MICs	
Format	`msub -l nodes=<VALUE>:ppn=<VALUE>:mics=<VALUE>[:mode]` Where `mode` is one of: *exclusive* - The default setting. The MIC is used exclusively by one process thread. *exclusive_thread* - The MIC is used exclusively by one process thread. *exclusive_process* - The MIC is used exclusively by one process regardless of process thread. Moab passes the `mode` portion of the request to TORQUE for processing. Moab does not support requesting MICs as a GRES. Submitting `msub -l gres=mics:x` does not work.
Description	Moab schedules MICs as a special type of node-locked generic resources. When TORQUE reports MICs to Moab, Moab can schedule jobs and correctly assign MICs to ensure that jobs are scheduled efficiently. To have Moab schedule MICs , configure them in TORQUE then submit jobs using the "MIC" attribute. Moab automatically parses the "MIC" attribute and assigns them in the correct manner.

MICs	
Examples	`> msub -l nodes=2:ppn=2:mics=1:exclusive_process` *Submits a job that requests 2 tasks, 2 processors and 1 MIC per task (2 MICs total). Each MIC runs only threads related to the task.* `> msub -l nodes=4:mics=1,tpn=2` *Submits a job that requests 4 tasks, 1 MIC per node (4 MICs total), and 2 tasks per node. Each MIC is dedicated exclusively to one task process.* `> msub -l nodes=4:mics=1` *Submits a job that requests 4 tasks, 1 processor and 1 MIC per task (4 MICs total). Each MIC is dedicated exclusively to one task process.* `> msub -l nodes=4:mics=2+1:ppn=2,walltime=600` *Submits a job that requests two different types of tasks, the first is 4 tasks, each with 1 processor and 2 MICs , and the second is 1 task with 2 processors. Each MIC is dedicated exclusively to one task process.*

MINPREEMPTTIME	
Format	[[DD:]HH:]MM:]SS
Description	Minimum time job must run before being eligible for preemption. Can only be specified if associated QoS allows per-job preemption configuration by setting the preemptconfig flag.
Example	`> qsub -l minpreempttime=900 bw.cmd` *Job cannot be preempted until it has run for 15 minutes.*

MINPROCSPEED	
Format	*<INTEGER>*
Default	*0*

MINPROCSPEED	
Description	Minimum processor speed (in MHz) for every node that this job will run on.
Example	`> qsub -W x=MINPROCSPEED:2000 bw.cmd` *Every node that runs this job must have a processor speed of at least 2000 MHz.*

MINWCLIMIT	
Format	`[[DD:]HH:]MM:]SS`
Default	---
Description	Minimum wallclock limit job must run before being eligible for extension (See JOBEXTENDDURATION or JOBEXTENDSTARTWALLTIME.).
Example	`> qsub -l minwclimit=300,walltime=16000 bw.cmd` *Job will run for at least 300 seconds but up to 16,000 seconds if possible (without interfering with other jobs).*

MSTAGEIN	
Format	`[<SRCURL>[|<SRCRUL>...]%]<DSTURL>`

MSTAGEIN	
Description	Indicates a job has data staging requirements. The source URL(s) listed will be transferred to the execution system for use by the job. If more than one source URL is specified, the destination URL must be a directory. The format of *<SRCURL>* is: `[PROTO://][HOST][:PORT]][/PATH]` where the path is local. The format of *<DSTURL>* is: `[PROTO://][HOST][:PORT]][/PATH]` where the path is remote. PROTO can be any of the following protocols: ssh, file, or gsiftp. HOST is the name of the host where the file resides. PATH is the path of the source or destination file. The destination path may be a directory when sending a single file and must be a directory when sending multiple files. If a directory is specified, it must end with a forward slash (/). Valid variables include: $JOBID $HOME - Path the script was run from $RHOME - Home dir of the user on the remote system $SUBMITHOST $DEST - This is the Moab where the job will run $LOCALDATASTAGEHEAD If no destination is given, the protocol and file name will be set to the same as the source. The $RHOME (remote home directory) variable is for when a user's home directory on the compute node is different than on the submission host.
Example:	`> msub -Wx='mstagein=file://$HOME/helperscript.sh|file:///home/dev/datafile.txt%ssh://host/home/dev/' script.sh` Copy helperscript.sh and datafile.txt from the local machine to /home/dev/ on host for use in execution of script.sh. $HOME is a path containing a preceding / (i.e. /home/adaptive)

MSTAGEOUT	
Format	`[`*<SRCURL>*`[|`*<SRCRUL>*`...]%]`*<DSTURL>*

MSTAGEOUT	
Description	Indicates whether a job has data staging requirements. The source URL(s) listed will be transferred from the execution system after the completion of the job. If more than one source URL is specified, the destination URL must be a directory. The format of *<SRCURL>* is: `[PROTO://][HOST][:PORT]][/PATH]` where the path is remote. The format of *<DSTURL>* is: `[PROTO://][HOST][:PORT]][/PATH]` where the path is local. PROTO can be any of the following protocols: ssh, file, or gsiftp. HOST is the name of the host where the file resides. PATH is the path of the source or destination file. The destination path may be a directory when sending a single file and must be a directory when sending multiple files. If a directory is specified, it must end with a forward slash (/). Valid variables include: $JOBID $HOME - Path the script was run from $RHOME - Home dir of the user on the remote system $SUBMITHOST $DEST - This is the Moab where the job will run $LOCALDATASTAGEHEAD If no destination is given, the protocol and file name will be set to the same as the source. The $RHOME (remote home directory) variable is for when a user's home directory on the compute node is different than on the submission host.
Example	`> msub -W x='mstageout=ssh://$DEST/$HOME/resultfile1.txt|ssh://host/home/dev/resultscript.sh%file:///home/dev/' script.sh` *Copy `resultfile1.txt` and `resultscript.sh` from the execution system to `/home/dev/` after the execution of script.sh is complete. $HOME is a path containing a preceding / (i.e. `/home/adaptive`).*

NACCESSPOLICY	
Format	One of SHARED, SINGLEJOB, SINGLETASK, SINGLEUSER, or UNIQUEUSER

NACCESSPOLICY

Description	Specifies how node resources should be accessed. (See Node Access Policies for more information). The **naccesspolicy** option can only be used to make node access more constraining than is specified by the system, partition, or node policies. If the effective node access policy is *shared*, **naccesspolicy** can be set to *singleuser*, if the effective node access policy is *singlejob*, **naccesspolicy** can be set to *singletask*.
Example	`> qsub -l naccesspolicy=singleuser bw.cmd` `> bsub -ext naccesspolicy=singleuser lancer.cmd` *Job can only allocate free nodes or nodes running jobs by same user.*

NALLOCPOLICY

Format	One of the valid settings for the parameter NODEALLOCATIONPOLICY
Description	Specifies how node resources should be selected and allocated to the job. (See Node Allocation Policies for more information.)
Example	`> qsub -l nallocpolicy=minresource bw.cmd` *Job should use the minresource node allocation policy.*

NCPUS

Format	*<INTEGER>*
Description	The number of processors in one task where a task cannot span nodes. If **NCPUS** is used, then the resource manager's SUBMITPOLICY should be set to *NODECENTRIC* to get correct behavior. `-l ncpus=<#>` is equivalent to `-l nodes=1:ppn=<#>`when **JOBNODEMATCHPOLICY** is set to *EXACTNODE*. **NCPUS** is used when submitting jobs to an SMP. When using GPUs to submit to an SMP, use `-l ncpus=<#>:GPUs=<#>`. You cannot request both ncpus and nodes in the same job.

NMATCHPOLICY	
Format	One of the valid settings for the parameter JOBNODEMATCHPOLICY
Description	Specifies how node resources should be selected and allocated to the job.
Example	`> qsub -l nodes=2 -W x=nmatchpolicy:exactnode bw.cmd` *Job should use the* `EXACTNODE`***JOBNODEMATCHPOLICY****.*

NODESET	
Format	`<SETTYPE>:<SETATTR>[:<SETLIST>]`
Description	Specifies nodeset constraints for job resource allocation (See the NodeSet Overview for more information.).
Example	`> qsub -l nodeset=ONEOF:FEATURE:fastos:hiprio:bigmem bw.cmd`

NODESETCOUNT	
Format	`<INTEGER>`
Description	Specifies how many node sets a job uses.
Example	`> msub -l nodesetcount=2`

NODESETDELAY	
Format	`[[[DD:]HH:]MM:]SS`
Description	Causes Moab to attempt to span a job evenly across nodesets unless doing so delays the job beyond the requested **NODESETDELAY**.
Example	`> qsub -l nodesetdelay=300,walltime=16000 bw.cmd`

NODESETISOPTIONAL	
Format	*<BOOLEAN>*
Description	Specifies whether the nodeset constraint is optional (See the NodeSet Overview for more information.). Requires SCHEDCFG[] FLAGS=allowperjobnodesetisoptional.
Example	`> msub -l nodesetisoptional=true bw.cmd`

OPSYS	
Format	*<OperatingSystem>*
Description	Specifies the job's required operating system.
Example	`> qsub -l nodes=1,opsys=rh73 chem92.cmd`

PARTITION	
Format	*<STRING>*[:*<STRING>*]...
Description	Specifies the partition (or partitions) in which the job must run. The job must have access to this partition based on system wide or credential based partition access lists.
Example	`> qsub -l nodes=1,partition=math:geology` *The job must only run in the math partition or the geology partition.*

PMEM	
Format	*<INTEGER>*
Description	Specifies the maximum amount of physical memory used by any single process of the job.

PMEM	
Example	`> msub -l nodes=4:ppn=2,pmem=1024mb` *The job must have 4 compute nodes with 2 processors per node, and each process of the job is limited to 1024 MB of physical memory.*

PREF	
Format	`[{feature|variable}:]<STRING>[:<STRING>]...` If feature or variable are not specified, then feature is assumed.
Description	Specifies which node features are preferred by the job and should be allocated if available. If preferred node criteria are specified, Moab favors the allocation of matching resources but is not bound to only consider these resources. Preferences are not honored unless the node allocation policy is set to PRIORITY and the PREF priority component is set within the node's PRIORITYF attribute.
Example	`> qsub -l nodes=1,pref=bigmem` *The job may run on any nodes but prefers to allocate nodes with the bigmem feature.*

PROCS	
Format	*<INTEGER>*
Description	Requests a specific amount of processors for the job. Instead of users trying to determine the amount of nodes they need, they can instead decide how many processors they need and Moab will automatically request the appropriate amount of nodes from the RM. This also works with feature requests, such as `procs=12[:feature1[:feature2[-]]]`. Using this resource request overrides any other processor or node related request, such as `nodes=4`.
Example	`> msub -l procs=32 myjob.pl` *Moab will request as many nodes as is necessary to meet the 32-processor requirement for the job.*

PROLOGUE	
Format	*<STRING>*
Description	Specifies a user owned prologue script which will be run after the system prologue and `prologue.user` scripts at the beginning of a job. The syntax is`prologue=<file>`. The file can be designated with an absolute or relative path. This parameter works only with TORQUE.
Example	`> msub -l prologue=prologue_script.sh job.s`

PVMEM	
Format	*<INTEGER>*
Description	Specify the maximum amount of virtual memory used by any single process in the job.
Example	`> msub -l nodes=4:ppn=2,pvmem=1024mb` *The job must have 4 compute nodes with 2 processors per node, and each process of the job is limited to 1024 MB of virtual memory.*

QoS	
Format	*<STRING>*
Description	Requests the specified QoS for the job.
Example	`> qsub -l walltime=1000,qos=highprio biojob.cmd`

QUEUEJOB	
Format	*<BOOLEAN>*
Default	*TRUE*

QUEUEJOB	
Description	Indicates whether or not the scheduler should queue the job if resources are not available to run the job immediately
Example	`> msub -l nodes=1,queuejob=false test.cmd`

REQATTR	
Format	Required node attributes with version number support: *reqattr=[<must|must not|should|should not>]:<ATTRIBUTE>*[{>=|>|<=|<|=}*<VERSION>*]
Description	Indicates required node attributes. Values may include letters, numbers, dashes, underscores, and spaces. You can choose one of four requirement types for each node attribute you request: • **must** – The node on which this job runs must include the attribute at the value specified. If no node matches this requirement, Moab will not schedule the job. • **must not** – The node on which this job runs must not include the attribute at the value specified. If no node matches this requirement, Moab will not schedule the job. • **should** – If possible, the node on which this job runs should include the attribute at the value specified. If no node matches this requirement, Moab selects a node without it. • **should not** – If possible, the node on which this job runs should not include the attribute at the value specified. If no node matches this requirement, Moab selects a node without it. If you do not specify a requirement type, Moab assumes "must." For information about using reqattr to request dynamic features, see **Configuring dynamic features in TORQUE and Moab** on page 647.
Example	`> qsub -l reqattr=matlab=7.1 testj.txt`

RESFAILPOLICY	
Format	One of *CANCEL*, *HOLD*, *IGNORE*, *NOTIFY*, or *REQUEUE*
Description	Specifies the action to take on an executing job if one or more allocated nodes fail. This setting overrides the global value specified with the NODEALLOCRESFAILUREPOLICY parameter.
Example	`> msub -l resfailpolicy=ignore` *For this particular job, ignore node failures.*

RMTYPE

Format	*<STRING>*
Description	One of the resource manager types currently available within the cluster or grid. Typically, this is one of *PBS*, *LSF*, *LL*, *SGE*, *SLURM*, *BProc*, and so forth.
Example	`> msub -l rmtype=ll` Only run job on a Loadleveler destination resource manager.

SIGNAL

Format	*<INTEGER>*[@*<OFFSET>*]
Description	Specifies the pre-termination signal to be sent to a job prior to it reaching its walltime limit or being terminated by Moab. The optional offset value specifies how long before job termination the signal should be sent. By default, the pre-termination signal is sent one minute before a job is terminated
Example	`> msub -l signal=32@120 bio45.cmd`

SPRIORITY

Format	*<INTEGER>*
Default	*0*
Description	Allows Moab administrators to set a system priority on a job (similar to setspri). This only works if the job submitter is an administrator.
Example	`> qsub -l nodes=16,spriority=100 job.cmd`

TEMPLATE

Format	*<STRING>*
Description	Specifies a job template to be used as a set template. The requested template must have `SELECTT=TRUE` (See Job Templates.).

TEMPLATE	
Example	`> msub -l walltime=1000,nodes=16,template=biojob job.cmd`

TERMTIME	
Format	<*TIMESPEC*>
Default	*0*
Description	Specifies the time at which Moab should cancel a queued or active job (See Job Deadline Support.).
Example	`> msub -l nodes=10,walltime=600,termtime=12:00_Jun/14 job.cmd`

TPN	
Format	<*INTEGER*>[+]
Default	*0*

TPN	
Description	Tasks per node allowed on allocated hosts. If the plus (+) character is specified, the tasks per node value is interpreted as a minimum tasks per node constraint; otherwise it is interpreted as an exact tasks per node constraint. **Differences between TPN and PPN**: There are two key differences between the following: (A) `qsub -l nodes=12:ppn=3` and (B) `qsub -l nodes=12,tpn=3`. The first difference is that **ppn** is interpreted as the *minimum* required tasks per node while **tpn** defaults to exact tasks per node; case (B) executes the job with exactly *3* tasks on each allocated node while case (A) executes the job with at least *3* tasks on each allocated node-`nodeA:4,nodeB:3,nodeC:5` The second major difference is that the line, `nodes=X:ppn=Y` actually requests X*Y tasks, whereas `nodes=X,tpn=Y` requests only X tasks. **TPN with TORQUE as an RM**: Moab interprets nodes loosely as procs. TORQUE interprets nodes as the number of nodes from the actual number of nodes that you have in your nodes file, not your total number of procs. This means that if TORQUE is your resource manager and you specify `msub -l nodes=16:tpn=8` but do not have 16 nodes, TORQUE will not run the job. Instead, you should specify `msub -l procs=16:tpn=8`. To resolve the problem long term, you can also set `server resources_available.nodect` to the total number of procs in your system and use `msub -l nodes=16:tpn=8` as you would in a non-TORQUE Moab environment. For more information, see **resources_available** on page 2285 in the TORQUE Administrator Guide.
Example	`> msub -l nodes=10,walltime=600,tpn=4 job.cmd`

TRIG	
Format:	*<TRIGSPEC>*
Description:	Adds trigger(s) to the job (See **Creating a trigger** on page 727 for specific syntax.). Job triggers can only be specified if allowed by the QoS flag trigger. See **Enabling job triggers** on page 734 for more information.
Example:	`> qsub -l trig=etype=start\&atype=exec\&action="/tmp/email.sh job.cmd"`

TRL (Format 1)	
Format	*<INTEGER>*[@*<INTEGER>*][:*<INTEGER>*[@*<INTEGER>*]]...

TRL (Format 1)	
Default:	*0*
Description:	Specifies alternate task requests with their optional walltimes (See Malleable Jobs.).
Example:	`> msub -l trl=2@500:4@250:8@125:16@62 job.cmd` or `> qsub -l trl=2:3:4`

TRL (Format 2)	
Format	*<INTEGER>*-*<INTEGER>*
Default	*0*
Description	Specifies a range of task requests that require the same walltime (See Malleable Jobs.).
Example	`> msub -l trl=32-64 job.cmd` For optimization purposes Moab does not perform an exhaustive search of all possible values but will at least do the beginning, the end, and 4 equally distributed choices in between.

VAR	
Format	*<ATTR>*[:*<VALUE>*]
Description	Adds a generic variable or variables to the job.
Example	`> msub -l VAR=testvar1:testvalue1` *Single variable* `> msub -l VAR=testvar1:testvalue1+testvar2:testvalue2+testvar3:testvalue3` *Multiple variables*

VC	
Format	vc=*<NAME>*
Description	Submits the job or workflow to a virtual container (VC).
Example	`vc=vc13`

VMEM	
Format:	*<INTEGER>*
Description:	Specify the maximum amount of virtual memory used by all concurrent processes in the job.
Example:	`> msub -l nodes=4:ppn=2,vmem=1024mb` *The job must have 4 compute nodes with 2 processors per node, and the job is limited to 1024 MB of virtual memory.*

Resource Manager Extension Examples

If more than one extension is required in a given job, extensions can be concatenated with a semicolon separator using the format *<ATTR>*:*<VALUE>*[;*<ATTR>*:*<VALUE>*]...

Example 3-143:

```
#@comment="HOSTLIST:node1,node2;QOS:special;SID:silverA"
```

Job must run on nodes `node1` and `node2` using the QoS `special`. The job is also associated with the system ID `silverA`allowing the silver daemon to monitor and control the job.

Example 3-144:

```
# PBS -W x=\"NODESET:ONEOF:NETWORK;DMEM:64\"
```

Job will have resources allocated subject to network based nodeset constraints. Further, each task will dedicate 64 MB of memory.

Example 3-145:

```
>  qsub -l nodes=4,walltime=1:00:00 -W x="FLAGS:ADVRES:john.1"
```

Job will be forced to run within the `john.1` reservation.

Configuring dynamic features in TORQUE and Moab

Used together, the reqattr RM extension and TORQUE **$varattr** on page 2451 parameter allow you to create jobs that request resources that may change or disappear. For example, if you wanted a job to request a certain version of Octave but different versions are configured on each node and updated at any time, you can create a script that searches for the feature and version on the nodes at a specified interval. Your Moab job can then retrieve the dynamic node attributes from the latest poll and use them for scheduling.

This functionality is available when you use the TORQUE `$varattr` parameter to configure a script that regularly retrieves updates on the nodes' feature(s) and the `reqattr` RM extension to require a feature with a certain value.

To set up a dynamic feature in TORQUE and Moab

1. Create a script that pulls the information you need. For instance, the following script pulls the version of Octave on each node and prints it.

```
#!/bin/bash
# pull the version string for octave and print it for $varattr
version_str=`octave -v | grep version`
[[ $version_str =~ ([[:digit:]].[[:digit:]].[[:digit:]]) ]]
echo "Octave: ${BASH_REMATCH[1]}"
```

2. Use the TORQUE `$varattr` parameter to configure the script. Specify both the number of seconds between each time TORQUE runs the script and the path to the script. If you set the seconds to `-1`, the script will run just once. You may include arguments if desired. In the following example, the `varattr` parameter specifies that TORQUE calls the Octave script every 30 seconds.

```
$varattr 30 /usr/local/scripts/octave.sh
```

3. Submit your job in Moab, specifying `reqattr` as a resource. In this example, the job requests a node where the octave feature has a value of 3.2.4 (that the node has Octave version 3.2.4 installed).

```
> msub -l rerqattr=octave=3.2.4 myJob.sh
```

Your job requests a node with Octave version 3.2.4. TORQUE passes the most recent (pulled within the last 30 seconds) version of Octave on each node. Moab then schedules the job on a node that currently has Octave 3.2.4.

Related topics

- Resource Manager Overview

PBS Resource Manager Extensions

Resource manager extensions within PBS are used by setting the -W flag. To enable this flag, some versions of PBS must be rebuilt. TORQUE and recent OSCAR distributions come with the flag enabled by default. Most other versions do not. The required steps are documented in what follows:

1.
```
> qterm -t quick
#shutdown PBS server
```

cd to the directory from which you executed the PBS 'configure' at install time

```
> make distclean
> ./configure <WITH OPTIONS>
```

2. Create addparam script
   (chmod +x addparam)
3.
```
> addparam x
> make
```

   Backup current $PBS_HOMEDIR directory contents

   > $PBS_HOMEDIR defaults to `/usr/spool/PBS`.

4.
```
> make install
```

   Restore old $PBS_HOMEDIR directory contents

5.
```
> pbs_server    # restart PBS server
```

A job's QOS level can then be specified using the qsub -W flag. For example, `qsub -W x=iQOS:hi -l nodes=4 ...`

```
#!/bin/sh
#script:  addparam
#usage:   addparam $Parameter  [S|L]
NewParameter=$1
ParameterType=x$2
if [ ! -d src/include ];  then
  echo "error: `basename $0` src/include doesn't exist, run configure"
1>&2
  exit 1
fi
#   run make in this directory to pull over the template files
cd src/include
if make
then
  if grep -q "\"$NewParameter\"" site_*.h 2>/dev/null;  then
    echo "parameter $NewParameter previously added"
    exit 0
  fi
fi
chmod +w site_job_attr_enum.h
echo "
  JOB_SITE_ATR_$1,
" >> site_job_attr_enum.h
chmod +w site_job_attr_def.h
if [ $ParameterType = "xS" ]
then
  echo "
    {   \"$NewParameter\",
        decode_str,
        encode_str,
        set_str,
        comp_str,
        free_str,
        NULL_FUNC,
        READ_WRITE,
        ATR_TYPE_STR,
        PARENT_TYPE_JOB
    },
  " >> site_job_attr_def.h
else
  echo "
    {   \"$NewParameter\",
        decode_l,
        encode_l,
        set_l,
        comp_l,
        free_null,
        NULL_FUNC,
        READ_WRITE,
        ATR_TYPE_LONG,
        PARENT_TYPE_JOB
    },
  " >> site_job_attr_def.h
fi
exit 0
```

Adding New Resource Manager Interfaces

Moab interfaces with numerous resource management systems. Some of these interact through a resource manager specific interface (OpenPBS/PBSPro, Loadleveler, LSF), while others interact through generalized interfaces such as SSS or Wiki (See the Wiki Overview). For most resource managers, either route is possible depending on where it is easiest to focus development effort. Use of Wiki generally requires modifications on the resource manager side while creation of a new resource manager specific Moab interface would require more changes to Moab modules.

Regardless of the interface approach selected, adding support for a new resource manager is typically a straightforward process for about 95% of all supported features. The final 5% of features usually requires a bit more effort as each resource manager has a number of distinct concepts that must be addressed.

- Resource Manager Specific Interfaces
- Wiki Interface
- SSS Interface

Resource Manager Specific Interfaces

If you require tighter integration and need additional instruction, see Managing Resources Directly with the Native Interface. If you would like consultation on support for a new resource manager type, please contact the Professional Services group at Adaptive Computing.

Wiki Interface

The Wiki interface is already defined as a resource manager type, so no modifications are required within Moab. Additionally, no resource manager specific library or header file is required. However, within the resource manager, internal job and node objects and attributes must be manipulated and placed within Wiki based interface concepts as defined in the Wiki Overview. Additionally, resource manager parameters must be created to allow a site to configure this interface appropriately.

SSS Interface

The SSS interface is an XML based generalized resource manager interface. It provides an extensible, scalable, and secure method of querying and modifying general workload and resource information.

Related topics

- Creating New Tools within the Native Resource Manager Interface

Managing Resources Directly with the Native Interface

- Native Interface Overview
- Configuring the Native Interface
    - Configuring the Resource Manager
    - Reporting Resources

Native Interface Overview

The Native interface allows a site to augment or even fully replace a resource manager for managing resources. In some situations, the full capabilities of the resource manager are not needed and a lower cost or lower overhead alternative is preferred. In other cases, the nature of the environment may make use of a resource manager impossible due to lack of support. Still, in other situations it is desirable to provide information about additional resource attributes, constraints, or state from alternate sources.

In any case, Moab provides the ability to directly query and manage resources along side of or without the use of a resource manager. This interface, called the NATIVE interface can also be used to launch, cancel, and otherwise manage jobs. This NATIVE interface offers several advantages including the following:

- No cost associated with purchasing a resource manager
- No effort required to install or configure the resource manager
- Ability to support abstract resources
- Ability to support abstract jobs
- Ability to integrate node availability information from multiple sources
- Ability to augment node configuration and utilization information provided by a resource manager

However, the NATIVE interface may also have some drawbacks.

- No support for standard job submission languages
- Limited default configured and utilized resource tracking (additional resource tracking available with additional effort)

At a high level, the native interface works by launching threaded calls to perform standard resource manager activities such as managing resources and jobs. The desired calls are configured within Moab and used whenever an action or updated information is required.

Configuring the Native Interface

Using the native interface consists of defining the interface type and location. As mentioned earlier, a single object may be fully defined by multiple interfaces simultaneously with each interface updating a particular aspect of the object.

Configuring the Resource Manager

The Native resource manager must be configured using the RMCFG parameter. To specify the native interface, the **TYPE** attribute must be set to *NATIVE*.

```
RMCFG[local] TYPE=NATIVE
RMCFG[local] CLUSTERQUERYURL=exec:///tmp/query.sh
```

Reporting Resources

To indicate the source of the resource information, the **CLUSTERQUERYURL** attribute of the RMCFG parameter should be specified. This attribute is specified as a URL where the protocols *FILE*, *EXEC* and *SQL* are allowed. If a protocol is not specified, the protocol *EXEC* is assumed.

Format	Description
EXEC	Execute the script specified by the URL path. Use the script stdout as data.
FILE	Load the file specified by the URL path. Use the file contents as data.
SQL	Load data directly from an SQL database using the FULL format described below.

Moab considers a NativeRM script to have failed if it returns with a non-zero exit code or if the CHILDSTDERRCHECK parameter is set and its appropriate conditions are met. In addition, the NativeRM script associated with a job submit URL will be considered as having failed if its standard output stream contains the text `ERROR`.

This simple example queries a file on the server for information about every node in the cluster. This differs from Moab remotely querying the status of each node individually.

```
RMCFG[local]    TYPE=NATIVE
RMCFG[local]    CLUSTERQUERYURL=file:///tmp/query.txt
```

Generating Cluster Query Data

Flat Cluster Query Data

If the *EXEC* or *FILE* protocol is specified in the **CLUSTERQUERYURL** attribute, the data should provide flat text strings indicating the state and attributes of the node. The format follows the Moab Resource Manager Language Interface Specification where attributes are delimited by white space rather than ';' (See Resource Data Format):

Describes any set of node attributes with format: `<NAME><ATTR>=<VAL> [<ATTR>=<VAL>]...`

<NAME>	Name of node
<ATTR>	Node attribute

<VAL>	Value of node attribute

```
n17 CPROC=4 AMEMORY=100980 STATE=idle
```

Interfacing to FLEXlm

Moab can interface with FLEXlm to provide scheduling based on license availability. Informing Moab of license dependencies can reduce the number of costly licenses required by your cluster by allowing Moab to intelligently schedule around license limitations.

Provided with Moab in the tools directory is a Perl script, `license.mon.flexLM.pl`. This script queries a FLEXlm license server and gathers data about available licenses. This script then formats this data for Moab to read through a native interface. This script can easily be used by any site to help facilitate FLEXlm integration--the only modification necessary to the script is setting the @FLEXlmCmd to specify the local command to query FLEXlm. To make this change, edit `license.mon.flexLM.pl` and, near the top of the file, look for the line:

```
my @FLEXlmCmd = ("SETME");
```

Set the @FLEXlmCmd to the appropriate value for your system to query a license server and license file (if applicable). If lmutil is not in the PATH variable, specify its full path. Using the lmutil -a argument will cause it to report all licenses. The -c option can be used to specify an optional license file.

To test this script, run it manually. If working correctly, it will produce output similar to the following:

```
> ./license.mon.flexLM.pl
GLOBAL UPDATETIME=1104688300 STATE=idle ARES=autoCAD:130,idl_mpeg:160
CRES=autoCAD:200,idl_mpeg:330
```

If the output looks incorrect, set the $LOGLEVEL variable inside of `license.mon.flexLM.pl`, run it again, and address the reported failure.

Once the license interface script is properly configured, the next step is to add a *license* native resource manager to Moab via the `moab.cfg` file:

```
RMCFG[FLEXlm]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm]   CLUSTERQUERYURL=exec://$TOOLSDIR/flexlm/license.mon.flexLM.pl
...
```

Once this change is made, restart Moab. The command mdiag -R can be used to verify that the resource manager is properly configured and is in the state `Active`. Detailed information regarding configured and utilized licenses can be viewed by issuing the mdiag -n. Floating licenses (non-node-locked) will be reported as belonging to the GLOBAL node.

> Due to the inherent conflict with the plus sign (+), the provided license manager script replaces occurrences of the plus sign in license names with the underscore symbol (_). This replacement requires that licenses with a plus sign in their names be requested with an underscore in place of any plus signs.

Interfacing to Multiple License Managers Simultaneously

If multiple license managers are used within a cluster, Moab can interface to each of them to obtain the needed license information. In the case of FLEXlm, this can be done by making one copy of the `license.mon.flexLM.pl` script for each license manager and configuring each copy to point to a different license manager. Then, within Moab, create one native resource manager interface for each license manager and point it to the corresponding script as in the following example:

```
RMCFG[FLEXlm1]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm1]   CLUSTERQUERYURL=exec://$TOOLSDIR/flexlm/license.mon.flexLM1.pl
RMCFG[FLEXlm2]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm2]   CLUSTERQUERYURL=exec://$TOOLSDIR/flexlm/license.mon.flexLM2.pl
RMCFG[FLEXlm3]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm3]   CLUSTERQUERYURL=exec://$TOOLSDIR/flexlm/license.mon.flexLM3.pl
...
```

For an overview of license management, including job submission syntax, see License Management.

It may be necessary to increase the default limit, MMAX_GRES. See Appendix D for more implementation details.

Interfacing to Nagios

Moab can interface with Nagios to provide scheduling based on network hosts and services availability.

Nagios installation and configuration documentation can be found at Nagios.org.

Provided with Moab in the tools directory is a Perl script, node.query.nagios.pl. This script reads the Nagios `status.dat` file and gathers data about network hosts and services. This script then formats data for Moab to read through a native interface. This script can be used by any site to help facilitate Nagios integration. To customize the data that will be formatted for Moab, make the changes in this script.

You may need to customize the associated configuration file in the etc directory, `config.nagios.pl`. The statusFile line in this script tells Moab where the Nagios `status.dat` file is located. Make sure that the path name specified is correct for your site. Note that the interval which Nagios updates the Nagios status.dat file is specified in the Nagios `nagios.cfg` file. Refer to Nagios documentation for further details.

To make these changes, familiarize yourself with the format of the Nagios `status.dat` file and make the appropriate additions to the script to include the desired Moab RM language (formerly WIKI) Interface attributes in the Moab output.

To test this script, run it manually. If working correctly, it will produce output similar to the following:

```
> ./node.query.nagios.pl
gateway STATE=Running
localhost STATE=Running CPULOAD=1.22 ADISK=75332
```

Once the Nagios interface script is properly configured, the next step is to add a Nagios native resource manager to Moab via the `moab.cfg` file:

```
RMCFG[nagios] TYPE=NATIVE
RMCFG[nagios] CLUSTERQUERYURL=exec://$TOOLSDIR/node.query.nagios.pl
...
```

Once this change is made, restart Moab. The command mdiag -R can be used to verify that the resource manager is properly configured and is in the state `Active`. Detailed information regarding configured Nagios node information can be viewed by issuing the mdiag -n -v.

```
> mdiag -n -v
compute node summary
Name                   State    Procs      Memory         Disk          Swap
Speed    Opsys    Arch Par    Load Rsv Classes                        Network
           Features
gateway               Running    0:0         0:0           0:0           0:0
1.00       -      - dav   0.00   0 -                              -
          -
  WARNING:  node 'gateway' is busy/running but not assigned to an active job
  WARNING:  node 'gateway' has no configured processors
localhost             Running    0:0         0:0       75343:75347       0:0
1.00       -      - dav   0.48   0 -                              -
          -
  WARNING:  node 'localhost' is busy/running but not assigned to an active job
  WARNING:  node 'localhost' has no configured processors
-----                     ---    3:8      1956:1956    75345:75349    5309:6273
Total Nodes: 2  (Active: 2  Idle: 0  Down: 0)
```

Interfacing to Supermon

Moab can integrate with Supermon to gather additional information regarding the nodes in a cluster. A Perl script is provided in the tools directory that allows Moab to connect to the Supermon server. By default the Perl script assumes that Supermon has been started on port 2709 on localhost. These defaults can be modified by editing the respective parameter in `config.supermon.pl` in the `etc` directory. An example setup is shown below.

```
RMCFG[TORQUE]  TYPE=pbs
RMCFG[supermon] TYPE=NATIVE CLUSTERQUERYURL=exec://$HOME/tools/node.query.supermon.pl
```

To confirm that Supermon is properly connected to Moab, issue mdiag -R -v. The output should be similar to the following example, specifically there are no errors about the **CLUSTERQUERYURL**.

```
diagnosing resource managers
RM[TORQUE]  State: Active
  Type:               PBS  ResourceType: COMPUTE
  Server:             keche
  Version:            '2.2.0-snap.200707181818'
  Job Submit URL:     exec:///usr/local/bin/qsub
  Objects Reported:   Nodes=3 (6 procs)  Jobs=0
  Flags:              executionServer
  Partition:          TORQUE
  Event Management:   EPORT=15004  (no events received)
  Note:  SSS protocol enabled
  Submit Command:     /usr/local/bin/qsub
  DefaultClass:       batch
  RM Performance:     AvgTime=0.26s  MaxTime=1.04s  (4 samples)
  RM Languages:       PBS
  RM Sub-Languages:   -
RM[supermon]  State: Active
  Type:               NATIVE  ResourceType: COMPUTE
  Cluster Query URL:  exec://$HOME/node.query.supermon.pl
  Objects Reported:   Nodes=3 (0 procs)  Jobs=0
  Partition:          supermon
  Event Management:   (event interface disabled)
  RM Performance:     AvgTime=0.03s  MaxTime=0.11s  (4 samples)
  RM Languages:       NATIVE
  RM Sub-Languages:   -

Note:  use 'mrmctl -f messages ' to clear stats/failures
```

Run the Perl script by itself. The script's results should look similar to this:

```
vm01 GMETRIC[CPULOAD]=0.571428571428571 GMETRIC[NETIN]=133 GMETRIC[NETOUT]=702 GMETRIC
[NETUSAGE]=835
vm02 GMETRIC[CPULOAD]=0.428571428571429 GMETRIC[NETIN]=133 GMETRIC[NETOUT]=687 GMETRIC
[NETUSAGE]=820
keche GMETRIC[CPULOAD]=31 GMETRIC[NETIN]=5353 GMETRIC[NETOUT]=4937 GMETRIC[NETUSAGE]
=10290
```

If the preceding functioned properly, issue a checknode command on one of the nodes that Supermon is gathering statistics for. The output should look similar to below.

```
node keche
State:      Idle  (in current state for 00:32:43)
Configured Resources: PROCS: 2  MEM: 1003M  SWAP: 3353M  DISK: 1M
Utilized   Resources: ---
Dedicated  Resources: ---
Generic Metrics:  CPULOAD=33.38,NETIN=11749.00,NETOUT=9507.00,NETUSAGE=21256.00
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   0.500
Network Load: 0.87 kB/s
Flags:      rmdetected
Network:    DEFAULT
Classes:    [batch 2:2][interactive 2:2]
RM[TORQUE]: TYPE=PBS
EffNodeAccessPolicy: SHARED
Total Time: 2:03:27  Up: 2:03:27 (100.00%)  Active: 00:00:00 (0.00%)
Reservations:  ---
```

Configuring Resource Types

Native Resource managers can also perform special tasks when they are given a specific resource type. These types are specified using the RESOURCETYPE attribute of the RMCFG parameter.

Type	Description
COMPUTE	Normal compute resources (no special handling)
FS	File system resource manager (see Multiple Resource Managers for an example)
LICENSE	Software license manager (see Interfacing with FLEXlm and License Management)
NETWORK	Network resource manager
PROV	Provisioning resource manager. This is the RM Moab uses to modify the OS of a node (not a VM) and to power a node on or off.

Creating New Tools to Manage the Cluster

Using the scripts found in the `$TOOLSDIR` (`$INSTDIR/tools`) directory as a template, new tools can be quickly created to monitor or manage most any resource. Each tool should be associated with a particular resource manager service and specified using one of the following resource manager URL attributes.

CLUSTERQUERYURL	
Description	Queries resource state, configuration, and utilization information for compute nodes, networks, storage systems, software licenses, and other resources. For more details, see RM configuration.
Output	Node status and configuration for one or more nodes. See Resource Data Format.
Example	`RMCFG[v-stor] CLUSTERQUERYURL=exec://$HOME/storquery.pl` *Moab will execute the `storquery.pl` script to obtain information about 'v-stor' resources.*

JOBCANCELURL	
Description	Cancels a job.
Input	*<JOBID>*

JOBCANCELURL

Example	`RMCFG[v-stor] JOBCANCELURL=exec://$HOME/cancel.pl` *Moab will execute the* `cancel.pl` *script to cancel jobs.*

JOBMODIFYURL

Description	Modifies a job or application. For more details, see RM configuration.
Input	`[-j` *<JOBEXPR>*`] [--s[et]|--c[lear]|--i[ncrement]|--d[ecrement]]` *<ATTR>* `[=`*<VALUE>*`] [`*<ATTR>*`[=`*<VALUE>*`]]...`
Example	`RMCFG[v-stor] JOBMODIFYURL=exec://$HOME/jobmodify.pl` *Moab will execute the* `jobmodify.pl` *script to modify the specified job.*

JOBREQUEUEURL

Description	Requeues a job.
Input	*<JOBID>*
Example	`RMCFG[v-stor] JOBREQUEUEURL=exec://$HOME/requeue.pl` *Moab will execute the* `requeue.pl` *script to requeue jobs.*

JOBRESUMEURL

Description	Resumes a suspended job or application.
Input	*<JOBID>*

JOBRESUMEURL

Example	`RMCFG[v-stor] JOBRESUMEURL=exec://$HOME/jobresume.pl` *Moab will execute the* `jobresume.pl` *script to resume suspended jobs.*

JOBSTARTURL

Description	Launches a job or application on a specified set of resources.
Input	`<JOBID><TASKLIST><USERNAME> [ARCH=<ARCH>] [OS=<OPSYS>] [IDATA=<STAGEINFILEPATH>[,<STAGEINFILEPATH>]...] [EXEC=<EXECUTABLEPATH>]`
Example	`RMCFG[v-stor] JOBSTARTURL=exec://$HOME/jobstart.pl` *Moab will execute the* `jobstart.pl` *script to execute jobs.*

JOBSUBMITURL

Description	Submits a job to the resource manager, but it does not execute the job. The job executes when the **JOBSTARTURL** is called.
Input	`[ACCOUNT=<ACCOUNT>] [ERROR=<ERROR>] [GATTR=<GATTR>] [GNAME=<GNAME>] [GRES=<GRES>:<Value>[,<GRES>:<Value>]*] [HOSTLIST=<HOSTLIST>] [INPUT=<INPUT>] [IWD=<IWD>] [NAME=<NAME>] [OUTPUT=<OUTPUT>] [RCLASS=<RCLASS>] [REQUEST=<REQUEST>] [RFEATURES=<RFEATURES>] [RMFLAGS=<RMFLAGS>] [SHELL=<SHELL>] [TASKLIST=<TASKLIST>] [TASKS=<TASKS>] [TEMPLATE=<TEMPLATE>] [UNAME=<UNAME>] [VARIABLE=<VARIABLE>] [WCLIMIT=<WCLIMIT>] [ARGS=<Value>[<Value>]*]` ARGS must be the last submitted attribute because there can be multiple space-separated values for ARGS.
Example	`RMCFG[v-stor] JOBSUBMITURL=exec://$HOME/jobsubmit.pl` *Moab submits the job to the* `jobsubmit.pl` *script for future job execution.*

JOBSUSPENDURL	
Description	Suspends in memory an active job or application.
Input	*<JOBID>*
Example	`RMCFG[v-stor] JOBSUSPENDURL=exec://$HOME/jobsuspend.pl` *Moab will execute the* `jobsuspend.pl` *script to suspend active jobs.*

NODEMODIFYURL	
Description	Provide method to dynamically modify/provision compute resources including operating system, applications, queues, node features, power states, etc.
Input	`<NODEID>[,<NODEID>] [--force] {--set <ATTR>=<VAL>|--clear <ATTR>}` ATTR is one of the node attributes listed in Resource Data Format
Example	`RMCFG[warewulf] NODEMODIFYURL=exec://$HOME/provision.pl` *Moab will reprovision compute nodes using the* `provision.pl`*script.*

NODEPOWERURL	
Description	Allows Moab to issue IPMI power commands.
Input	`<NODEID>[,<NODEID>] ON | OFF`
Example	`RMCFG[node17rm] NODEPOWERURL=exec://$TOOLSDIR/ipmi.power.pl` *Moab will issue a power command contained in the* `ipmi.power.pl`*script.*

SYSTEMMODIFYURL	
Description	Provide method to dynamically modify aspects of the compute environment which are directly associated with cluster resources. For more details, see RM configuration.

SYSTEMQUERYURL	
Description	Provide method to dynamically query aspects of the compute environment which are directly associated with cluster resources. For more details, see RM configuration.
Input	default *<ATTR>* ATTR is one of `images`
Output	*<STRING>*
Example	`RMCFG[warewulf] SYSTEMQUERYURL=exec://$HOME/checkimage.pl` *Moab will load the list of images available from warewulf using the* `checkimage.pl` *script.*

WORKLOADQUERYURL	
Description:	Provide method to dynamically query the system workload (jobs, services, etc.) of the compute environment which are associated with managed resources. Job/workload information should be reported back from the URL (script, file, web service, etc.) using the Moab RM language (formerly WIKI). For more details, see RM configuration.
Output:	*<STRING>*
Example:	`RMCFG[xt] WORKLOADQUERYURL=exec://$HOME/job.query.xt3.pl` *Moab will load job/workload information by executing the* `job.query.xt3.pl` *script.*

Related topics

- mdiag -R command (evaluate resource managers)
- License Management
- Moab Resource Manager Language Data Format
- Managing Resources with SLURM

Utilizing Multiple Resource Managers

Multi-RM Overview

In many instances a site may have certain resources controlled by different resource managers. For example, a site may use a particular resource manager for licensing software for jobs, another resource manager for managing file systems, another resource manager for job control, and another for node monitoring. Moab can be configured to communicate with each of these resource managers, gathering all their data and incorporating such into scheduling decisions. With a more distributed approach to resource handling, failures are more contained and scheduling decisions can be more intelligent.

Configuring Multiple Independent Resource Manager Partitions

Moab must know how to communicate with each resource manager. In most instances, this is simply done by configuring a query command.

Migrating Jobs between Resource Managers

With multi-resource manager support, a job may be submitted either to a local resource manager queue or to the Moab global queue. In most cases, submitting a job to a resource manager queue constrains the job to only run within the resources controlled by that resource manager. However, if the job is submitted to the Moab global queue, it can use resources of any active resource manager. This is accomplished through job translation and staging.

When Moab evaluates resource availability, it determines the cost in terms of both data and job staging. If staging a job's executable or input data requires a significant amount of time, Moab integrates data and compute resource availability to determine a job's earliest potential start time on a per resource manager basis and makes an optimal scheduling decision accordingly. If the optimal decision requires a data stage operation, Moab reserves the required compute resources, stages the data, and then starts the job when the required data and compute resources are available.

Aggregating Information into a Cohesive Node View

Using the native interface, Moab can actually perform most of these functions without the need for an external resource manager. First, configure the native resource managers:

```
RMCFG[base]    TYPE=PBS
RMCFG[network] TYPE=NATIVE
RMCFG[network] CLUSTERQUERYURL=/tmp/network.sh
RMCFG[fs]      TYPE=NATIVE
RMCFG[fs]      CLUSTERQUERYURL=/tmp/fs.sh
```

The network script can be as simple as the following:

```
> _RX=`/sbin/ifconfig eth0 | grep "RX by" | cut -d: -f2 | cut -d' ' -f1`; \
> _TX=`/sbin/ifconfig eth0 | grep "TX by" | cut -d: -f3 | cut -d' ' -f1`; \
> echo `hostname` GMETRIC[netusage]=`echo "$_RX + $_TX" | bc`;
```

The preceding script would output something like the following:

```
node01 GMETRIC[netusage]=10928374
```

Moab grabs information from each resource manager and includes its data in the final view of the node.

```
> checknode node01
node node01
State:   Running  (in current state for 00:00:20)
Configured Resources: PROCS: 2  MEM: 949M  SWAP: 2000M  disk: 1000000
Utilized   Resources: SWAP: 9M
Dedicated  Resources: PROCS: 1  disk: 1000
Opsys:      Linux-2.6.5-1.358  Arch:       linux
Speed:      1.00  CPULoad:       0.320
Location:   Partition: DEFAULT  Rack/Slot:  NA
Network Load: 464.11 b/s
Network:    DEFAULT
Features:   fast
Classes:    [batch 1:2][serial 2:2]
Total Time: 00:30:39  Up: 00:30:39 (100.00%)  Active: 00:09:57 (32.46%)
Reservations:
  Job '5452'(x1)  -00:00:20 -> 00:09:40 (00:10:00)
JobList:  5452
```

Notice that the Network Load is now being reported along with disk usage.

Example File System Utilization Tracker (per user)

The following configuration can be used to track file system usage on a per user basis:

```
.....
RMCFG[file]    TYPE=NATIVE
RMCFG[file]    RESOURCETYPE=FS
RMCFG[file]    CLUSTERQUERYURL=/tmp/fs.pl
.....
```

Assuming that */tmp/fs.pl* outputs something of the following format:

```
DEFAULT STATE=idle AFS=<fs id="user1" size="789456"></fs><fs id="user2" size="123456"></fs>
```

This will track disk usage for users *user1* and *user2* every 24 hours.

License Management

- License Management Overview
- Controlling and Monitoring License Availability
- Requesting Licenses w/in Jobs

License Management Overview

Software license management is typically enabled in one of two models: node-locked and floating. Under a node-locked license, use of a given application is constrained to certain hosts. For example, `node013` may support up to two simultaneous jobs accessing application matlab. In a floating license model, a limited number of software licenses are made available cluster wide, and these licenses may be used on any combination of compute hosts. In each case, these licenses are consumable and application access is denied once they are gone.

Moab supports both node-locked and floating license models and even allows mixing the two models simultaneously. Moab monitors license usage and only launches an application when required software license availability is guaranteed. In addition, Moab also reserves licenses in conjunction with future jobs to ensure these jobs can run at the appropriate time.

> By default, Moab supports up to 128 independent license types.

> Moab license recognition is case insensitive. This means that two licenses with the same spelling and different capitalization are still recognized as the same license. When this occurs, Moab considers the license invalid.

Controlling and Monitoring License Availability

Moab can use one of three methods to determine license availability. These methods include locally specifying consumable generic resources, obtaining consumable generic resource information from the resource manager, and interfacing directly with a license manager.

Local Consumable Resources

Both node-locked and floating licenses can be locally specified within Moab using the NODECFG parameter. In all cases, this is accomplished by associating the license with a node using the GRES (or generic resource) attribute. If floating, the total cluster-wide license count should be associated with the GLOBAL node. If node-locked, the per node license count should be associated with each compute host (or globally using the *DEFAULT* node). For example, if a site has two node-locked licenses for application *EvalA* and six floating licenses for application *EvalB*, the following configuration could be used:

```
NODECFG[node001]  GRES=EvalA:2
NODECFG[node002]  GRES=EvalA:2
NODECFG[GLOBAL]   GRES=EvalB:6
...
```

Resource Manager Based Consumable Resources

Some resource managers support the ability to define and track generic resource usage at a per node level. In such cases, support for node-locked licenses may be enabled by specifying this information within the resource manager. Moab automatically detects and schedules these resources. For example, in the case of TORQUE, this can be accomplished by adding generic resource specification lines to the MOM configuration file.

Interfacing to an External License Manager

Moab may also obtain live software license information from a running license manager. Direct interfaces to supported license managers such as FlexLM may be created using the Native Resource Manager feature. A complete example on interfacing to an external license manager is provided in the FLEXlm section of the native resource manager overview.

Interfacing to Multiple License Managers

Moab may interface to multiple external license managers simultaneously simply by defining additional native resource manager interfaces. See the FLEXlm Native Resource Manager Overview for more information.

Requesting Licenses within Jobs

Requesting use of software licenses within jobs is typically done in one of two ways. In most cases, the native resource manager job submission language provides a direct method of license specification; for example, in the case of TORQUE, OpenPBS, or PBSPro, the software argument could be specified using the format *<SOFTWARE_NAME>*[+*<LICENSE_COUNT>*] as in the following example:

```
> qsub -l nodes=2,software=blast cmdscript.txt
```

> The license count is a job total, not a per task total, and the license count value defaults to 1.

An alternative to direct specification is the use of the Moab resource manager extensions. With these extensions, licenses can be requested as generic resources, using the GRES attribute. The job in the preceding example could also be requested using the following syntax:

```
> qsub -l nodes=2 -W x=GRES:blast cmdscript.txt
```

In each case, Moab automatically determines if the software licenses are node-locked or floating and applies resource requirements accordingly.

If a job requires multiple software licenses, whether of the same or different types, a user would use the following syntax:

```
> qsub -l nodes=2 -W x=GRES:blast+2 cmdscript.txt   # two 'blast' licenses required
> qsub -l nodes=2 -W x=GRES:blast+2%bkeep+3 cmdscript.txt   # two 'blast' and three
'bkeep' licenses are required
```

Related topics

- Native Resource Manager License Configuration
- License Ownership with Advance Reservations
- Multi-Cluster License Sharing with Moab Workload Manager for Grids Interfaces

Resource Provisioning

- Resource Provisioning Overview
- Configuring Provisioning

Resource Provisioning Overview

When processing a resource request, Moab attempts to match the request to an existing available resource. However, if the scheduler determines that the resource is not available or will not be available due to load or policy for an appreciable amount of time, it can select a resource to modify to meet the needs of the current requests. This process of modifying resources to meet existing needs is called provisioning.

Currently, there are two types of provisioning supported: operating system (OS) and application. As its name suggests, OS provisioning allows the scheduler to modify the operating system of an existing compute node while application level provisioning allows the scheduler to request that a software application be made available on a given compute node. In each case, Moab evaluates the costs of making the change in terms of time and other resources consumed before making the decision. Only if the benefits outweigh the costs will the scheduler initiate the change required to support the current workload.

> Preemption (requeueing) does not work with dynamic provisioning.

Configuring Provisioning

Enabling provisioning consists of configuring an interface to a provisioning manager, specifying which nodes can take advantage of this service, and what the estimated cost and duration of each change will be. This interface can be used to contact provisioning software such as xCat or HP's Server Automation tool. Additionally, locally developed systems can be interfaced via a script or web service.

Related topics

- Native Resource Manager Overview
- Appendix O: Resource Manager Integration

Managing Networks

Network Management Overview

Network resources can be tightly integrated with the rest of a compute cluster using the Moab multi-resource manager management interface. This interface has the following capabilities:

- Dynamic per job and per partition VLAN creation and management
- Monitoring and reporting of network health and failure events
- Monitoring and reporting of network load
- Creation of subnets with guaranteed performance criteria
- Automated workload-aware configuration and router maintenance
- Intelligent network-aware scheduling algorithms

Dynamic VLAN Creation

Most sites using dynamic VLAN's operate under the following assumptions:

- Each compute node has access to two or more networks, one of which is the compute network, and another which is the administrator network.
- Each compute node may only access other compute nodes via the compute network.
- Each compute node may only communicate with the head node via the administrator network.
- Logins on the head node may not be requested from a compute node.

In this environment, organizations may choose to have VLANs automatically configured that encapsulate individual jobs. These VLAN's essentially disconnect the job from either incoming or outgoing communication with other compute nodes.

Configuring VLANs

Automated VLAN management can be enabled by setting up a network resource manager that supports dynamic VLAN configuration and a QoS to request this feature. The example configuration highlights this setup:

```
...
RMCFG[cisco] TYPE=NATIVE RESOURCETYPE=NETWORK FLAGS=VLAN
RMCFG[cisco] CLUSTERQUERYURL=exec://$TOOLSDIR/node.query.cisco.pl
RMCFG[cisco] SYSTEMMODIFYURL=exec://$TOOLSDIR/system.modify.cisco.pl
QOSCFG[netsecure] SECURITY=VLAN
```

Requesting a VLAN

VLANs can be requested on a per job basis directly using the associated resource manager extension or indirectly by requesting a QoS with a VLAN security requirement.

```
> qsub -l nodes=256,walltime=24:00:00,qos=netsecure biojob.cmd
143325.umc.com submitted
```

Network Load and Health Monitoring

Network-level load and health monitoring is enabled by supporting the cluster query action in the network resource manager and specifying the appropriate **CLUSTERQUERYURL** attribute in the associated resource manager interface. Node (virtual node) query commands (mnodectl,checknode) can be used to view this load and health information that will also be correlated with associated workload and written to persistent accounting records. Network load and health based event information can also be fed into generic events and used to drive appropriate event based triggers.

At present, load and health attributes such as fan speed, temperature, port failures, and various core switch failures can be monitored and reported. Additional failure events are monitored and reported as support is added within the network management system.

Providing Per-QoS and Per-Job Bandwidth Guarantees

Intra-job bandwidth guarantees can be requested on a per job basis using the BANDWIDTH resource manager extensions. If specified, Moab does not allow a job to start unless these criteria can be satisfied via proper resource allocation or dynamic network partitions. As needed, Moab makes future resource reservations to be able to guarantee required allocations.

Example 3-146:

```
> qsub -l nodes=24,walltime=8:00:00,bandwidth=1000 hex3chem.cmd
job 44362.qjc submitted
```

If dynamic network partitions are enabled, a **NODEMODIFYURL** attribute must be properly configured to drive the network resource manager. See Native Resource Manager Overview for details.

Enabling Workload-Aware Network Maintenance

Network-aware maintenance is enabled by supporting the modify action in the network resource manager and specifying the appropriate **NODEMODIFYURL** attribute in the associated resource manager interface. Administrator resource management commands, (mnodectl and mrmctl), will then be routed directly through the resource manager to the network management system. In addition, reservation and real-time generic event and generic metric triggers can be configured to intelligently drive these facilities for maintenance and auto-recovery purposes.

Maintenance actions can include powering on and off the switch as well as rebooting/recycling all or part of the network. Additional operations are enabled as supported by the underlying networks.

Creating a Resource Management Interface for a New Network

Many popular networks are supported using interfaces provided in the Moab `tools` directory. If a required network interface is not available, a new one can be created using the following guidelines:

General Requirements

In all cases, a network resource manager should respond to a cluster query request by reporting a single node with a node name that will not conflict with any existing compute nodes. This node should report as a minimum the **state** attribute.

Monitoring Load

Network load is reported to Moab using the generic resource bandwidth. For greatest value, both configured and used bandwidth (in megabytes per second) should be reported as in the following example:

```
force10 state=idle ares=bandwidth:5466 cres=bandwidth:10000
```

Monitoring Failures

Network warning and failure events can be reported to Moab using the **gevent** metric. If automated responses are enabled, embedded epochtime information should be included.

```
force10 state=idle gevent[checksum]='ECC failure detected on port 13'
```

Controlling Router State

Router power state can be controlled as a system modify interface is created that supports the commands on, off, and reset.

Creating VLANs

VLAN creation, management, and reporting is more advanced requiring persistent VLAN ID tracking, global pool creation, and other features. Use of existing routing interface tools as templates is highly advised. VLAN management requires use of both the cluster query interface and the system modify interface.

Per-Job Network Monitoring

It is possible to gather network usage on a per job basis using the Native Interface. When the native interface has been configured to report netin and netout Moab automatically gathers this data through the life of a job and reports total usage statistics upon job completion.

```
...
node99  netin=78658 netout=1256
...
```

This information is visible to users and administrators via command-line utilities, the web portal, and the desktop graphical interfaces.

Related topics

- Native Resource Manager Overview
- Network Utilization Statistics

Intelligent Platform Management Interface

- IPMI Overview
- Node IPMI Configuration
- Installing IPMItool
- Setting-up the BMC-Node Map File
- Configuring Moab's IPMI Tools
- Configuring Moab
- Ensuring Proper Setup

IPMI Overview

The Intelligent Platform Management Interface (IPMI) specification defines a set of common interfaces system administrators can use to monitor system health and manage the system. The IPMI interface can monitor temperature and other sensor information, query platform status and power-on/power-off compute nodes. As IPMI operates independently of the node's OS interaction with the node can happen even when powered down. Moab can use IPMI to monitor temperature information, check power status, power-up, power-down, and reboot compute nodes.

Node IPMI Configuration

IPMI must be enabled on each node in the compute cluster. This is usually done either through the node's BIOS or by using a boot CD containing IPMI utilities provided by the manufacturer. With regard to configuring IPMI on the nodes, be sure to enable IPMI-over-LAN and set a common login and password on all the nodes. Additionally, you must set a unique IP address for each node's BMC. Take note of these addresses as you will need them when reviewing the Creating the IPMI BMC-Node Map File section.

Installing IPMItool

IPMItool is an open-source tool used to retrieve sensor information from the IPMI Baseboard Management Controller (BMC) or to send remote chassis power control commands. The IPMItool developer provides Fedora Core binary packages as well as a source tarball on the IPMItool download page.

> Installing IPMItool is not guaranteed to address all power management control. Manufacturer-specific tools may be required for some hardware integrations.

Download and install IPMItool on the Moab head node and make sure the `ipmitool` binary is in the current shell PATH.

Proper IPMI setup and IPMItool configuration can be confirmed by issuing the following command on the Moab head node.

```
> ipmitool -I lan -U username -P password -H BMC IP chassis status
```

The output of this command should be similar to the following.

```
System Power         : off
 Power Overload       : false
 Power Interlock      : inactive
 Main Power Fault     : false
 Power Control Fault  : false
 Power Restore Policy : previous
 Last Power Event     :
 Chassis Intrusion    : inactive
 Front-Panel Lockout  : inactive
 Drive Fault          : false
 Cooling/Fan Fault    : false
```

Creating the IPMI BMC-Node Map File [OPTIONAL]

Since the BMC can be controlled via LAN, it is possible for the BMC to have its own unique IP address. Since this IP address is separate from the IP address of the node, a simple mapping file is required for Moab to know each node's BMC address. The file is a flat text file and should be stored in the Moab home directory. If a mapping file is needed, specify the name in the `config.ipmi.pl` configuration file in the `etc/` directory. The following is an example of the mapping file:

```
#<NodeID> <BMC IP>
node01  10.10.10.101
node02  10.10.10.102
node03  10.10.10.103
node04  10.10.10.104
node05  10.10.10.105
# NodeID = the name of the nodes returned with "mdiag -n"
# BMC IP = the IP address of the IPMI BMC network interface
```

Note that only the nodes specified in this file are queried for IPMI information. Also note that the mapping file is disabled by default and the nodes that are returned from Moab with mdiag -n are the ones that are queried for IPMI sensor data.

Configuring the Moab IPMI Tools

The `tools/` subdirectory in the install directory already contains the Perl scripts needed to interface with IPMI. The following is a list of the Perl scripts that should be in the `tools/` directory; confirm these are present and executable.

```
ipmi.mon.pl     # The daemon front-end called by Moab
ipmi.power.pl   # The power control script called by Moab
__mon.ipmi.pl   # The IPMI monitor daemon that updates and caches IPMI data from nodes
```

Next, a few configuration settings need to be adjusted in the `config.ipmi.pl` file found in the `etc` subdirectory. The IPMI-over-LAN username and password need to be set to the values that were set in the Node IPMI Configuration section. Also, the IPMI query daemon's polling interval can be modified by adjusting $pollInterval. This specifies how often the IPMI-enabled nodes are queried to retrieve sensor data.

Configuring Moab

To allow Moab to use the IPMI tools, a native resource manager is configured. To do this, the following lines must be added to `moab.cfg`:

```
...
# IPMI - Node monitor script
RMCFG[ipminative] TYPE=NATIVE CLUSTERQUERYURL=exec://$TOOLSDIR/ipmi.mon.pl
...
```

Next, the following lines can be added to allow Moab to issue IPMI power commands.

```
...
# IPMI - Power on/off/reboot script
RMCFG[ipminative] NODEPOWERURL=exec://$TOOLSDIR/ipmi.power.pl
...
```

Moab can be configured to perform actions based on sensor data. For example, Moab can shut down a compute node if its CPU temperature exceeds 100 degrees Celsius, or it can power down idle compute nodes if workload is low. Generic event thresholds are used to tell Moab to perform certain duties given certain conditions. The following example is of a way for Moab to recognize it should power off a compute node if its CPU0 temperature exceeds 100 degrees Celsius.

```
...
# IPMI - Power off compute node if its CPU0 temperature exceeds 100 degrees Celsius.
GEVENTCFG[CPU0_TEMP>100] action=off
...
```

Ensuring Proper Setup

Once the preceding steps have been taken, Moab should be started as normal. The IPMI monitoring daemon should start automatically, which can be confirmed with the following:

```
moab@headnode:~/$ ps aux | grep __mon
moab   11444  0.0  0.3   6204  3172 pts/3    S    10:54   0:00 /usr/bin/perl -w
/opt/moab/tools/_mon.ipmi.pl --start
```

After a few minutes, IPMI data should be retrieved and cached. This can be confirmed with the following command:

```
moab@headnode:~/$ cat spool/ipmicache.gm
node01 GMETRIC[CPU0_TEMP]=49
node01 GMETRIC[CPU1_TEMP]=32
node01 GMETRIC[SYS_TEMP]=31
node01 POWER=ON
```

Finally, issue the following to ensure Moab is grabbing the IPMI data. Temperature data should be present in the `Generic Metrics` row.

```
moab@headnode:~/$ checknode node01
node node01
State:      Idle  (in current state for 00:03:12)
Configured Resources: PROCS: 1  MEM: 2000M  SWAP: 3952M  DISK: 1M
Utilized   Resources: ---
Dedicated  Resources: ---
Generic Metrics:  CPU0_TEMP=42.00,CPU1_TEMP=30.00,SYS_TEMP=29.00
...
```

Resource Manager Translation

- Translation Overview
- Translation Enablement Steps

Translation Overview

Resource manager translation allows end-users to continue to use existing job command scripts and familiar job management and resource query commands. This is accomplished by emulating external commands, routing the underlying queries to Moab, and then formatting the responses in a familiar manner. Using translation, job submission clients, job query clients, job control clients, and resource query clients can be emulated making switching from one resource manager to another transparent and preserving investment in legacy scripts, tools, and experience.

Translation Enablement Steps

To enable translation, you must:

- Edit the Moab tools configuration file.
- Copy, rename, and link the emulation scripts to a shorter, easier-to-use name.

Configure Translation Tools

Located in the $MOABHOMEDIR/etc directory are tools-specific configuration files. For each resource manager that has installed translation tools, edit the Moab tools configuration file in the etc directory. For example, if enabling LSF translation, do the following:

```
> vi $MOABHOMEDIR/etc/config.moab.pl
# Set the PATH to include directories for moab client commands — mjobctl, etc.
$ENV{PATH} = "/opt/moab/bin:$ENV{PATH}";
```

Add Translation Tools

In a directory accessible to users, create links to (or copy) the emulation scripts you want your users to use. For example, the emulation script tools/bjobs.lsf.pl could be copied to bin/bjobs, or, a symbolic link could be created in bin/bjobs that points to tools/bjobs.lsf.pl.

```
> ln -s tools/bjobs.lsf.pl bin/bjobs
> ln -s tools/bhosts.lsf.pl bin/bhosts
```

Troubleshooting and System Maintenance

Internal Diagnostics/Diagnosing System Behavior and Problems

Moab provides a number of commands for diagnosing system behavior. These diagnostic commands present detailed state information about various aspects of the scheduling problem, summarize performance, and evaluate current operation reporting on any unexpected or potentially erroneous conditions found. Where possible, Moab's diagnostic commands even correct detected problems if desired.

At a high level, the diagnostic commands are organized along functionality and object based delineations. Diagnostic commands exist to help prioritize workload, evaluate fairness, and determine effectiveness of scheduling optimizations. Commands are also available to evaluate reservations reporting state information, potential reservation conflicts, and possible corruption issues. Scheduling is a complicated task. Failures and unexpected conditions can occur as a result of resource failures, job failures, or conflicting policies.

Moab's diagnostics can intelligently organize information to help isolate these failures and allow them to be resolved quickly. Another powerful use of the diagnostic commands is to address the situation in which there are no hard failures. In these cases, the jobs, compute nodes, and scheduler are all functioning properly, but the cluster is not behaving exactly as desired. Moab diagnostics can help a site determine how the current configuration is performing and how it can be changed to obtain the desired behavior.

The mdiag Command

The cornerstone of Moab's diagnostics is the mdiag command. This command provides detailed information about scheduler state and also performs a large number of internal sanity checks presenting problems it finds as warning messages.

Currently, the mdiag command provides in-depth analysis of the following objects and subsystems:

Object/Subsystem	mdiag Flag	Use
Account	-a	Shows detailed account configuration information.
Blocked	-b	Indicates why blocked (ineligible) jobs are not allowed to run.
Class	-c	Shows detailed class configuration information.
Config	-C	Shows configuration lines from `moab.cfg` and whether or not they are valid.
FairShare	-f	Shows detailed fairshare configuration information as well as current fair-share usage.
Group	-g	Shows detailed group information.
Job	-j	Shows detailed job information. Reports corrupt job attributes, unexpected states, and excessive job failures.
Frame/Rack	-m	Shows detailed frame/rack information.

Object/Subsystem	mdiag Flag	Use
Node	-n	Shows detailed node information. Reports unexpected node states and resource allocation conditions.
Priority	-p	Shows detailed job priority information including priority factor contributions to all idle jobs.
QoS	-q	Shows detailed QoS information.
Reservation	-r	Shows detailed reservation information. Reports reservation corruption and unexpected reservation conditions.
Resource Manager	-R	Shows detailed resource manager information. Reports configured and detected state, configuration, performance, and failures of all configured resource manager interfaces.
Standing Reservations	-s	Shows detailed standing reservation information. Reports reservation corruption and unexpected reservation conditions.
Scheduler	-S	Shows detailed scheduler state information. Indicates if scheduler is stopped, reports status of grid interface, and identifies and reports high-level scheduler failures.
Partition	-t	Shows detailed partition information.
User	-u	Shows detailed user information.

Other Diagnostic Commands

Beyond mdiag, the checkjob and checknode commands also provide detailed information and sanity checking on individual jobs and nodes respectively. These commands can indicate why a job cannot start, which nodes can be available, and information regarding the recent events impacting current job or nodes state.

Using Moab Logs for Troubleshooting

Moab logging is extremely useful in determining the cause of a problem. Where other systems may be cursed for not providing adequate logging to diagnose a problem, Moab may be cursed for the opposite reason. If the logging level is configured too high, huge volumes of log output may be recorded, potentially obscuring the problems in a flood of data. Intelligent searching combined with the use of the LOGLEVEL and LOGFACILITY parameters can mine out the needed information. Key information associated with various problems is generally marked with the keywords WARNING, ALERT, or ERROR. See the Logging Overview for further information.

Automating Recovery Actions after a Failure

The RECOVERYACTION parameter of SCHEDCFG can be used to control scheduler action in the case of a catastrophic internal failure. Valid actions include die, ignore, restart, and trap.

Recovery Mode	Description
die	Moab will exit and, if core files are externally enabled, create a core file for analysis (This is the default behavior.).
ignore	Moab will ignore the signal and continue processing. This may cause Moab to continue running with corrupt data which may be dangerous. Use this setting with caution.
restart	When a SIGSEGV is received, Moab will relaunch using the current checkpoint file, the original launch environment, and the original command line flags. The receipt of the signal will be logged but Moab will continue scheduling. Because the scheduler is restarted with a new memory image, no corrupt scheduler data should exist. One caution with this mode is that it may mask underlying system failures by allowing Moab to overcome them. If used, the event log should be checked occasionally to determine if failures are being detected.
trap	When a SIGSEGV is received, Moab stays alive but enters diagnostic mode. In this mode, Moab stops scheduling but responds to client requests allowing analysis of the failure to occur using internal diagnostics available via the mdiag command.

Related topics

- Troubleshooting Individual Jobs

Logging Overview

The Moab Workload Manager provides the ability to produce detailed logging of all of its activities. This is accomplished using verbose server logging, event logging, and system logging facilities.

Log Facility Configuration

The LOGFILE and/or LOGDIR parameters within the `moab.cfg` file specify the destination of this logging information. Logging information will be written in the file *<MOABHOMEDIR>/<LOGDIR><LOGFILE>* unless *<LOGDIR>* or *<LOGFILE>* is specified using an absolute path. If the log file is not specified or points to an invalid file, all logging information is directed to STDERR. However, because of the sheer volume of information that can be logged, it is not recommended that this be done while in production. By default, **LOGDIR** and **LOGFILE** are set to *log* and *moab.log* respectively, resulting in scheduler logs being written to *<MOABHOMEDIR>*`/log/moab.log`.

The parameter LOGFILEMAXSIZE determines how large the log file is allowed to become before it is rolled and is set to 10 MB by default. When the log file reaches this specified size, the log file is rolled. The parameter LOGFILEROLLDEPTH controls the number of old logs maintained and defaults to 3. Rolled log files have a numeric suffix appended indicating their order.

The parameter LOGLEVEL controls the verbosity of the information. **LOGLEVEL** values between 1 and 6 are used to control the amount of information logged with 1 being the least verbose (recording only the worst events that occur) while 6 is the most verbose. The amount of information provided at each log level is approximately an order of magnitude greater than what is provided at the log level immediately below it. The first three log levels (1-3) measure the severity of an event and the rest of the levels (4-6) measure verbosity and how much detail is logged.

If a problem is detected, you may want to increase the **LOGLEVEL** value to get more details. However, doing so will cause the logs to roll faster and will also cause a lot of possibly unrelated information to clutter up the logs. Also be aware of the fact that high **LOGLEVEL** values results in large volumes of possibly unnecessary file I/O to occur on the scheduling machine. Consequently, it is not recommended that high **LOGLEVEL** values be used unless tracking a problem or similar circumstances warrant the I/O cost.

> If high log levels are desired for an extended period of time and your Moab home directory is located on a network file system, performance may be improved by moving your log directory to a local file system using the **LOGDIR** parameter.

Visibility	LOGLEVEL value	Description
FATAL	n/a	FATAL events are errors that render part of the system unusable. An example would be failing to create a connection to a database. FATAL event logging cannot be suppressed.
ERROR	1	This is the minimum level of logging that Moab accepts. ERROR events are problems that occur in circumstances where a user's goal has failed. For example, when a user submits a job but the job fails to start, the cause of the failure will be an error. Not all failures are ERROR events, such as failing to open a file because it does not exist. Like FATAL events, ERROR events cannot be suppressed.

Visibility	LOGLEVEL value	Description
WARNING	2	WARNING events are problems that have user consequences that Moab cannot easily evaluate. Their impact has to be judged by users. An example would be if a user job asked Moab to copy a folder and Moab was unable to copy one file in the folder because the file was a temp file and was opened exclusively by another process. The user might consider that failure irrelevant. WARNING event logging can be suppressed at user discretion.
INFO	3	INFO events are occurrences that might be interesting but do not represent problems. An example would be the transition to a "terminated phase" when a service is successfully terminated. This event is potentially interesting to both human and automated observers but is not a problem in any sense.
TRACE1	4	These log levels are generally not used in production environments. They are used mainly by Adaptive Computing developers to analyze various issues. Setting LOGLEVEL to one of these levels could seriously impact performance due to Moab attempting to write to the log potentially hundreds of times per second.
TRACE2	5	
TRACE3	6	

A final log related parameter is **LOGFACILITY**. This parameter can be used to focus logging on a subset of scheduler activities. This parameter is specified as a list of one or more scheduling facilities as listed in the parameters documentation.

Example 3-147:

```
# moab.cfg
# allow up to 30 100MB logfiles
LOGLEVEL          3
LOGDIR            /var/tmp/moab
LOGFILEMAXSIZE    100000000
LOGFILEROLLDEPTH 30
```

Standard Log Format

Each log event line follows a standard, tab-delimited log format:

timestamp <tab> **thread ID** <tab> **visibility** <tab> **origin** <tab> **event code** <tab> **scope IDs** <tab> **message**

- **timestamp**: Timestamps are given in local time, in ISO 8601 format, with a 4-digit timezone offset suffix. For example, `2012-01-27T15:18:30.000-0700`.
- **thread ID**: The ID of the thread that is producing the log output.
- **visibility**: Visibility is either a severity (FATAL, ERROR, WARNING, INFO) or a trace level (TRACE1, TRACE2, TRACE3).
- **origin**: Origin is where the log event came from.

- **event code**: The event code provides a way to determine what kind of event happened. For a full list of event codes, see **Event Dictionary** on page 1076.
- **scope IDs**: The scope ID associates the event with a specific job or service.
- **message**: Messages can give details about the event and possibly some action information to resolve issues.

Example 3-148:

```
2014-08-15T05:26:18.108-0600    846     TRACE1  MQueue.c:MQueueCheckStatus:3081     0
       MQueueCheckStatus()
2014-08-15T05:26:18.108-0600   846     TRACE1  MNode.c:MNodeCheckStatus:949        0
MNodeCheckStatus()
2014-08-15T05:26:18.108-0600   846     TRACE1  MVC.c:MVCHarvestVCs:2911            0           Check
VCs to harvest
2014-08-15T05:26:18.108-0600   846     TRACE1  MSU.c:MUClearChild:5301             0           MUCle
(PID)
2014-08-15T05:26:18.108-0600   846     INFO    MSysMainLoop.c:MSysMainLoop:1059    0x1002a14
Scheduling complete. Sleeping for 60 seconds.
2014-08-15T05:26:18.108-0600   846     TRACE1  MSchedStats.c:MSchedUpdateStats:36  0
MSchedUpdateStats()
2014-08-15T05:26:18.108-0600   846     INFO    MSchedStats.c:MSchedUpdateStats:45  0x100a9da
Iteration: 23; scheduling time: 0.00 seconds.
2014-08-15T05:26:18.108-0600   846     TRACE1  MRsv.c:MRsvUpdateStats:605          0
MRsvUpdateStats()
2014-08-15T05:26:18.108-0600   846     TRACE1  MSchedStats.c:MSchedUpdateStats:164 0           curre
util[23]:  0/1d (0.002f%)  PH: 0.072f%  active jobs: 0 of 0 (completed: 6217)
2014-08-15T05:26:18.109-0600   846     INFO    MSysMainLoop.c:MSysMainLoop:1138    0x1000193
scheduler:Moab  A scheduler iteration is ending.
```

Searching Moab Logs

While major failures are reported via the mdiag -S command, these failures can also be uncovered by searching the logs using the grep command as in the following:

```
> grep -E "WARNING|ALERT|ERROR" moab.log
```

On a production system working normally, this list usually includes some ALERT and WARNING messages. The messages are usually self-explanatory, but if not, viewing the log can give context to the message.

If a problem is occurring early when starting the Moab scheduler (before the configuration file is read) Moab can be started up using the `-L` *<LOGLEVEL>*flag. If this is the first flag on the command line, then the **LOGLEVEL** is set to the specified level immediately before any setup processing is done and additional logging is recorded.

If problems are detected in the use of one of the client commands, the client command can be re-issued with the `--loglevel=`*<LOGLEVEL>* command line argument specified. This argument causes log information to be written to STDERR as the client command is running. As with the server, <LOGLEVEL> values from 0 to 9 are supported.

The **LOGLEVEL** can be changed dynamically by use of the mschedctl -m command, or by modifying the `moab.cfg` file and restarting the scheduler. Also, if the scheduler appears to be hung or is not properly responding, the log level can be incremented by one by sending a *SIGUSR1* signal to the scheduler process. Repeated *SIGUSR1*signals continue to increase the log level. The *SIGUSR2* signal can be used to decrease the log level by one.

If an unexpected problem does occur, save the log file as it is often very helpful in isolating and correcting the problem.

Event Logs

Major events are reported to both the Moab log file as well as the Moab event log. By default, the event log is maintained in the statistics directory and rolls on a daily basis, using the naming convention `events.WWW_MMM_DD_YYYY` as in `events.Tue_Mar_18_2008`.

Event Log Format

The event log contains information about major job, reservation, node, and scheduler events and failures and reports this information in the following format:

```
<EVENTTIME> <EPOCHTIME>:<EID> <OBJECT> <OBJECTID> <EVENT> <DETAILS>
```

Example 3-149:

```
VERSION 500
07:03:21 110244322:0 sched clusterA   start
07:03:26 110244327:1 rsv   system.1   start   1124142432 1324142432 2 2 0.0 2342155.3
node1|node2 NA RSV=%=system.1=
07:03:54 110244355:2 job   1413       end     8 16 llw mcc 432000 Completed [batch:1]
11 08708752 1108703981 ...
07:04:59 110244410:3 rm    base       failure cannot connect to RM
07:05:20 110244431:4 sched clusterA   stop    admin
...
```

The parameter RECORDEVENTLIST can be used to control which events are reported to the event log. See the sections on job and reservation trace format for more information regarding the values reported in the details section for those records.

Record Type Specific Details Format

The format for each record type is unique and is described in the following table:

Record Type	Event Types	Description
gevent	See Enabling Generic Events for gevent information.	Generic events are included within node records. See node detail format that follows.

Record Type	Event Types	Description
job	*JOBCANCEL*, *JOBCHECKPOINT*, *JOBEND*, *JOBHOLD*, *JOBMIGRATE*, *JOBMODIFY*, *JOBPREEMPT*, *JOBREJECT*, *JOBRESUME*, *JOBSTART*, *JOBSUBMIT*	See Workload Accounting Records.
node	*NODEDOWN*, *NODEFAILURE*, *NODEUP*	The following fields are displayed in the event file in a space-delimited line as long as Moab has information pertaining to it: state, partition, disk, memory, maxprocs, swap, os, rm, nodeaccesspolicy, class, and message, where state is the node's current state and message is a human readable message indicating reason for node state change.
rm	*RMDOWN*, *RMPOLLEND*, *RMPOLLSTART*, *RMUP*	Human readable message indicating reason for resource manager state change. For *SCHEDCOMMAND*, only create/modify commands are recorded. No record is created for general list/query commands. *ALLSCHEDCOMMAND* does the same thing as *SCHEDCOMMAND*, but it also logs info query commands.
trigger	*TRIGEND*, *TRIGFAILURE*, *TRIGSTART*	`<ATTR>="<VALUE>"[ <ATTR>="<VALUE>"]...` where *<ATTR>* is one of the following: actiondata, actiontype, description, ebuf, eventtime, eventtype, flags, name, objectid, objecttype, obuf, offset, period, requires, sets, threshold, timeout, and so forth. See **About object triggers** on page 724 for more information.
vm	*VMCREATE*, *VMDESTROY*, *VMMIGRATE*, *VMPOWEROFF*, *VMPOWERON*	The following fields are displayed in the event file in a space-delimited line as long as Moab has information pertaining to it: name, sovereign, powerstate, parentnode, swap, memory, disk, maxprocs, opsys, class, and variables, where class and variables may have 0 or multiple entries.

Exporting Events in Real-Time

Moab event information can be exported to external systems in real-time using the ACCOUNTINGINTERFACEURL parameter. When set, Moab activates this URL each time one of the default events or one of the events specified by the RECORDEVENTLIST occurs.

While various protocols can be used, the most common protocol is `exec`, which indicates that Moab should launch the specified tool or script and pass in event information as command line arguments. This tool can then select those events and fields of interest and re-direct them as appropriate providing significant flexibility and control to the organization.

Exec Protocol Format

When a URL with an `exec` protocol is specified, the target is launched with the event fields passed in as STDIN. These fields appear exactly as they do in the event logs with the same values and order.

> The `tools/sql` directory included with the Moab distribution contains `event.create.sql.pl`, a sample accounting interface processing script that may be used as a template.

Enabling Syslog

In addition to the log file, the Moab scheduler can report events it determines to be critical to the UNIX syslog facility via the daemon facility using priorities ranging from `INFO` to `ERROR`. (See USESYSLOG). The verbosity of this logging is not affected by the LOGLEVEL parameter. In addition to errors and critical events, user commands that affect the state of the jobs, nodes, or the scheduler may also be logged to syslog. Moab syslog messages are reported using the INFO, NOTICE, and ERR syslog priorities.

By default, messages are logged to syslog's user facility. However, using the **USESYSLOG** parameter, Moab can be configured to use any of the following:

- *user*
- *daemon*
- *local0*
- *local1*
- *local2*
- *local3*
- *local4*
- *local5*
- *local6*
- *local7*

Managing Verbosity

In very large systems, a highly verbose log may roll too quickly to be of use in tracking specific targeted behaviors. In these cases, one or more of the following approaches may be of use:

- Use the LOGFACILITY parameter to log only functions and services of interest.
- Use syslog to maintain a permanent record of critical events and failures.
- Specify higher object loglevels on jobs, nodes, and reservations of interest (such as `NODECFG [orion13] LOGLEVEL=6`).
- Increase the range of events reported to the event log using the RECORDEVENTLIST parameter.

- Review object messages for required details.
- Run Moab in monitor mode using IGNOREUSERS, IGNOREJOBS, IGNORECLASSES, or IGNORENODES.

Related topics

- RECORDEVENTLIST parameter
- USESYSLOG parameter
- Notifying Admins
- Simulation Workload Trace Overview
- mschedctl -L command

Object Messages

Object Message Overview

Messages can be associated with the scheduler, jobs, and nodes. Their primary use is a line of communication between resource managers, the scheduler, and end-users. When a node goes offline, or when a job fails to run, both the resource manager and the scheduler will post messages to the object's message buffer, giving the administrators and end-users a reason for the failure. They can also be used as a way for different administrators and users to send messages associated with the various objects. For example, an administrator can set the message `Node going down for maintenance Apr/6/08 12pm," on node node01`, which would then be visible to other administrators.

Viewing Messages

To view messages associated with a job (either from users, the resource manager, or Moab), run the checkjob command.

To view messages associated with a node (either from users, the resource manager, or Moab), run the checknode command.

To view system messages, use the mschedctl -l message command.

To view the messages associated with a credential, run the mcredctl -c command.

Creating Messages

To create a message use the mschedctl -c message `<STRING> [-o <OBJECTTYPE>:<OBJECTID>] [-w <ATTRIBUTE>=<VALUE>[-w ...]]` command.

The *<OBJECTTYPE>* can be one of the following:

- node
- job
- rsv
- user

- acct
- qos
- class
- group

The *<ATTRIBUTE>* can be one of the following:

- owner
- priority
- expiretime
- type

Valid types include:

- annotation
- other
- hold
- pendactionerror

Deleting Messages

Deleting, or removing, messages is straightforward. The commands used depend on the type of object to which the message is attached:

- Scheduler: Use the "mschedctl -d message:*<INDEX>*" command (where INDEX is the index of the message you want to delete).

- Node: Use the mnodectl*<NODE>* -d message:*<INDEX>* command.

Notifying Administrators of Failures

Enabling Administrator Email

In the case of certain events, Moab can automatically send email to administrators. To enable mail notification, the MAILPROGRAM parameter must be set to *DEFAULT* or point to the locally available mail client. With this set, policies such as JOBREJECTPOLICY will send email to administrators if set to a value of *MAIL*.

Handling Events with the Notification Routine

Moab possesses a primitive event management system through the use of the notify program. The program is called each time an event of interest occurs. Currently, most events are associated with failures of some sort but use of this facility need not be limited in this way. The NOTIFICATIONPROGRAM parameter allows a site to specify the name of the program to run. This program is most often locally developed and designed to take action based on the event that has occurred. The location of the notification program may be specified as a relative or absolute path. If a relative path is specified, Moab looks for the notification relative to the `$(INSTDIR)/tools` directory. In all cases, Moab verifies the existence of the notification program at start up and disables it if it cannot be found or is not executable.

The notification program's action may include steps such as reporting the event via email, adjusting scheduling parameters, rebooting a node, or even recycling the scheduler.

For most events, the notification program is called with command line arguments in a simple *<EVENTTYPE>*: *<MESSAGE>* format. The following event types are currently enabled:

Event Type	Format	Description
JOBCORRUPTION	*<MESSAGE>*	An active job is in an unexpected state or has one or more allocated nodes that are in unexpected states.
JOBHOLD	*<MESSAGE>*	A job hold has been placed on a job.
JOBWCVIOLATION	*<MESSAGE>*	A job has exceeded its wallclock limit.
RESERVATIONCORRUPTION	*<MESSAGE>*	Reservation corruption has been detected.
RESERVATIONCREATED	*<RSVNAME> <RSVTYPE> <NAME> <PRESENTTIME> <STARTTIME> <ENDTIME> <NODECOUNT>*	A new reservation has been created.
RESERVATIONDESTROYED	*<RSVNAME> <RSVTYPE> <PRESENTTIME> <STARTTIME> <ENDTIME> <NODECOUNT>*	A reservation has been destroyed.
RMFAILURE	*<MESSAGE>*	The interface to the resource manager has failed.

Perhaps the most valuable use of the notify program stems from the fact that additional notifications can be easily inserted into Moab to handle site specific issues. To do this, locate the proper block routine, specify the correct conditional statement, and add a call to the routine `notify(<MESSAGE>);`.

Related topics

- JOBREJECTPOLICY parameter
- MAILPROGRAM parameter
- Event Log Overview

Issues with Client Commands

- Client Overview
- Diagnosing Client Problems

Client Overview

Moab client commands are implemented as links to the executable `mclient`. When a Moab client command runs, the client executable determines the name under which it runs and behaves accordingly. At the time Moab was configured, a home directory was specified. The Moab client attempts to open the configuration file, `moab.cfg`, in the `etc/` folder of this home directory on the node where the client command executes. This means that the home directory specified at configure time must be available on all hosts where the Moab client commands are executed. This also means that a `moab.cfg` file must be available in the `etc/` folder of this home directory. When the clients open this file, they will try to load the **SCHEDCFG** parameter to determine how to contact the Moab server.

> The home directory value specified at configure time can be overridden by creating an `/etc/moab.cfg` file or by setting the MOABHOMEDIR environment variable.

Once the client has determined where the Moab server is located, it creates a message, adds an encrypted checksum, and sends the message to the server. The Moab client and Moab server must use a shared secret key for this to work. When the Moab server receives the client request and verifies the message, it processes the command and returns a reply.

Diagnosing Client Problems

The easiest way to determine where client failures are occurring is to use built-in Moab logging. On the client side, use the `--loglevel` flag. For example:

```
> showq --loglevel=9
```

This will display verbose logging information regarding the loading of the configuration file, connecting to the Moab server, sending the request, and receiving a response.

This information almost always reveals the source of the problem. If it does not, the next step is to look at the Moab server side logs; this is done using the following steps:

- Stop Moab scheduling so that the only activity is handling Moab client requests.

```
> mschedctl -s
```

- Set the logging level to *very* verbose.

```
> mschedctl -m loglevel 7
```

- Watch Moab activity.

```
> tail -f log/moab.log | more
```

Now, in a second window, issue any failing client command, such as showq.

The `moab.log` file will record the client request and any reasons it was rejected.

Tracking System Failures

System Failures

The scheduler has a number of dependencies that may cause failures if not satisfied. These dependencies are in the areas of disk space, network access, memory, and processor utilization.

Disk Space

The scheduler uses a number of files. If the file system is full or otherwise inaccessible, the following behaviors might be noted:

Unavailable File	Behavior
moab.pid	Scheduler cannot perform single instance check.
moab.ck*	Scheduler cannot store persistent record of reservations, jobs, policies, summary statistics, and so forth.
moab.cfg **/moab.dat**	Scheduler cannot load local configuration.
log/*	Scheduler cannot log activities.
stats/*	Scheduler cannot write job records.

When possible, configure Moab to use local disk space for configuration files, statistics files, and logs files. If any of these files are located in a networked file system (such as NFS, DFS, or AFS) and the network or file server experience heavy loads or failures, Moab server may appear sluggish or unresponsive and client command may fail. Use of local disk space eliminates susceptibility to this potential issue.

Network

The scheduler uses a number of socket connections to perform basic functions. Network failures may affect the following facilities.

Network Connection	Behavior
scheduler client	Scheduler client commands fail.

Network Connection	Behavior
resource manager	Scheduler is unable to load/update information regarding nodes and jobs.
allocation manager	Scheduler is unable to validate account access or reserve/debit account balances.

Memory

Depending on cluster size and configuration, the scheduler may require up to 120 MB of memory on the server host. If inadequate memory is available, multiple aspects of scheduling may be negatively affected. The scheduler log files should indicate if memory failures are detected and mark any such messages with the ERROR or ALERT keywords.

Processor Utilization

On a heavily loaded system, the scheduler may appear sluggish and unresponsive. However, no direct failures should result from this slowdown. Indirect failures may include timeouts of peer services (such as the resource manager or allocation manager) or timeouts of client commands. All timeouts should be recorded in the scheduler log files.

Internal Errors

The Moab scheduling system contains features to assist in diagnosing internal failures. If the scheduler exits unexpectedly, the scheduler logs may provide information regarding the cause. If no reason can be determined, use of a debugger may be required.

Logs

The first step in any exit failure is to check the last few lines of the scheduler log. In many cases, the scheduler may have exited due to misconfiguration or detected system failures. The last few lines of the log should indicate why the scheduler exited and what changes would be required to correct the situation. If the scheduler did not intentionally exit, increasing the LOGLEVEL parameter to *7*, or higher, may help isolate the problem.

Reporting Failures

If an internal failure is detected on your system, the information of greatest value to developers in isolating the problem will be the output of the gdb where subcommand and a printout of all variables associated with the failure. In addition, a level 7 log covering the failure can also help in determining the environment that caused the failure. If you encounter such and require assistance, please submit a ticket at the following address:

http://www.adaptivecomputing.com/services/techsupport.php

> If you do not already have a support username and password, please create a free account to request a support ticket

Problems with Individual Jobs

To determine why a particular job will not start, there are several helpful commands:

checkjob -v

checkjob evaluates the ability of a job to start immediately. Tests include resource access, node state, job constraints (such as startdate, taskspernode, and QoS). Additionally, command line flags may be specified to provide further information.

Flag	Description
-l *<POLICYLEVEL>*	Evaluates impact of throttling policies on job feasibility.
-n *<NODENAME>*	Evaluates resource access on specific node.
-r *<RESERVATION_LIST>*	Evaluates access to specified reservations.

checknode

Displays detailed status of node.

mdiag -b

Displays various reasons job is considered blocked or non-queued.

mdiag -j

Displays high level summary of job attributes and performs sanity check on job attributes/state.

showbf -v

Determines general resource availability subject to specified constraints.

Diagnostic Scripts

Moab Workload Manager provides diagnostic scripts that can help aid in monitoring the state of the scheduler, resource managers, and other important components of the cluster software stack. These scripts can also be used to help diagnose issues that may need to be resolved with the help of Adaptive Computing support staff. This section introduces available diagnostic scripts.

support-diag.py

The `support-diag.py` script has a two-fold purpose. First, it can be used by a Moab trigger or cron job to create a regular snapshot of the state of Moab. The script captures the output of several Moab diagnostic commands (such as showq, mdiag -n, and mdiag -S), gathers configuration/log files, and records pertinent operating system information. This data is then compressed in a time-stamped tarball for easy long-term storage.

Second, the script provides Adaptive Computing support personnel with a complete package of information that can be used to help diagnose configuration issues or system bugs. After capturing the state of Moab, the resulting tarball can be sent to your Adaptive Computing support contact for further diagnosis.

The script asks you for the trouble ticket number, `-t` *<TICKET#>*, or `-n`. If you chose to enter `-t` *<TICKET#>* the script uploads your support diagnostic output to Adaptive Computing Customer Support. The upload and ticket number request can be prevented using the `-n` option.

Synopsis

`support-diag.py` [*<options>*]

Arguments

Argument	Description
`-h, --help`	Show this help message and exit.
`-q, --diag-torque-off, --without-torque`	Disable TORQUE diagnostics.
`-p TMPDIR, --tmp-dir=TMPDIR`	Use a different tmp directory to store output.
`-n, --no-upload`	Do not upload to Adaptive Computing.
`-t TICKET#`	Support ticket number.
`-f, --full-mode`	Gather additional logs, stats and, `moab.db` files.
`-u TIMEOUT, --moab-timeout=TIMEOUT`	Define Moab command timeout (Default 300 seconds).
`-d, --debug-mode`	`support-diag` print debug variables.
`-o, --offline-mode`	Gather offline logging only.
`-r, --ftp`	Use ftp instead of scp.
`-V, --version`	Print version information.

support.diag.pl

This script is deprecated with the 8.0 release. Use the `support-diag.py` script instead.

The `support.diag.pl` script has a two-fold purpose. First, it can be used by a Moab trigger or cron job to create a regular snapshot of the state of Moab. The script captures the output of several Moab diagnostic commands (such as showq, mdiag -n, and mdiag -S), gathers configuration/log files, and records pertinent operating system information. This data is then compressed in a time-stamped tarball for easy long-term storage.

The second purpose of the `support.diag.pl` script is to provide Adaptive Computing support personnel with a complete package of information that can be used to help diagnose configuration issues or system bugs. After capturing the state of Moab, the resulting tarball could be sent to your Adaptive Computing support contact for further diagnosis.

The `support.diag.pl` will ask you for the trouble ticket number then guide you through the process of uploading the data to Adaptive Computing Customer Support. The uploading and ticket number request may be prevented using the `--no-upload` and `--support-ticket=<SUPPORT_TICKET_ID>` flags detailed in the Arguments table that follows.

Synopsis

`support.diag.pl [--include-log-lines=<NUM>] [--diag-torque]`

Arguments

Argument	Description
`--include-log-lines=<NUM>`	Instead of including the entire `moab.log` file, only the last *<NUM>* lines are captured in the diagnostics.
`--diag-torque`	Diagnostic commands pertinent to the TORQUE resource manager are included.
`--no-upload`	Prevents the system from asking the user if they want to upload the tarball to Adaptive Computing Customer Support.
`--support-ticket=<SUPPORT_TICKET_ID>`	Prevents the system from asking the user for a support ticket number.

Improving User Effectiveness

- **Providing Resource Availability Information** on page 694
- **Collecting Performance Information on Individual Jobs** on page 694

User Feedback Loops

Almost invariably, real world systems outperform simulated systems, even when all policies, reservations, workload, and resource distributions are fully captured and emulated. What is it about real world usage that is not emulated via a simulation? The answer is the user feedback loop, the impact of users making decisions to optimize their level of service based on real time information.

A user feedback loop is created any time information is provided to a user that modifies job submission or job management behavior. As in a market economy, the cumulative effect of many users taking steps to improve their individual scheduling performance results in better job packing, lower queue time, and better overall system utilization. Because this behavior is beneficial to the system at large, system administrators and management should encourage this behavior and provide the best possible information to them.

There are two primary types of information that help users make improved decisions: cluster wide resource availability information and per job resource utilization information.

Improving Job Size/Duration Requests

Moab provides a number of informational commands that help users make improved job management decisions based on real-time cluster wide resource availability information. These commands include showbf, showstats -f, and showq. Using these commands, a user can determine what resources are available and what job configurations statistically receive the best scheduling performance.

Improving Resource Requirement Specification

A job's resource requirement specification tells the scheduler what type of compute nodes are required to run the job. These requirements may state that a certain amount of memory is required per node or that a node has a minimum processor speed. At many sites, users will determine the resource requirements needed to run an initial job. Then, for the next several years, they will use the same basic batch command file to run all of their remaining jobs even though the resource requirements of their subsequent jobs may be very different from their initial run. Users often do not update their batch command files even though these constraints may be unnecessarily limiting the resources available to their jobs for two reasons: (1) users do not know how much their performance will improve if better information were provided and (2) users do not know exactly what resources their jobs are using and are afraid to lower their job's resource requirements since doing so might cause their job to fail.

To help with determining accurate per job resource utilization information, Moab provides the FEEDBACKPROGRAM facility. This tool allows sites to send detailed resource utilization information back to users via email, to store it in a centralized database for report preparation, or use it in other ways to help users refine their batch jobs.

User Level Statistics

Besides displaying job queues, end-users can display a number of their own statistics. The showstats -u *<USER_ID>* command displays current and historical statistics for a user as seen in what follows:

```
$ showstats -u john
statistics initialized Wed Dec 31 17:00:00

         |------ Active ------|---------------------------------- Completed -----------
------------------------|
user      Jobs Procs ProcHours Jobs    %    PHReq    %    PHDed    %   FSTgt  AvgXF
MaxXF  AvgQH  Effic  WCAcc
john         1     1     30.96    9   0.00  300.0   0.00  148.9   0.00 -----   0.62
0.00   4.33 100.00  48.87
```

Users can query available system resources with the showbf command. This can aid users in requesting node configurations that are idle. Also, users can use the checkjob command to determine what parameter(s) are restricting their job from running. Moab performs better with more accurate wallclock estimates.

Enhancing Wallclock Limit Estimates

As explained in the previous section, showstats -u *<USER_ID>* reports statistics for a given user. The showstats -u command can be accessed by all users. They can use fields such as `PHReq`, `PHDed`, or `WCAcc` to gauge wallclock estimates. Accurate wallclock estimates allow a job to be scheduled as soon as possible in a slot that it will fit in. Low or high estimates can cause a job to be scheduled in a less favorable position.

Job Start Time Estimates

Each user can use the showstart command to display estimated start and completion times. The following example illustrates a typical response from issuing this command:

```
> showstart orion.13762
job orion.13762 requires 2 procs for 0:33:20
Estimated Rsv based start in                  1:04:55 on Fri Jul 15 12:53:40
Estimated Rsv based completion in             2:44:55 on Fri Jul 15 14:33:40
Estimated Priority based start in             5:14:55 on Fri Jul 15 17:03:40
Estimated Priority based completion in        6:54:55 on Fri Jul 15 18:43:40
Estimated Historical based start in          00:00:00 on Fri Jul 15 11:48:45
Estimated Historical based completion in      1:40:00 on Fri Jul 15 13:28:45
Best Partition: fast
```

Estimation Types

Reservation-Based Estimates

Reservation-based start time estimation incorporates information regarding current administrative, user, and job reservations to determine the earliest time the specified job can allocate the needed resources and start running. In essence, this estimate indicates the earliest time the job will start, assuming this job is the highest priority job in the queue.

> For reservation-based estimates, the information provided by this command is more highly accurate if the job is highest priority, if the job has a reservation, or if the majority of the jobs that are of higher priority have reservations. Consequently, site administrators wanting to make decisions based on this information may want to consider using the RESERVATIONDEPTH parameter to increase the number of priority-based reservations. This can be set so that most, or even all, idle jobs receive priority reservations and make the results of this command generally useful. The only caution of this approach is that increasing the **RESERVATIONDEPTH** parameter more tightly constrains the decisions of the scheduler and may result in slightly lower system utilization (typically less than 8% reduction).

Backlog/Priority Estimates

Priority-based job start analysis determines when the queried job will fit in the queue and determines the estimated amount of time required to complete the jobs currently running or scheduled to run before this job can start.

In all cases, if the job is running, this command returns the time the job starts. If the job already has a reservation, this command returns the start time of the reservation.

Historical Estimates

Historical analysis uses historical queue times for jobs that match a similar processor count and job duration profile. This information is updated on a sliding window that is configurable within `moab.cfg`.

Related topics

- ENABLESTARTESTIMATESTATS parameter
- showstart command

Providing Resource Availability Information

Moab provides commands to allow the user to query available resources. The showbf command displays what resources are available for immediate use. Using different command line parameters, such as `-m`, `-n`, and `-q` allows the user to query resources based on memory, nodecount, or QoS respectively.

Collecting Performance Information on Individual Jobs

Individual job information can be collected from the statistics file in STATDIR, which contains start time, end time, end state, QoS requested, QoS delivered, and so forth for different jobs. Also, Moab optionally provides similar information to a site's feedback program. See section 21.1 User Feedback Overview for more information about the feedback program.

Cluster Analysis and Testing

Moab has a number of unique features that allow site administrators to visualize current cluster behavior and performance, safely evaluate changes on production systems, and analyze probable future behaviors within a variety of environments.

These capabilities are enabled through a number of Moab facilities that may not appear to be closely related at first. However, taken together, these facilities allow organizations the ability to analyze their cluster without the losses associated with policy conflicts, unnecessary downtime, and faulty systems middleware.

Testing New Releases and Policies

Moab Evaluation Modes

MONITOR Mode

Moab supports a scheduling mode called *MONITOR*. In this mode, the scheduler initializes, contacts the resource manager and other peer services, and conducts scheduling cycles exactly as it would if running in *NORMAL* or production mode. Jobs are prioritized, reservations created, policies and limits enforced, and administrator and end-user commands enabled. The key difference is that although live resource management information is loaded, *MONITOR* mode disables Moab's ability to start, preempt, cancel, or otherwise modify jobs or resources. Moab continues to attempt to schedule exactly as it would in *NORMAL* mode, but its ability to actually impact the system is disabled. Using this mode, a site can quickly verify correct resource manager configuration and scheduler operation. This mode can also be used to validate new policies and constraints. In fact, Moab can be run in *MONITOR* mode on a production system while another scheduler or even another version of Moab is running on the same system. This unique ability can allow new versions and configurations to be fully tested without any exposure to potential failures and with no cluster downtime.

To run Moab in *MONITOR* mode, simply set the **MODE** attribute of the **SCHEDCFG** parameter to *MONITOR* and start Moab. Normal scheduler commands can be used to evaluate configuration and performance. Diagnostic commands can be used to look for any potential issues. Further, the Moab log file can be used to determine which jobs Moab attempted to start, and which resources Moab attempted to allocate.

If another instance of Moab is running in production and a site administrator wants to evaluate an alternate configuration or new version, this is easily done but care should be taken to avoid conflicts with the primary scheduler. Potential conflicts include statistics files, logs, checkpoint files, and user interface ports. One of the easiest ways to avoid these conflicts is to create a new test directory with its own log and statistics subdirectories. The new `moab.cfg` file can be created from scratch or based on the existing `moab.cfg` file already in use. In either case, make certain that the **PORT** attribute of the **SCHEDCFG** parameter differs from that used by the production scheduler by at least two ports. If testing with the production binary executable, the MOABHOMEDIR environment variable should be set to point to the new test directory to prevent Moab from loading the production `moab.cfg` file.

TEST Mode

TEST mode behaves much like *MONITOR* mode with the exception that Moab will log the scheduling actions it would have taken to the `stats/<DAY>.events` file. Using this file, sites can determine the actions Moab would have taken if running in *NORMAL* mode and verify all actions are in agreement with expected behavior.

INTERACTIVE Mode

INTERACTIVE mode allows for evaluation of new versions and configurations in a manner different from *MONITOR* mode. Instead of disabling all resource and job control functions, Moab sends the desired change request to the screen and requests permission to complete it. For example, before starting a job, Moab may print something like the following to the screen:

```
Command:  start job 1139.ncsa.edu on node list test013,test017,test018,test021
Accept:  (y/n) [default: n]?
```

The administrator must specifically accept each command request after verifying it correctly meets desired site policies. Moab will then execute the specified command. This mode is highly useful in validating scheduler behavior and can be used until configuration is appropriately tuned and all parties are comfortable with the scheduler's performance. In most cases, sites will want to set the scheduling mode to *NORMAL* after verifying correct behavior.

Testing New Releases

By default, Moab runs in a mode called *NORMAL*, which indicates that it is responsible for the cluster. It loads workload and resource information, and is responsible for managing that workload according to mission objectives and policies. It starts, cancels, preempts, and modifies jobs according to these policies.

If Moab is configured to use a mode called TEST, it loads all information, performs all analysis, but, instead of actually starting or modifying a job, it merely logs the fact that it would have done so. A test instance of Moab can run at the same time as a production instance of Moab. A test instance of Moab can also run while a production scheduler of another type (such as PBS, LSF, or SLURM) is simultaneously running. This multi-scheduler ability allows stability and performance tests to be conducted that can help answer the following questions:

- What impact do Moab services have on network, processor, and memory load?
- What impact do Moab services have on the underlying resource manager?
- Is Moab able to correctly import resource, workload, policy, and credential information from the underlying resource manager?
- Are Moab's logged scheduling decisions in line with mission objectives?

In test mode, all of Moab's commands and services operate normally allowing the use of client commands to perform analysis. In most cases, the mdiag command is of greatest value, displaying loaded values as well as reporting detected failures, inconsistencies, and object corruption. The following table highlights the most common diagnostics performed.

Command	Object
mdiag -n	Compute nodes, storage systems, network systems, and generic resources
mdiag -j	Applications and static jobs
mdiag -u **mdiag -g** **mdiag -a**	User, group, and account credentials
mdiag -c	Queues and policies
mdiag -R	Resource manager interface and performance
mdiag -S	Scheduler/system level failures introduced by corrupt information

These commands will not only verify proper scheduling objects but will also analyze the behavior of each resource manager, recording failures, and delivered performance. If any misconfiguration, corruption, interface failure, or internal failure is detected, it can be addressed in the test mode instance of Moab with no urgency or risk to production cluster activities.

Testing New Policies

Verifying Correct Specification of New Policies

The first aspect of verifying a new policy is verifying correct syntax and semantics. If using Moab Cluster Manager, this step is not necessary as this tool automatically verifies proper policy specification. If manually editing the `moab.cfg` file, the following command can be used for validation:

```
> mdiag -C
```

This command will validate the configuration file and report any misconfiguration.

Verifying Correct Behavior of New Policies

If concern exists over the impact of a new policy, an administrator can babysit Moab by putting it into INTERACTIVE mode. In this mode, Moab will schedule according to all mission objectives and policies, but before taking any action, it will request that the administrator confirm the action. See the interactive mode overview for more information.

In this mode, only actions approved by the administrator will be carried out. Once proper behavior is verified, the Moab mode can be set to *NORMAL*.

Moab Side-by-Side

Moab provides an additional evaluation method that allows a production cluster or other resource to be logically partitioned along resource and workload boundaries and allows different instances of Moab to schedule different partitions. The parameters IGNORENODES, IGNORECLASSES, IGNOREJOBS, and IGNOREUSERS are used to specify how the system is to be partitioned. In the following example, a small portion of an existing cluster is partitioned for temporary grid testing so that there is no impact on the production workload.

```
SCHEDCFG[prod] MODE=NORMAL SERVER=orion.cxz.com:42020
RMCFG[TORQUE]  TYPE=PBS
IGNORENODES    node61,node62,node63,node64
IGNOREUSERS    gridtest1,gridtest2
...
SCHEDCFG[prod] MODE=NORMAL SERVER=orion.cxz.com:42030
RMCFG[TORQUE]  TYPE=PBS
IGNORENODES    !node61,node62,node63,node64
IGNOREUSERS    !gridtest1,gridtest2
...
```

Two completely independent Moab servers schedule the cluster. The first server handles all jobs and nodes except for the ones involved in the test. The second server handles only test nodes and test jobs. While both servers actively talk and interact with a single TORQUE resource manager, the ***IGNORE**** *parameters cause them to not schedule, nor even see the other partition and its associated workload.*

When enabling Moab side-by-side, each Moab server should have an independent home directory to prevent logging and statistics conflicts. Also, in this environment, each Moab server should communicate with its client commands using a different port as shown in the previous example.

When specifying the **IGNORENODES** parameter, the exact node names, as returned by the resource manager, should be specified.

Related topics

- Testing New Versions and Configurations

Testing New Middleware

Moab can be used to drive new middleware stress testing resource management systems, information services, allocation services, security services, data staging services, and other aspects. Moab is unique when compared to other stress testing tools as it can perform the tests in response to actual or recorded workload traces, performing a playback of events and driving the underlying system as if it were part of the production environment.

This feature can be used to identify scalability issues, pathological use cases, and accounting irregularities in anything from LDAP, to NIS, and NFS.

Using Moab's time management facilities, Moab can drive the underlying systems in accordance with the real recorded distribution of time, at a multiplier of real time, or as fast as possible.

The following table describes some aspects of cluster analysis that can be driven by Moab.

System	Details
Allocation Manager	Use *test* or *simulation* mode to drive scheduling queries, allocation debits, and reservations to accounting packages. Verify synchronization of cluster statistics and stress test interfaces and underlying databases.
On-Demand/Provisioning Services	Use *simulation* or native resource manager mode to drive triggers and resource management interfaces to enable dynamic provisioning of hardware, operating systems, application software, and services. Test reliability and scalability of data servers, networks, and provisioning software as well as the interfaces and business logic coordinating these changes.
Resource Monitoring	Use *test* or native resource manager mode to actively load information from compute, network, storage, and software license managers confirming validity of data, availability during failures, and scalability.

With each evaluation, the following tests can be enabled:

- functionality
- reliability
    - hard failure
        - hardware failure - compute, network, and data failures
        - software failure - loss of software services (NIS, LDAP, NFS, database)
        - soft failure
        - network delays, full file system, dropped network packets
    - corrupt data
- performance
- determine peak responsiveness in seconds/request
- determine peak throughput in requests/second
- determine responsiveness under heavy load conditions

- determine throughput under external load conditions
  - large user base (many users, groups, accounts)
  - large workload (many jobs)
  - large cluster (many nodes)
- manageability
  - full accounting for all actions/events
  - actions/failures can be easily and fully diagnosed

> If using a native resource manager and you do not want to actually submit real workload, you can set the environment variable MFORCESUBMIT to allow virtual workload to be managed without ever launching a real process.

General Analysis

For all middleware interfaces, Moab provides built-in performance analysis and failure reporting. Diagnostics for these interfaces are available via the mdiag command.

Native Mode Analysis

Using native mode analysis, organizations can run Moab in *normal* mode with all facilities fully enabled, but with the resource manager fully emulated. With a native resource manager interface, any arbitrary cluster can be emulated with a simple script or flat text file. Artificial failures can be introduced, jobs can be virtually running, and artificial performance information generated and reported.

In the simplest case, emulation can be accomplished using the following configuration:

```
SCHEDCFG[natcluster] MODE=NORMAL SERVER=test1.bbli.com
ADMINCFG[1] USERS=dev
RMCFG[natcluster] TYPE=NATIVE CLUSTERQUERYURL=file://$HOME/cluster.dat
```

The preceding configuration will load cluster resource information from the file `cluster.dat`. An example resource information file follows:

```
node01 state=idle cproc=2
node02 state=idle cproc=2
node03 state=idle cproc=2
node04 state=idle cproc=2
node05 state=idle cproc=2
node06 state=idle cproc=2
node07 state=idle cproc=2
node08 state=idle cproc=2
```

In actual usage, any number of node attributes may be specified to customize these nodes, but in this example, only the node state and node configured processors attributes are specified.

The **RMCFG** flag *NORMSTART* indicates that Moab should not actually issue a job start command to an external entity to start the job, but rather start the job logically internally only.

If it is desirable to take an arbitrary action at the start of a job, end of a job, or anywhere in between, the **JOBCFG** parameter can be used to create one or more arbitrary triggers to initiate internal or external events. The triggers can do anything from executing a script, to updating a database, to using a Web service.

Using native resource manager mode, jobs may be introduced using the msub command according to any arbitrary schedule. Moab will load them, schedule them, and start them according to all site mission objectives and policies and drive all interfaced services as if running in a full production environment.

Green computing

Green computing overview

SearchDataCenter.com defines green computing as the environmentally responsible use of computers and related resources. Such practices include the implementation of energy-efficient central processing units (CPUs), servers, and peripherals as well as reduced resource consumption and proper disposal of electronic waste (e-waste).

The Moab HPC Suites, both Basic Edition and Enterprise Edition, contain power management features that give a Moab administrator the ability to implement policies that can conserve energy and save on operational costs, often without affecting an HPC system's performance with regard to job execution times.

Effective power management means managing power or energy consumption while a compute node is actively running jobs, and when a compute node is idle. Both scenarios require different tools and policies.

- Active compute node power management is mainly performed through control of the clock frequency of the processor(s) on a compute node while a job is executing. Decreasing the clock frequency can reduce energy usage.
- Idle compute node power management is mainly performed by placing a compute node into different low-power states, such as standby and suspend, or no-power states, such as hibernate and shutdown.

The table below identifies the Moab power management features and/or methods available for the different Moab HPC Suite editions.

Feature or Method	Moab HPC Suite Edition	
	Basic	Enterprise
CPU Clock Frequency Control		
• Moab Job Submission Option	X	X
• TORQUE Job Submission Option	X	X
• Moab Job Template Option	X	X
Manual Power Management		
• Moab-based `on` and `off` states	–	X
• TORQUE-based low-power and no-power states	X	X
Automated Power Management and Green Policies		
• Moab-only global-level policies and power management for `on` and `off` states	–	X
• Moab/Moab Web Services-based global, partition, and node-level policies and power management for low-power and no-power states	–	X
• Green Idle Node Pool Management Policies	–	X
Energy-Consumption-by-Job Accounting		
• Moab as the ability to report, record and charge a cost for the electrical energy consumed by a job. Currently, Moab 8.0.x supports this capability only for Cray XC systems running CLE 5.2 or later.	–	X

Moab Power Management Methods

Moab supports two separate and mutually-exclusive methods for managing the power state of compute nodes, which affects energy consumption. The first method, introduced in Moab 7.2, allows an administrator to manually power on and power off compute nodes and to create a global set of green policies that automatically perform these two functions based on specific conditions involving idle compute nodes. The second method, introduced in Moab 8.0 and TORQUE 5.0, give an administrator additional power states besides `on` and `off` and offer finer control of green policies at the global, partition, and node levels. Before delving into the theory of operation of these two separate methods, an administrator must understand how Moab views power management regardless of which method is used.

Moab View of Power Management

Moab is not aware of the actual power state of nodes. From Moab's perspective, nodes are only on or off. If Moab needs a node that is off, it issues a power-on job prior to scheduling the incoming job.

In addition, in order to schedule a job to a compute node, Moab requires the compute node's workload resource manager, which in our example is TORQUE, to report the compute node's state is `idle`. When the compute node's binary power state indicates `on` and the RM indicates the compute node's state is `idle`, Moab will schedule jobs to the compute node. Any value other than `idle` for the node's state and

Moab will not schedule a job to the node. If the power state is `off`, Moab issues a power-on job as a dependency to the regular job.

Moab performs compute node power management entirely through power management resource managers, or Power RMs. Each of the two power management methods mentioned above has its own Power RM implementation. The older Moab-only method uses Python-based scripts to implement a power RM while the newer Moab+Moab Web Services (MWS)-based method uses a Java-based MWS RM power management plug-in that runs much simpler Python-based scripts.

These Power RMs perform all power-related management and monitoring, meaning power state control and power state query, respectively, and only report back to Moab whether a compute node is in a state in which it can run jobs (on) or not (off). All actual power state-aware control and management is performed by the power RMs.

Moab Power RMs

Adaptive Computing provides two power management methods to handle different site scenarios; mainly for site-specific security policies. The older method handles sites with a security policy that does not permit web service-based services, which can be an attack vector, or sites that do not want to run an MWS service.

The newer method uses the MWS RM plug-in feature, which allows an administrator to instantiate a separate RM power management plug-in instance for different partitions, or different compute nodes for situations where different compute node hardware requires the use of different power management commands run from Python scripts.

Power Management Scripts

Each power management method, old or new, employs at some point a script that allows the administrator to customize power management for a site, which may be required because the working reference scripts provided by Adaptive Computing (based on OpenIPMI tools) do not use the power management commands specific to the site's vendor-provided hardware.

Moab System Jobs

Moab performs power management functions through a mechanism known as system jobs. A Moab system job is a special, separately scheduled job that performs some Moab system function (e.g., power management, data-staging) that Moab executes on the Moab head node and not on a compute node. This allows Moab to apply policies such as a job wallclock estimate, etc, to system-related functions, which can aid error recovery procedures, etc.

System jobs perform internal Moab-related functions on Moab's behalf, are nearly always script-based, and usually require some customization by the Moab administrator in order to perform the needed function for the HPC system site. For example, the administrator may have to modify power management scripts so they use a site's hardware vendor-specific power management commands to effect power state changes in compute nodes.

To create a system job, Moab internally submits an administrator-defined script, with a path typically specified as a Moab `*URL` parameter, to itself, which it flags as a system job. Moab schedules the job and because it is flagged as a system job, executes the script on the head node. Moab submits a system job whenever it needs to send a power on or off command to a Power RM. Administrators can easily recognize queued and running power management system jobs in the showq command output as their job id has the format `id.poweron` and `id.poweroff`, where id is the internally generated Moab job id number and `.poweron` and `.poweroff` are suffixes appended to the job id number that represent Moab's on and off commands sent to Power RMs.

Green Policies

Moab provides green policies that automate power management for idle compute nodes, which an administrator can modify and/or configure to control the power state of compute nodes not always in use. These policies allow Moab to dynamically control the power state of compute nodes between the active running state or power-on nodes that may be needed. It also allows Moab to power-off nodes that are idle and wasting energy. Which power state such compute nodes enter depends entirely on the commands the administrator configures and/or modifies in a power RM's scripts and, for the newer Moab+MWS method, on configuration information specified for each MWS RM power management plug-in instance.

The green policies maintain a green idle node pool, the size of which the administrator configures. As jobs start and use idle nodes from the pool, Moab replenishes the pool by performing an `on` command on those compute nodes on which it previously had performed an `off` command, thus bringing them into the idle node pool as they enter into an active running state. When jobs finish and the pool has excess idle nodes, Moab performs an `off` command on the excess nodes, which removes them from the idle pool. Thus, Moab maintains a pool of available idle nodes for immediate use by submitted jobs and reduces energy consumption by powering off any idle nodes in excess of the pool size.

Theory of Operation

Moab itself operates the same regardless of the method of power management, Moab-only or Moab+MWS, chosen. This is especially true for the green policies as Moab simply uses the configured power management method to carry out the policies. In order to know how to configure the different parts and components of each power management method so they work well together, it is necessary for a site administrator to understand how the power management methods work; that is, how the components work together to implement a power management method.

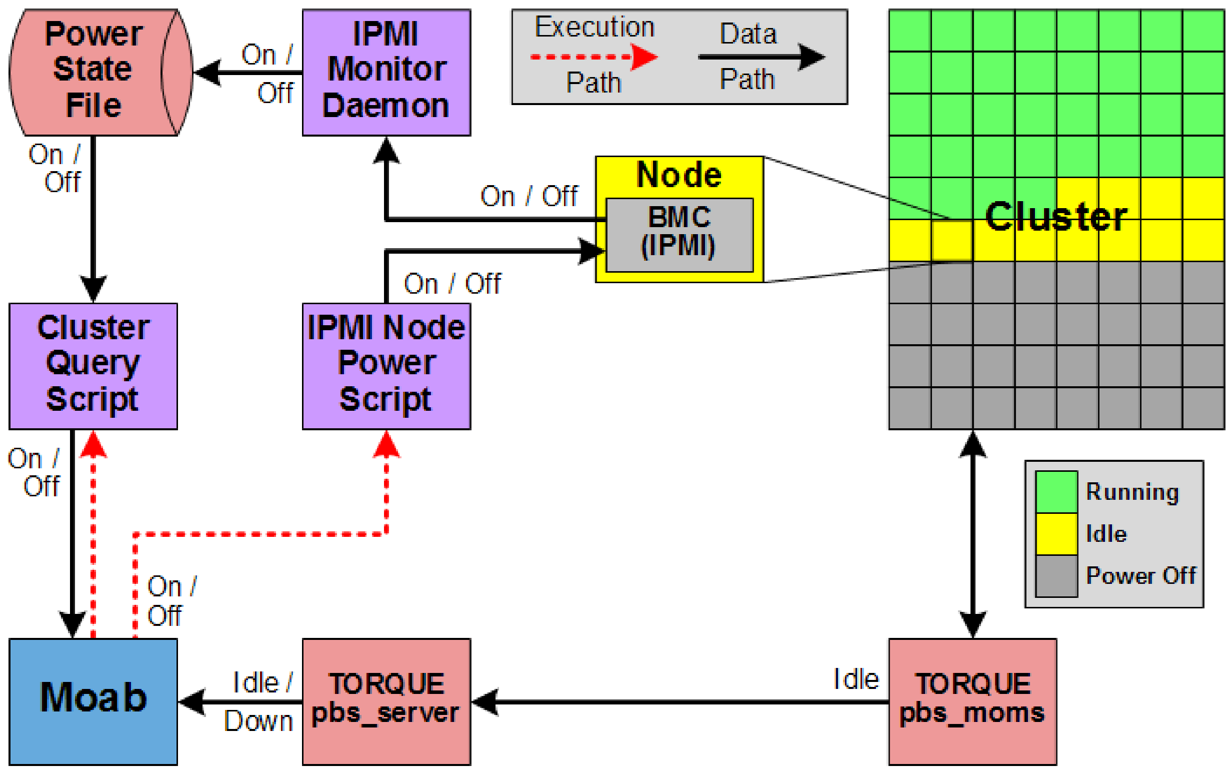
Moab-only Method

The Moab-only method has a Power RM composed entirely of Python-based scripts. The script must maintain a Power Query daemon that queries the power state of all compute nodes and saves their state for Moab to query, the actual power state query Moab runs to find out the current power state of all compute nodes, and a power state control that places compute nodes into the state of on so Moab can schedule jobs to them or into the state of off so energy consumption is minimized and operational costs reduced. The administrator determines what the actual power state Moab's `off` represents by configuring the off command in the power management control script with the actual hardware vendor-supplied command that effects the desired power state (remember, Moab is not aware of actual power states).

The list below enumerates the advantages and disadvantages of the Moab-only method.

- Advantages
  - Do not have to run the MWS service and its MongoDB database.
  - Power management command scripts execute as Moab system jobs.
  - Ability to customize the node power and cluster query power management scripts
    - For more information on how to specify the node power control script, see the NODEPOWERURL parameter.
    - For more information on how to specify the power cluster query script, see the CLUSTERQUERYURL parameter.
  - Moab power control using `mnodectl -m power=[on|off] <nodelist>`.
    - For more information on how to diagnose power states, see mdiag -n.
- Disadvantages
  - More complex scripts to customize.
  - Only global power management control (no partition-based or node-based).
  - Heterogeneous compute node hardware from different vendors requires more modification of the control and query scripts.
  - Reference scripts not scalable (did not take advantage of Python multi-threading).
  - Administrator must maintain complex scripts that must maintain the entire cluster query information.

The following architecture diagram shows the Moab-only architecture and what occurs between its components.



The Python-based IPMI Monitor daemon script running in the background periodically polls the power state of all compute nodes through IPMI using the command customized by the administrator. As it gathers power state information, it saves the information in a text file in a specific format understood by Moab (binary power state). In order to prevent race conditions, it actually writes to a temporary file and then moves the temporary file on top of the permanent file (not shown).

When Moab starts a scheduling cycle/iteration, it directly executes the power RM's Python-based Cluster Query script that reads the permanent text file and delivers the compute node power states to Moab. Moab then performs the scheduling cycle and based on green policies and the state of the HPC cluster will run the IPMI Node Power script as a Moab system job to perform an `on` or `off` (which may be something different than a `power off`) command using the actual commands customized by the administrator in the script.

Moab+MWS Method

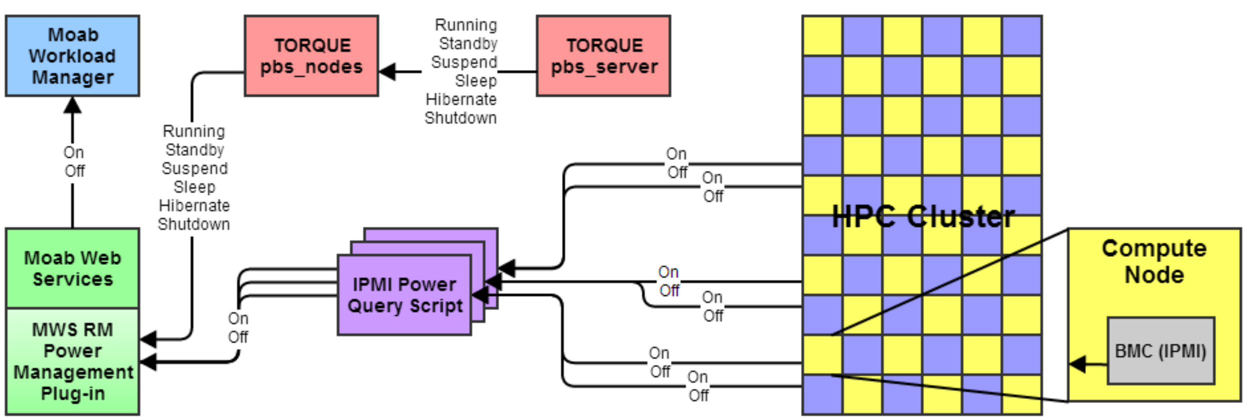
The Moab+MWS method has a Power RM composed of a MWS RM plug-in that encapsulates all power management logic, which itself uses the TORQUE pbsnodes command to effect compute node power state changes into low-power and no-power states of standby and suspend, and hibernate and shutdown, respectively, as well as the IPMI Node Power script to effect compute node power on, power off (pull the plug) and awaken (resume active running state from low-power state). The Power RM Power Management plug-in also performs the power query daemon function identified in the Moab-only method using its built-in power management logic, thus handling more actual power states and allowing much better power control than the Moab-only method offers.

The advantages and disadvantages of the Moab+MWS-based method are enumerated below.

- Advantages
  - More power states to choose from.
    - Low-power states of standby and suspend.
    - No-power states of hibernate and shutdown.
    - On and Off (pull the plug) power states still available.
  - TORQUE power control of low-power and no-power states using `pbsnodes -m <state> <nodelist>`.
    - You can view node power states with the pbsnodes command.
  - Power management command scripts execute as Moab system jobs.
  - Much simpler `moab.cfg` customization and maintenance.
  - Global, partition-based, and node-based granularity for power management control.
  - Heterogeneous compute node hardware from different vendors handled by creating multiple instances of MWS RM power management plug-in with different configurations.
  - Reference scripts are scalable (use Python multi-threading).
  - The MWS RM architecture is easier to support DRAC, ILO, and other protocols.
- Disadvantages
  - Must run the MWS service and its MongoDB database.
  - Configuration of the MWS RM Power Management plug-in and possible multiple instances.

The following architecture diagrams show the Moab+MWS-based method architecture and what occurs between its components.

The diagram below illustrates power state query:



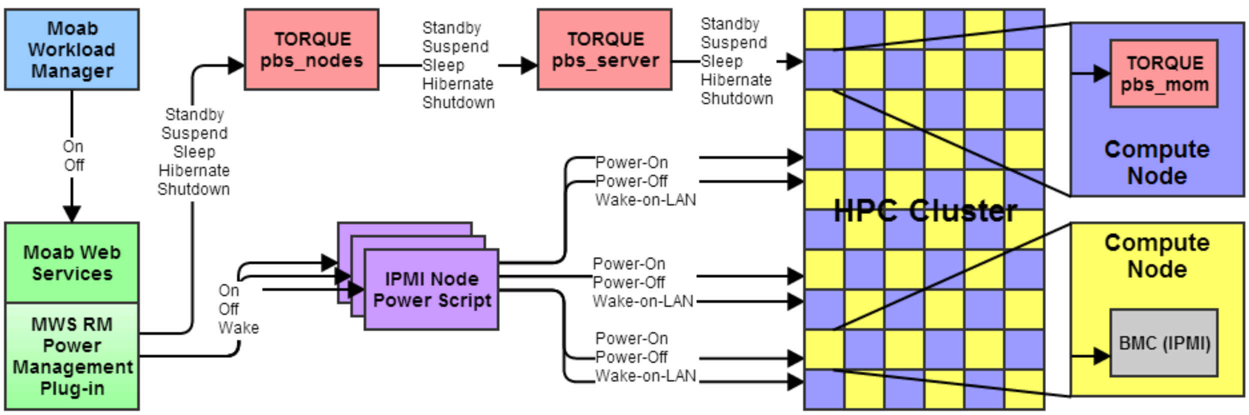
The MWS RM power management plug-in runs the multi-threaded Power Query script for sets of compute nodes which obtain their actual power state through IPMI, or more specifically, a hardware vendor's IPMI implementation (e.g., Dell DRAC, HP iLO, etc), which the RM plug-in saves. It also runs the TORQUE pbsnodes command to obtain the low-power or no-power states that may have been set via

TORQUE earlier (pbs_server retains knowledge of any previous command to set a node's power state to one of the low-power or no-power states).

Note it is quite possible for IPMI to report `off` and TORQUE to report hibernate or shutdown, both of which indicate a compute node has no power, and for IPMI to report `on` and TORQUE to report standby or suspend, both of which indicate a compute node is in a low-power state from which it can be quickly awakened. It is also possible for IPMI to report `on` and TORQUE to report hibernate or shutdown, which can indicate a booting node that has not yet started the TORQUE pbs_mom daemon or a node hibernating or shutting down that has not yet powered off. The MWS plug-in's power management logic reconciles the IPMI and TORQUE reports to produce a single `on` or `off` understood by Moab, which it passes to MWS.

When Moab queries MWS for the current state information of compute nodes at the start of a scheduling cycle/iteration, MWS passes all node information including the binary power on/off Moab understands and the TORQUE node state, at which point Moab has the information it needs to perform green policy-based automated power management.

The diagram below illustrates Moab+MWS power state control interactions.



When Moab detects a condition that requires changing the power state of a compute node, usually as a result of green policies, it performs the appropriate `on` or `off` command as a system job that sends the command to MWS with a list of the host names of compute nodes that should enter an appropriate power state.

MWS interacts with the appropriate MWS RM power management plug-in for each compute node and passes it the `on` or `off` command. For the `off` command, the plug-in examines its configuration of what `off` means and passes the configured standby, suspend, hibernate, or shutdown command to the TORQUE pbsnodes command, or passes the configured `off` command to the Node Power script.

If the RM plug-in executes the TORQUE pbsnodes command for the configured power state and requested list of compute node host names, it sends the command to the pbs_server, which passes the command to each compute node's pbs_mom daemon. The pbs_mom executes software to place the node into the requested state. The pbs_server daemon keeps the requested state in a file for each compute node, which it passes on to the MWS RM power management plug-in as part of a node update report.

> In clusters where there is a TORQUE pbs_server and pbs_mom on the same machine, the administrator should set the **POWERPOLICY** to *STATIC* on this node, because the pbs_server should not be powered down. If the pbs_server is powered down, Moab will be unable to get cluster query updates from all pbs_moms managed by that that pbs_server.
>
> On all TORQUE nodes where pbs_moms are running, the pbs_mom must be configured to auto-start after being rebooted. If the pbs_mom isn't auto-started, the pbs_server will not be able to determine when it has been powered up and entered an idle state, and therefore won't have the ability to inform Moab on a cluster query the node is idle. Refer to **Startup/Shutdown service script for TORQUE/Moab (OPTIONAL)** on page 2488 for Torque/Moab for details on how to have the pbs_mom auto-start on boot.

When the RM plug-in executes the Node Power script for the configured `off` power state and requested list of compute node host names, the script executes its IPMI `on` command (whatever the administrator configured in the script) that tells the node's BMC to power off the node.

When the RM plug-in receives the `on` command from Moab via MWS, it checks the internal power state of each compute node in the requested list of compute node host names. If the internal power state is standby or suspend, the script executes its IPMI `wake` command (whatever the administrator configured in the script) that tells the node's BMC to bump the node into the active running state; otherwise, the script executes its IPMI `off` command (whatever the administrator configured in the script) that tells the node's BMC to power on the node.

> Some operating systems require the Wake-on-LAN bit to be enabled using a tool like ethtool. Also, Wake-on-LAN packets may be blocked by the router, but not always.

In this manner, the MWS RM power management plug-in queries the actual power state of individual compute nodes and returns to Moab the simple binary on/off state it understands for scheduling jobs to compute nodes. Likewise, Moab controls the actual power state of individual compute nodes using only its simple binary on/off command. This method of simple command and simple job-scheduling-ability state enables Moab to remain scalable and responsive for automatic power management control using green policies.

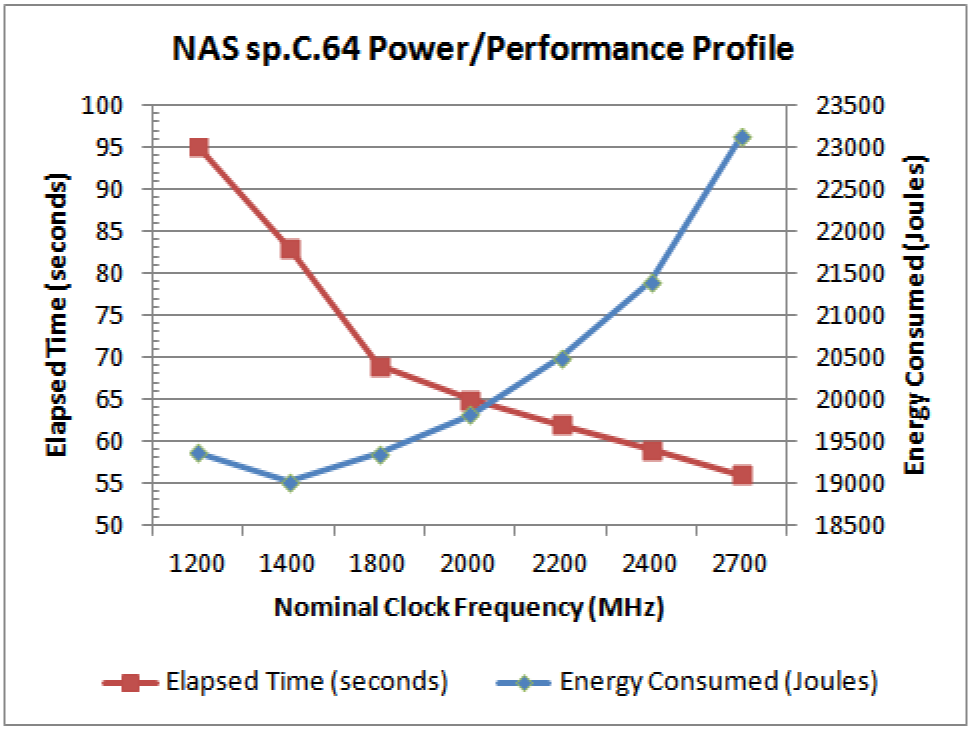
Active Node Power Management

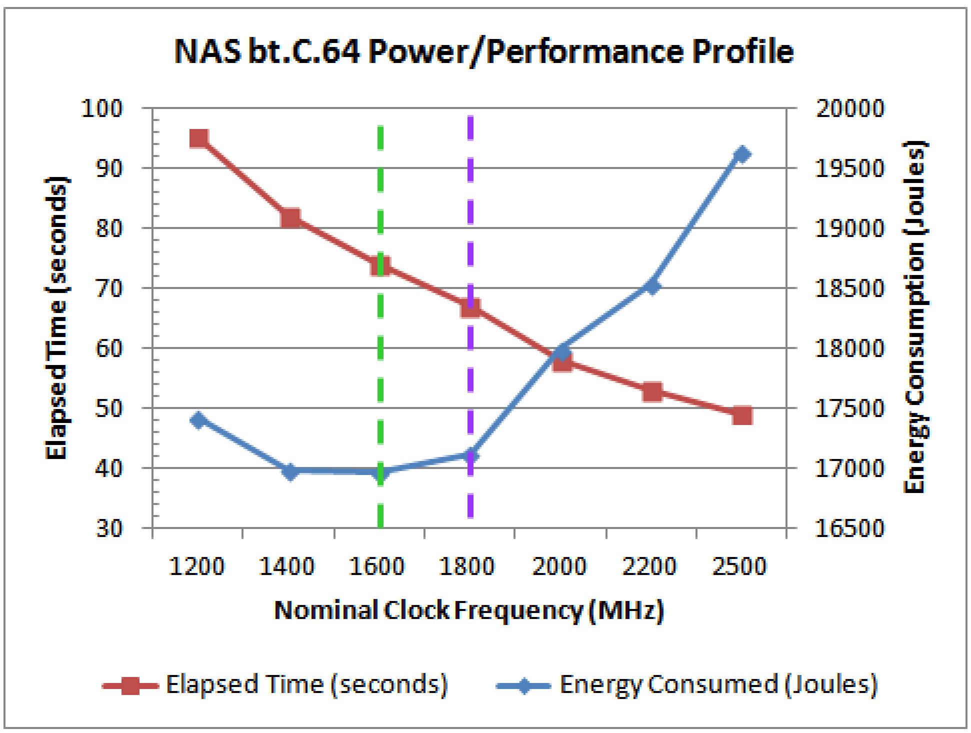
Moab 8.0 and TORQUE 5.0 introduce support for active node power management; that is, the management of energy consumption while a compute node is running a job, which the new CPU Clock Frequency Control feature provides.

The amount of energy consumption savings achievable through the CPU Clock Frequency Control feature is application-dependent. For example, memory, I/O, and/or network-bound applications, especially memory-bound applications, can often drop the clock frequency of their compute nodes' processors and still have the same execution time even though the compute nodes consume less power. Several studies have shown common power savings of 18-20% and one study showed one application saving 30% on power consumption, all of which translate directly into operational cost savings.

Power/Performance Profiling

To determine whether a lower clock frequency will produce energy consumption savings, applications must be profiled; that is, a job running a particular application with the same or equivalent data must be run at different clock frequencies while measuring the energy consumption of the job's compute node. Each pair of frequency/energy consumption data points are plotted in a chart to show the application's power performance profile. The charts below are an example of two such profiles for two NAS benchmark HPC applications.





The intersection of the two lines has no meaning as each line has its own vertical scale, either on the left or the right as noted!

Note both applications do not consume the least energy (vertical dashed green line) when running at the lowest clock frequency, which demonstrate the importance of profiling applications to determine the nominal clock frequency at which energy consumption is the lowest. The charts amply illustrate why a simplistic policy of using the lowest clock frequency is not the best policy when a site's objective is the least energy consumption possible.

If the least energy consumption is not a site's primary objective, but running jobs in a manner that balances energy consumption and job execution time, a power/performance profile chart is very useful to determine the clock frequency that meets a balanced objective. For example, the vertical dashed purple line on the right chart shows that running the bt.C.64 application at 1800 MHz has an increase in energy consumption of ~1% over the minimal energy consumption possible (vertical dashed green line) but results in a ~10% drop in execution time; a possibly very good trade-off!

Obviously, if a site's primary objective is to complete a job as fast as possible but do so saving energy where possible, profiling memory-bound and other bound applications can clearly show the lowest clock frequency at which the application takes longer to execute. The site would then institute a policy that the application should run at the next highest frequency to fulfill the twin objectives of job performance and energy consumption minimization.

For more information about the CPU clock frequency job submission option, see CPUCLOCK resource manager extension of msub -l.

Job Templates

Most users will not care or want to know about clock frequency control, so administrators can use a job template to specify the CPU clock frequency at which a particular recurring job should execute. A clock frequency specified on a job template overrides a clock frequency given on the job submission command line or inside a job script file with TORQUE PBS commands. This order of precedence allows an administrator to control clock frequency for commonly used applications and jobs based on site policies and objectives.

For more information about using a CPU clock frequency job submission option in job templates, see the CPUCLOCK job template extension attribute.

Idle node power management

Moab has so-called green policies that together configure Moab to manage and maintain a pool of idle nodes in an active running state so it can immediately schedule jobs to them. When Moab does so and diminishes the pool's idle compute node quantity, it powers on compute nodes by performing an `on` command for nodes in a powered-down state (actually, in a low-power or no-power state) to bring them on-line in order to replenish the pool of idle nodes up to its configured size. When jobs end and the idle node exceed the configured idle node pool size and there are no jobs to run on the now-idle nodes, Moab will power off excess idle nodes by performing an `off` command. In this manner, Moab achieves a site's power management and energy consumption objectives through the configured green policies.

See the Moab-only Method Architecture diagram above to see the color-coded compute nodes in the diagram's cluster illustrating Moab's green idle node pool management. The green nodes represent nodes running jobs, the yellow nodes are idle nodes in a green pool of size 12, and the gray nodes represent `off` nodes. Note Moab does not know what actual power state `off` means; what it means will be based on command customization inside Moab-only method scripts or Moab+MWS plug-in configuration information.

In order to perform green policy management of an idle node pool, Moab must first be configured to use either the Moab-only or the Moab+MWS method of power management. It is `best practice` to configure power management first and test its configuration before configuring green policies. Thus, if power management is misconfigured, an administrator will know it is the power management configuration and/or scripts and not the green computing policies that are incorrect. If the manual power management commands for the configured power management method work, green computing will work using the configured power management method. For information on how to configure each power management method in Moab, see **Enabling green computing** on page 714.

Green Policy Configuration

There are several green policies that affect how Moab performs green idle node pool management using automated power management operations. The policies are configured in the same manner regardless of the power management method used, whether Moab-only or Moab+MWS. The other sections of this chapter describe how to configure green policies that manage the idle node pool for site energy management objectives.

Related topics

- **Enabling green computing** on page 714
- **Deploying Adaptive Computing IPMI scripts** on page 712
- **pbsnodes** on page 2356

How-to's

Deploying Adaptive Computing IPMI scripts

Context

If you want to enable green computing on your system using the Adaptive Computing supplied IPMI reference scripts, follow the steps here. The IPMI scripts provided are meant as a reference for you to configure the solution to your environment, but can also be used as-is.

Prerequisites

- OpenIPMI and ipmitool must be installed and working.
- All nodes must have the same IPMI username and password.
- You must know the IPMI host names and/or IPMI IP addresses of your nodes.
- Python must be installed. The provided IPMI scripts were developed using Python 2.6.5.
- You must identify your Moab home directory. These instructions assume the default Moab home directory of `/opt/moab`.
- You must identify your Moab tools directory. These instructions assume the default Moab tools directory of `/opt/moab/tools`.

To deploy the Adaptive Computing IPMI scripts

1. Edit the `/opt/moab/tools/ipmi/config.py` script:
   a. Set **self.ipmiuser** to the IPMI username for your nodes.
   b. Set **self.ipmipass** to the location of the IPMI password file (`/opt/moab/passfile.txt` by default).

      > The permissions for the directory and the password file itself should be set so that they can be read only by root or the Moab user running the script.

   c. Set **self.homeDir** to your Moab home directory.
   d. If desired, change the **self.pollInterval** value. This is the interval, in seconds, between polls from the IPMI monitoring script.
   e. The **self.ipmifile** value is the name of a temporary file where the cluster query information is stored. You can change this or leave it alone.
   f. The **self.bmcaddrmap** value is the filename for the Moab node name/IPMI mapping. The file must exist in the Moab home directory and will be created in the next step.
2. Create a `node-bmc.txt` file in the Moab home directory. The file must contain a space-delimited list of Moab node names that map to the IPMI host names or IP address. For Example:

```
node01 node01_ipmi    # For all three of these entries, the first value is the
node02 node02_ipmi    # node name as Moab knows it. The second value is either
node03 10.1.1.1       # the node IPMI name or IPMI IP address.
```

3. Configure the `moab.cfg` file for green computing as described in Enabling green computing. Use the `ipmi.mon.py` script for the `CLUSTERQUERYURL` and the `ipmi.power.py` script for the `NODEPOWERURL`.
4. Restart Moab and verify green computing is working correctly. If you encounter trouble, see the Troubleshooting green computing topic for help.

Related topics

- **Enabling green computing** on page 714
- **Troubleshooting green computing** on page 721
- **Adjusting green pool size** on page 719
- **Handling power-related events** on page 719
- **Maximizing scheduling efficiency** on page 720

Choosing which nodes Moab powers on or off

Context

Moab can use the GREENPOOLPRIORITYF function to determine which nodes to power on or off. The PRIORITY node allocation policy is used to determine which nodes to allocate workload to. When Moab can no longer allocate workload to available nodes, it begins to power nodes on in the order specified by the GREENPOOLPRIORITYF function.

To choose which nodes Moab powers on or off

1. Set a GREENPOOLPRIORITYF function to describe which order nodes should be selected for power on/off actions. GREENPOOLPRIORITYF uses the PRIORITY node allocation policy options and syntax.

```
GREENPOOLPRIORITYF '10*RANDOM'
```

This tells Moab to randomly choose a node to power on to meet workload demands, and to randomly choose an idle node to power off to meet the MAXGREENSTANDBYPOOLSIZE goal.

To choose which nodes Moab allocates jobs to

1. Set a PRIORITY node allocation policy that uses power as the major factor. This causes Moab to allocate jobs to nodes that are already powered on. When no nodes are available to meet this policy, Moab uses the GREENPOOLPRIORITYF function to turn on nodes that are powered off.

```
NODEALLOCATIONPOLICY  PRIORITY
NODECFG[DEFAULT] PRIORITYF='10000*POWER + 10*RANDOM'
```

The nodes with the highest priority for workload are the nodes that are powered on. After that, Moab randomly allocates workload.

Related topics

- **Adjusting green pool size** on page 719
- **Maximizing scheduling efficiency** on page 720

Enabling green computing

Context

There are two ways to do green computing in Moab. With just Moab, nodes can be turned on or off. With MWS, however, you can put nodes into several low-power states. The MWS solution is also more scalable. The supported low-power states are:

- Running
- Standby
- Suspend
- Hibernate
- Shutdown

Nodes cannot be moved from one low-power state to another. The node must go from low-power to running, and then to the new low-power state.

To enable green computing with Moab and MWS

1. Edit `moab.cfg` to use MWS for green computing:

   a. Configure the POWERPOLICY attribute of the NODECFG parameter. The default value is *STATIC*. Set it to *OnDemand*.

   b. Set the resource manager type as *MWS*

   c. Set `FLAGS=UserSpaceIsSeparate` for the MWS resource manager.

   d. Point `BASEURL` to your MWS server.

```
NODECFG[DEFAULT]         POWERPOLICY=OnDemand
RMCFG[mws]               TYPE=MWS
RMCFG[mws]               FLAGS=UserSpaceIsSeparate
RMCFG[mws]               BASEURL=http://localhost:8080/mws
```

2. Configure the MWS **Power Management Plugin** on page 1736.

To enable green computing with just Moab

1. Edit `moab.cfg` to enable green computing. There are four things you must configure for basic functionality of green computing:

   a. Configure the POWERPOLICY attribute of the NODECFG parameter. The default value is *STATIC*. Set it to *OnDemand*.

   b. Configure a power provisioning resource manager to be TYPE=*NATIVE* and RESOURCETYPE=*PROV*. The resource type of *PROV* means the RM works only with node hardware and not workloads.

   c. Configure a CLUSTERQUERYURL attribute of the power provisioning RM to point to the power query script you'd like to use. Moab uses this script to query the current power state of the nodes. **CLUSTERQUERYURL** is traditionally used as a workload query but is also used by green computing for the node power state query. Adaptive Computing provides a reference IPMI script you can use.

   d. Configure a NODEPOWERURL attribute of the power provisioning RM to point to the power action script you'd like to use. Moab uses this script to turn nodes on or off. Adaptive Computing provides a reference IPMI script you can use.

```
NODECFG[DEFAULT] POWERPOLICY=OnDemand
RMCFG[ipmi] TYPE=NATIVE RESOURCETYPE=PROV
RMCFG[ipmi] CLUSTERQUERYURL=exec://$TOOLSDIR/ipmi/ipmi.mon.py
RMCFG[ipmi] NODEPOWERURL=exec://$TOOLSDIR/ipmi/ipmi.power.py
```

Sample moab.cfg for green computing

Below is a sample `moab.cfg` configuration file of a green computing setup using the Adaptive Computing IPMI scripts.

```
################################################################################
#
#  Use 'mdiag -C' to validate config file parameters
#
################################################################################

SCHEDCFG[Moab]        SERVER=myhostname:5150
ADMINCFG[1]           USERS=myusername,root
TOOLSDIR              /$HOME/tools
LOGLEVEL              1

################################################################################
#
#  Basic Resource Manager configuration
#
#  For more information on configuring a Resource Manager, see:
#  docs.adaptivecomputing.com
#
################################################################################

RMCFG[local]   TYPE=NATIVE
RMCFG[local]   CLUSTERQUERYURL=exec://$HOME/scripts/query.resource
RMCFG[local]   WORKLOADQUERYURL=exec://$HOME/scripts/query.workload

RMCFG[local]   JOBSUBMITURL=exec://$HOME/scripts/submit.pl
RMCFG[local]   JOBSTARTURL=exec://$HOME/scripts/job.start
RMCFG[local]   JOBCANCELURL=exec://$HOME/scripts/job.cancel
RMCFG[local]   JOBMODIFYURL=exec://$HOME/scripts/job.modify
RMCFG[local]   JOBREQUEUEURL=exec://$HOME/scripts/job.requeue
RMCFG[local]   JOBSUSPENDURL=exec://$HOME/scripts/job.suspend
RMCFG[local]   JOBRESUMEURL=exec://$HOME/scripts/job.resume

##################################
# GREEN configuration:
##################################
# Turn on "green" policy. (This is the policy that enables green computing).
# Here we are doing it for all nodes, but it can be controlled on a node-by-node
basis
# Default is STATIC, which means green computing is disabled.
#NODECFG[DEFAULT]  POWERPOLICY=STATIC
NODECFG[DEFAULT]  POWERPOLICY=OnDemand

# Use the MWS RM and the MWS power management plugin for power provisioning
# and power state.
```

```
RMCFG[mws]   TYPE=MWS
RMCFG[mws]   FLAGS=UserSpaceIsSeparate
RMCFG[mws]   BASEURL=http://localhost:8080/mws

# We want green policy to work so it allocates jobs to compute nodes already
# powered on and will power on powered-off compute nodes only when there are
# no powered-on compute nodes available. This requires using the PRIORITY
# node allocation policy with a PRIORITYF function that has the POWER variable
# as the greatest contributing factor to the function (1 = powered-on,
# 0 = powered-off).
# If we want all compute nodes to operate under green policy, we can assign
# the PRIORITYF function to the default node configuration, which is easier
# than assigning it to individual compute nodes. If only some compute nodes
# should operate under green policy, then the PRIORITYF function must be
# configured for the individual nodes. Note the POWER variable must be the
# largest factor in the function below; it is assigned the largest multiplier,
# which should be greater than the sum of all other factors! Doing so forces
# Moab to use all eligible powered-on nodes for workload placement before
# powering on any eligible powered-off nodes.

# Enable PRIORITYF functionality
#NODEALLOCATIONPOLICY  PRIORITY

# Use a priority function that uses power as the major factor (plus some other
imaginary factors)
#NODECFG[DEFAULT]      PRIORITYF='1000000*POWER + 1000*factor2 + 100*factor3...'
# Use a priority function where power is the only factor.
#NODECFG[DEFAULT]      PRIORITYF='10000*POWER'
# Use a priority function that adds some randomness but uses power as the major
factor.
#NODECFG[DEFAULT]      PRIORITYF='10000*POWER + 10*RANDOM'

# Set a priority function that specifies the order nodes should be chosen to power
# up/down. By default, Moab will start at the top of the node list and go down.
Some
# installations want to rotate power cycles among nodes in a different order.
# The configuration below forces Moab to power on/off random nodes, which
# eventually guarantees all nodes occasionally go through a power cycle.
#GREENPOOLPRIORITYF '10*RANDOM'

# Ensure we are recording power management events
# (powering on and off nodes are recorded as "node modification" events).
#RECORDEVENTLIST +NODEMODIFY

# Set the size of the standby pool. This is the number of idle nodes that will
# be powered on and idle. As the workload changes, Moab turns nodes on
# or off to try to meet this goal.
# Default value is 0
MAXGREENSTANDBYPOOLSIZE 5

# Set the length of time that it takes to power a node on/off. This will be the
# walltime of the system job that performs the power operation and should be the
# maximum expected time. If Moab detects (via the power RM) that the power
# operations have all completed, the system job will finish early.
# Default value is 10 minutes (600)
PARCFG[ALL]      NODEPOWEROFFDURATION=600
PARCFG[ALL]      NODEPOWERONDURATION=600
# Set the length of time a node should remain idle before it is powered off.
# This prevents Moab from immediately powering off nodes that have just finished
# a job. Increasing this number should decrease power on/off thrashing
# This should be set higher than NODEPOWEROFFDURATION and/or NODEPOWERONDURATION
```

```
NODEIDLEPOWERTHRESHOLD 660

# If a node fails to power on, we need to remove it from the available nodes so
# Moab won't keep [re-]trying to power it on. Do this by setting a reservation
# on the failed node to give time for manual investigation.
#RMCFG[torque] NODEFAILURERSVPROFILE=failure
#RSVPROFILE[failure] DURATION=3600
```

Related topics

- **Deploying Adaptive Computing IPMI scripts** on page 712
- **Choosing which nodes Moab powers on or off** on page 713
- **Adjusting green pool size** on page 719
- **Handling power-related events** on page 719
- **Maximizing scheduling efficiency** on page 720
- **Troubleshooting green computing** on page 721
- **Power Management Plugin** on page 1736

Adjusting green pool size

Context

The MAXGREENSTANDBYPOOLSIZE parameter allows you to allocate the number of nodes to keep powered on in the standby pool. This is the number of idle nodes that are allowed be powered on and idle. As the workload changes, Moab turns nodes on or off to try to meet this goal. The default value is 0.

To adjust the green pool size

1. Modify the MAXGREENSTANDBYPOOLSIZE parameter with the number of nodes you want Moab to keep powered on for the standby pool.

```
MAXGREENSTANDBYPOOLSIZE 10
```

Moab keeps up to 10 idle nodes powered on to be kept on standby.

Related topics

- **Maximizing scheduling efficiency** on page 720
- **Choosing which nodes Moab powers on or off** on page 713

Handling power-related events

Context

Power actions are considered NODEMODIFYURL events and are not recorded by default, but you can configure Moab to include power-related events in the logs. Also, if a node fails to turn on (or off), it's best to associate a reservation on the failed node so that Moab won't keep trying to perform the power action over and over.

To configure Moab to record power-related events

1. Modify the **RECORDEVENTLIST** parameter.

```
RECORDEVENTLIST +NODEMODIFY
```

Power-related events are logged to the Moab log file.

To put a reservation on a node that fails to perform a power action

1. Configure the NODEFAILURERSVPROFILE attribute of **RMCFG** and create an **RSVPROFILE** with a high duration.

```
RMCFG[torque] NODEFAILURERSVPROFILE=failure
RSVPROFILE[failure] DURATION=3600
```

Nodes that fail to power on or off have a 1-hour reservation placed on them.

Related topics

- **RECORDEVENTLIST** on page 1005
- **Event Logs** on page 680

Maximizing scheduling efficiency

Context

When considering whether to power a node on or off, Moab can take into account the amount of time that it takes to power on or power off the node. With this information, Moab can keep an idle node powered on if it knows that workload in the queue will be ready for the node in less time that it takes to power off/power on the node.

Moab can also wait to shut down nodes after they've been idle for a specific amount of time.

To specify node power on/power off duration

1. Modify the **NODEPOWERONDURATION** and **NODEPOWEROFFDURATION** attributes of **PARCFG** with the maximum amount of time it takes for your nodes to power on/power off. Make sure to use the keyword `ALL` for the resource manager name to avoid cases where Moab won't consider the power on/off duration for a node before making a power action decision.

```
PARCFG[ALL] NODEPOWERONDURATION=2:00
PARCFG[ALL] NODEPOWEROFFDURATION=2:00
```

If a node goes idle and has to wait for workload, Moab will not power off the node if the workload will be available within 4 minutes or less.

To shut down on nodes after they've been idle for a specified time

1. Modify the NODEIDLEPOWERTHRESHOLD parameter with the duration (in seconds) you want Moab to wait before shutting down an idle node. The default value is 60 seconds. Increasing the number should decrease power on/off thrashing. This should be set higher than **NODEPOWERONDURATION**

and/or **NODEPOWEROFFDURATION**.

```
NODEIDLEPOWERTHRESHOLD 300
```

Moab will wait 5 minutes before shutting down a node that has become idle.

Related topics

- **Adjusting green pool size** on page 719
- **Choosing which nodes Moab powers on or off** on page 713

Putting idle nodes in power-saving states

Context

When nodes exceed their idle threshold limits, the default behavior is to turn the nodes off. With the **NODEIDLEPOWERACTION** parameter, you can choose which power-saving state to put idle nodes into. This parameter is configured at the partition level. Configuring it for the `ALL` partition effectively makes it a global parameter.

To specify what to do with idle nodes

1. Modify the **NODEIDLEPOWERACTION** parameter.

```
NODEIDLEPOWERTHRESHOLD 300
PARCFG[ALL] NODEIDLEPOWERACTION SLEEP
```

All nodes that are idle for more than 5 minutes are put into a sleep state.

Related topics

- [Enter xrefs to related topics here. Use a simple ul style and apply the li.links style to the list items.]
-

Troubleshooting green computing

Context

If you've enabled green computing and are having trouble, here are some tips that can help you determine the cause of the issues you encounter. These tips are specifically for the Adaptive Computing supplied IPMI scripts, but can be generalized for whatever power management solution you use. Simply substitute your power management system, power query script (as specified by **CLUSTERQUERYURL**), and power action script (as specified by **NODEPOWERURL**) where appropriate.

Verify your IPMI access

1. Use the ipmitool command to verify you have access to the IPMI interface of your nodes. Try getting the current power state of a node. The syntax is `ipmitool -I lan -H` *`<host>`* `-U` *`<IPMI username>`* `-P` *`<IPMI password>`* `chassis power status`.

```
$ ipmitool -I lan -H qt06 -U ADMIN -P ADMIN chassis power status

Chassis Power is off
```

Verify the power query (CLUSTERQUERYURL) script is working

1. Execute the `impi.mon.py` script (should be found in `/<MOABHOMEDIR>/tools/ipmi`) to start the monitor.

```
$ cd /opt/moab/tools/ipmi
$ ./ipmi.mon.py
```

2. Execute the script again. The following is an example of the expected output:

```
$ ./ipmi.mon.py

qt09  GMETRIC[System_Temp]=27 GMETRIC[CPU_Temp]=25 POWER=on State=Unknown
qt08  GMETRIC[System_Temp]=31 GMETRIC[CPU_Temp]=25 POWER=on State=Unknown
qt07  GMETRIC[System_Temp]=30 GMETRIC[CPU_Temp]=29 POWER=on State=Unknown
qt06  GMETRIC[System_Temp]=Disabled GMETRIC[CPU_Temp]=Disabled POWER=off
State=Unknown
```

If the POWER attribute is not present the script is not working correctly.

Verify the power action (NODEPOWERURL) script is working

1. Execute the `ipmi.power.py` script (should be found in `/<MOABHOMEDIR>/tools/ipmi`) to see if you can force a node to power on or off. The syntax is `ipmi.power.py <node>,<node>,<node>... [off|on]`

```
$ /opt/moab/tools/ipmi/ipmi.power.py qt06 off
```

This example is trying to power off a node named qt06.

2. Verify the machine's power state was changed to what you attempted in the previous step. You can do this remotely via two methods:

   a. If the cluster query script is working, you can use that to verify the current power state of the node.

   b. If you have IPMI access, you can use the ipmitool command to verify the current power state of the node.

Verify the scripts are configured correctly

1. Run the mdiag -R command to verify your IPMI resource manager configuration.

```
$ mdiag -R -v
RM[ipmi]      State: Active  Type: NATIVE  ResourceType: PROV
Timeout:            30000.00 ms
Cluster Query URL:  exec://$TOOLSDIR/ipmi/ipmi.mon.py
Node Power URL:     exec://$TOOLSDIR/ipmi/ipmi.power.py
Objects Reported:   Nodes=3 (0 procs)  Jobs=0
Nodes Reported:     3 (N/A)
Partition:          SHARED
Event Management:   (event interface disabled)
RM Performance:     AvgTime=0.05s  MaxTime=0.06s  (176 samples)
RM Languages:       NATIVE
RM Sub-Languages:   NATIVE
```

2. Run the mdiag -G command to verify that power information is being reported correctly.

```
$ mdiag -G

NodeID      State      Power    Watts   PWatts
 qt09       Idle       On       0.00    0.00
 qt08       Idle       On       0.00    0.00
 qt07       Idle       Off      0.00    0.00
```

Verify the scripts are running

1. Once green is configured and Moab is running, Moab should start the power query script automatically. Use the ps command to verify the script is running.

```
$ ps -ef | grep <CLUSTERQUERYURL script name>
```

If this command does not show the power query script running then your settings in `moab.cfg` aren't working.

Verify Moab can power nodes on or off

1. Use the mnodectl command to turn a node on or off. The syntax is `mnodectl -m power=[off|on] <node>`.

```
mnodectl -m power=off qt06
```

Moab should turn off the node named qt06.

   a. Moab generates a system job called `poweron-<num>` or `poweroff-<num>` job as shown in showq. The system job calls the `ipmi.power.py` (**NODEPOWERURL**) script to execute the command.

   b. Moab waits until the cluster query reports the correct data. In this case, the `ipmi.power.py` script reports that the power attribute has changed.

   c. Moab does not change the power status based on the power script return code. Rather, Moab completes the system power job when it detects the power attribute has changed as indicated by the cluster query script.

Related topics

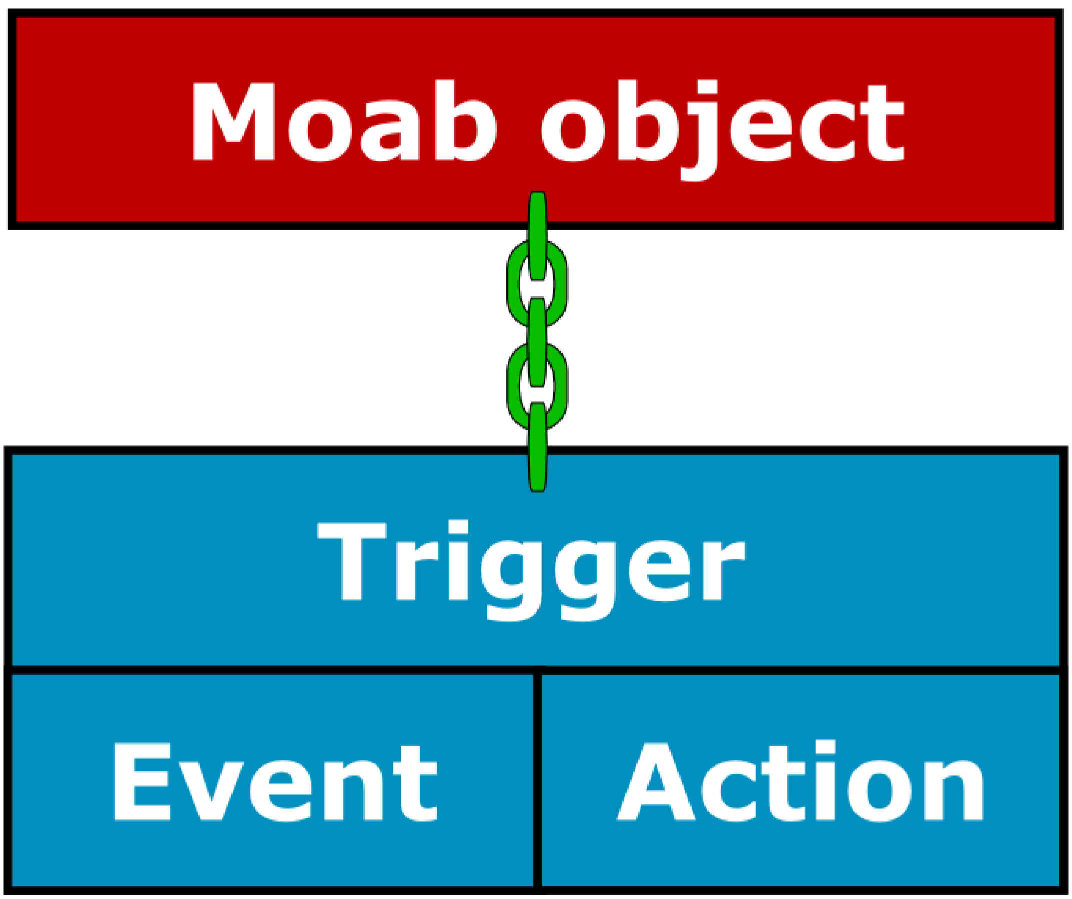
- **Enabling green computing** on page 714
- **Deploying Adaptive Computing IPMI scripts** on page 712

Object triggers

About object triggers

Moab triggers are configurable actions that respond to an event occurring on a Moab object. A trigger is attached to an object and consists of both an event that may take place on the object and the action that the trigger will take.

Image 3-9: Trigger attachment



> Triggers are a powerful tool. Extreme caution should be taken when using them. They are useful in creating automatic responses to well-understood Moab events; however, by default triggers run as root and do exactly as they are told, meaning they require great thought and consideration to ensure that they act appropriately in response to the event.

Use case

An administrator wants to create the following setup in Moab:

When a node's temperature exceeds 34°C, Moab reserves it. If the temperature increases to more than 40°C, Moab requeues all jobs on the node. If the node's temperature exceeds 50°C, Moab shuts it down. Moab removes the node's reservation and unsets the variables when the node cools to less than 25°C.

The administrator wants to receive an email whenever any of these events occur. All of this can be configured in Moab using triggers. To see a full example for this use case, see **Node maintenance example** on page 752.

Sub content

- **About trigger variables** on page 754

How-to's

- **Creating a trigger** on page 727
- **Using a trigger to send email** on page 731
- **Using a trigger to execute a script** on page 733
- **Using a trigger to perform internal Moab actions** on page 733
- **Requiring an object threshold for trigger execution** on page 734
- **Enabling job triggers** on page 734
- **Modifying a trigger** on page 735
- **Viewing a trigger** on page 736
- **Checkpointing a trigger** on page 736

References

- **Job triggers** on page 737
- **Node triggers** on page 738
- **Reservation triggers** on page 740
- **Resource manager triggers** on page 741
- **Scheduler triggers** on page 742
- **Threshold triggers** on page 743
- **Trigger components** on page 744
- **Trigger exit codes** on page 752
- **Node maintenance example** on page 752
- **Environment creation example** on page 753

How-to's

Creating a trigger

Context

Three methods exist for attaching a trigger to an object:

- Directly to the object via the command line
- Directly to the object via the configuration file
- As part of a template via the configuration file

<attr>=*<val>* pair delimiters, quotation marks, and other elements of the syntax may differ slightly from one method/object combination to another, but creating any trigger follows the same basic format:

`<attr>=<val>[[{&,}<attr>=<val>]...]`

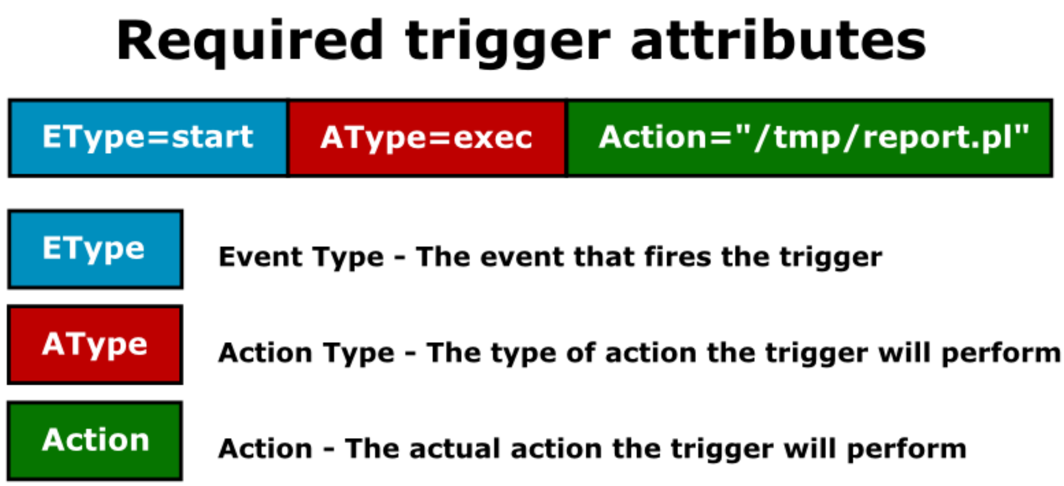
The beginning of the trigger is set off by the keyword *trigger*. It is followed by a delimited list (typically by commas) of *<attr>*=*<val>* pairs.

Each method of trigger creation can only be used for certain Moab objects. The following table displays which objects can receive triggers via each method. The links contain examples.

Method	Objects
Command line	job, reservation; a trigger can be attached to any existing object using mschedctl -c
Configuration file	node, reservation, RM, scheduler
Template	job, reservation

Triggers are composed of attributes. Only three are required for each trigger: an EType (event type), an AType (action type), and an Action.

Image 3-10: Required trigger attributes



Other attributes exist to further customize triggers. See **Trigger components** on page 744 for more information.

To create a Moab trigger

1. Choose an object to which, and a method by which, you will attach the trigger. Use the format and examples described in its corresponding documentation:
   - **Job triggers** on page 737
   - **Node triggers** on page 738
   - **Reservation triggers** on page 740
   - **Resource manager triggers** on page 741
   - **Scheduler triggers** on page 742

   If the trigger is to be attached to a job, you must first enable job triggers (see **Enabling job triggers** on page 734 for more information.). Please carefully review the warning before doing so.

2. Decide whether to attach the trigger via the command line or configuration file. Verify the correct syntax.
3. Set the **EType** equal to whichever event will launch the trigger if and when it occurs on the object.

   Each object has a different lifecycle, so not every event type will occur on every object. For a list of valid **ETypes** for your selected object, see the corresponding object reference page linked in step 1.

a. To modify the timing of the trigger in any of the following ways, see **Event-modifying trigger components** on page 748.

- To set the trigger as rearmable and specify the amount of time the trigger must wait before firing again.
- To set an amount of time before or after the event that the trigger will fire (See **Offset** on page 748 for restrictions).
- To set a specific threshold and the amount of time that the object must meet that threshold before the trigger will fire.

4. Configure the action that the trigger will take when the event happens. To do so, you must set the **AType** to a valid value for your object and specify the action. For instance, to execute a script, set the **AType** to *exec* and the **Action** to the location of the script in quotation marks. Include the name of the object on which the script will run.

```
NODECFG[node01] TRIGGER=EType=fail,AType=exec,Action="node.fail.sh node01"
```

a. To modify the action in any of the following ways, see **Action-modifying trigger components** on page 750.

- To specify environment variables available to the trigger
- To set a flag on the trigger
  - To attach any stderr output generated by the trigger to the parent object
  - To destroy the trigger if its object ends or cancels
  - To tell Moab to checkpoint the trigger
  - To set the trigger as periodic
  - To pass the object's XML information to the trigger's stdin
  - To set the trigger to reset if its object is modified
  - To set the trigger to fire under the user ID of the object's owner
- To specify an amount of time that Moab will suspend normal operation to wait for the trigger to execute
- To allot an amount of time that the trigger will attempt to run before it is marked as unsuccessful and the process, if any exists, is killed
- Set a maximum number of times that a trigger will attempt to fire before it fails

b. To give the trigger a name or description, see **Organizational trigger components** on page 751.

c. To configure the trigger to set or unset a variable when it fires or to require a variable to fire, see **Setting and receiving trigger variables** on page 755.

Creating VM triggers

Context

Triggers can be attached to virtual machines on the command line using the mvmctl command.

To create a VM trigger via the command line

1. Type the mvmctl -m command to modify a virtual machine. Use `trigger=` to mark the beginning of the trigger configuration.

```
> mvmctl -m trigger=
```

2. Set the EType to start. End with a backslash (\) and ampersand (&).

```
> mvmctl -m trigger=EType=start\&
```

3. Specify the action the trigger should take when the event occurs by setting the AType attribute to changeparam, exec, internal, query, or submit. End with a backslash and ampersand.

```
> mvmctl -m trigger=EType=start\&AType=exec\&
```

4. If the trigger launches when the job reaches a threshold, define the threshold.
5. Use the Action attribute to specify the action the trigger will take. Use single quotes.

```
> mvmctl -m trigger=EType=start\&AType=exec\&Action='trig.py $OID $HOSTLIST'
```

6. Configure any desired time-related attributes (The offset time, whether a trigger can fire multiple times, how often, etc.).Insert a comma between the time attribute(s) and the action.

```
> mvmctl -m trigger=EType=start\&AType=exec\&Action='trig.py $OID
$HOSTLIST',Offset=30
```

   Moab launches the `trig.py` script 30 seconds after the VM starts.

7. Configure any desired variables. Verify that all attributes are separated by commas.

```
> mvmctl -m trigger=EType=start\&AType=exec\&Action='trig.py $OID
$HOSTLIST',Offset=30,sets=vmStart
```

8. Set any desired flags.
9. Submit the trigger.

Using a trigger to send email

Context

Mail triggers can be attached to nodes, jobs, reservations, and the scheduler. The recipient of the email depends on the object to which the trigger is attached. To select different recipient(s) and add flexibility to formatting, send email via a script using an exec trigger.

To use a trigger to send email

1. For objects that send mail to the primary user, you must configure the Moab administrator email using the **MAILPROGRAM** on page 971 parameter.
2. Create a trigger on one of the four valid objects listed below, setting the **AType** to mail and the **Action** to the body of the message inside of quotation marks.

Object	Recipient
Node	The primary user (the first user listed in ADMINCFG[1], typically root)
Job	The job's owner
Reservation	The primary user
Scheduler	The primary user

3. When attaching a mail trigger to all objects of a certain type, use internal variables in the **Action** to add information that is specific to an object, such as the ID, owner, time the event occurred, etc. A variable must be preceded by a dollar sign ($).

Variable	Description
$OID	Name of the object to which the trigger is attached
$OTYPE	The type of object to which the trigger is attached
$TIME	Time the trigger launched
$HOSTLIST	Hostlist of the trigger's object (jobs and reservations)
$OWNER	Owner of the trigger's object (jobs and reservations)
$USER	User (jobs and reservations)

The variable is replaced with the information described above. For example, the following trigger is configured on all nodes:

```
NODECFG[DEFAULT] EType=fail,AType=mail,Action="node $OID failed at $TIME"
```

When, for example, node `node03` fails, an email is sent to the primary user with a message with the subject line "node node03 started on Sat Aug 18 11:42:00".

Using a trigger to execute a script

Context

Exec triggers launch a program or script when the event occurs. A few examples of what a script might do in response to an event include:

- Execute an external program
- Send a complex email to any desired recipient(s)
- Collect diagnostics

> It is important to note that when a script runs via a trigger, Moab forks and performs a direct OS exec, meaning there will be no pre-processing of the command by the shell. In addition, the script runs in a new, reduced environment without the same settings and variables as the environment from which it stemmed. The script must be able to run in the reduced environment.

To use a trigger to execute a script

1. Create or locate the script and note its location.
2. Create a trigger on the desired object, setting the **AType** to *exec* and the **Action** to location of the script or program.

```
JOBCFG[temp1] TRIGGER=EType=start,AType=exec,Offset=03:00,Action="/tmp/monitor.pl"
```

Jobs with the `temp1` template receive a trigger that executes `monitor.pl` three minutes after the job starts.

Using a trigger to perform internal Moab actions

To perform internal actions in Moab with a trigger

- Create a trigger on a job, node, or reservation, setting the **AType** to *internal* and the **Action** to one of the following:
  - *node:-:reserve* - reserves the node to which the trigger is attached
  - *job:-:cancel* - cancels the job to which the trigger is attached
  - *reservation:-:cancel* - cancels the reservation to which the trigger is attached

  The specified object reserves or cancels itself once the event occurs. See **Internal Action** on page 747 for examples.

Requiring an object threshold for trigger execution

Context

Threshold triggers allow sites to configure triggers to launch based on internal scheduler statistics, such as generic metrics. For example, you might configure a trigger to warn the administrator when the percentage of nodes available is less than 25.

To configure a threshold trigger

1. Create a trigger. Set its **EType** to *threshold*. Configure the **AType**, **Action**, and **Threshold** attributes' values based on the valid thresholds per object listed in the table found in **Threshold triggers** on page 743.

```
NODECFG[node04] TRIGGER=EType=threshold,AType=exec,Action="$HOME/hightemp.py
$OID",Threshold=gmetric
```

2. Insert the gmetric name between brackets (such as `gmetric[temp]`). Provide a comparison operator. For valid options, see the comparison operators table.

3. Provide a number or string to match against the threshold.

```
NODECFG[node04] TRIGGER=EType=threshold,AType=exec,Action="$HOME/hightemp.py
$OID",Threshold=gmetric[TEMP]>70,RearmTime=5:00
```

Moab launches a script that warns the administrator when `node04`'s gmetric `temp` exceeds 70. Moab rearms the trigger five minutes after it fires.

Enabling job triggers

Context

By default common users cannot create most objects, and as a result, common users also cannot create triggers. The exception, however, is jobs. Because common users can create jobs and triggers generally run as root, additional security is necessary to ensure that not all users can create triggers. For this reason, job triggers are disabled by default.

Because triggers generally run as root, any user given the power to attach triggers has the power to run scripts and commands as root. It is recommended that you only enable job triggers on closed systems in which human users do not have access to directly submit jobs.

To give specific users permission to create job triggers, you must create a QoS, set the *trigger* flag, and add users to it.

To enable job triggers

1. In the `moab.cfg` file, create a QoS and set the *trigger* flag.

```
QOSCFG[triggerok]   QFLAGS=trigger
```

2. Add users to the QoS who should be allowed to add triggers to jobs.

```
USERCFG[joe]        QDEF=triggerok
```

User joe is added to the *triggerok* QoS, giving him both the power to create job triggers and root access to the machine.

Modifying a trigger

Context

You can modify a trigger at any time by updating its settings in the Moab configuration file (moab.cfg). This will update most triggers at the beginning of the next Moab iteration; however, modifying template triggers (configured using RSVPROFILE or JOBCFG) will not update the instances of the trigger that were attached to individual reservations or jobs on creation. The modification will only affect the triggers that the template attaches to future objects.

Any trigger with a specified name can be modified using the mschedctl -m command in the following format:

mschedctl -m trigger: *<triggerID><attr1>=<val1><attr2>=<val2>*

> Modifying triggers on the command line does not change their configuration in moab.cfg. Except for reservations that are checkpointed, changes made dynamically are lost when Moab restarts.

For example, the procedure below demonstrates how to modify the following trigger so that the offset is 10 minutes instead of 5 and so that Moab will attempt to fire the trigger up to 10 times if it fails. Assume your trigger currently looks like this:

```
NODECFG[DEFAULT] EType=fail,AType=exec,Action="/scripts/node_
fail.pl",Name=nodeFailTrig,Offset=00:05:00,MultiFire=TRUE,RearmTime=01:00:00
```

To modify a trigger

1. Type mschedctl -m into the command line and set off the trigger modification with trigger:<id>. Use the trigger's assigned ID or specified name to state which trigger will receive the modification.

```
> mschedctl -m trigger:nodeFailTrig
```

2. Type any changing attributes equal to the new value. Separate multiple modifications with a space between each *<attr>=<val>* pair. In this case, set the **Offset** and **MaxRetry** attributes the following way:

```
> mschedctl -m trigger:nodeFailTrig Offset=00:10:00 MaxRetry=10
```

The newly-specified attributes replace the original ones. Trigger nodeFailTrig now has an offset of 10 minutes and will try to fire a maximum of 10 times if it fails. The new trigger has the following attributes:

```
EType=fail,AType=exec,Action="/scripts/start_
rsv.pl",Name=nodeFailTrig,Offset=00:10:00,MultiFire=TRUE,RearmTime=01:00:00,MaxRetry
=10
```

Viewing a trigger

Context

Moab provides a list of triggers when you run the mdiag -T command. You can view a specific trigger by running mdiag -T in the following format:

```
mdiag -T [<triggerID>|<objectID>|<triggerName>|<objectType>]
```

To view a trigger

1. Type mdiag -T in the command line.
2. Specify either the trigger ID, the trigger name, the name of the object to which the trigger is attached, or the type of object to which the trigger is attached. For example, if you wanted to view information about a trigger with ID `trigger.34` and name `jobFailTrigger`, which is attached to job `job.493`, you could run any of the following commands:

```
> mdiag -T trigger.34

> mdiag -T job.493

> mdiag -T jobFailTrigger

> mdiag -T job
```

The output of the first command would provide basic information about `trigger.34`; the second command, information about all triggers attached to `job.493` that the user can access; the third command, basic information about `jobFailTrigger`; and the fourth command, basic information about all triggers attached to jobs that the user can access.

3. Optional: to view additional information about the trigger, run the same command with the `-v` flag specified after `-T`.

```
> mdiag -T -v job.493
```

This mode outputs information in multiple lines.

4. Optional: to view detailed information about all triggers available to you, use the mdiag -T -v command. This outputs all triggers available to the user in a single line for each trigger. It provides additional state information about triggers, including reasons triggers are currently blocked.

```
> mdiag -T -v
```

Checkpointing a trigger

Context

Checkpointing is the process of saving state information when Moab is shut down. In general, triggers defined in the `moab.cfg` file are not checkpointed but are recreated when Moab starts. The exception is the JOBCFG parameter, which attaches triggers to jobs as they are created. There are two cases in which you may want to tell Moab to checkpoint a trigger:

- If a trigger is defined in the `moab.cfg` file but was created at the command line
- When creating a trigger using the **mschedctl** on page 268 command

To checkpoint a trigger

1. Locate the trigger to be checkpointed in the `moab.cfg` file, create one on the command line, or modify a trigger dynamically (See **Modifying a trigger** on page 735 for more information). Attach the *checkpoint* flag using the **FLAGS** attribute. For more information about flags, see **Flags** on page 750.

```
FLAGS=checkpoint
```

2. If you are working in the configuration file, save the changes. Moab will now checkpoint your trigger.

References

Job triggers

For security reasons, job triggers are disabled by default. They must be enabled in order to successfully attach triggers to jobs (See **Enabling job triggers** on page 734 for more information.).

Triggers attached to jobs follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. Jobs, like other objects, have a unique set of trigger rules. The table below details the methods, options, and other notable details associated with attaching triggers to jobs.

Creation methods

Method	Format	Example
Command line on job creation: msub -l	`msub <jobName> -l 'trig=<trigSpec>'` Attributes are delimited by backslash ampersand (\&).	`> msub my.job -l 'trig=EType=create\&AType=exec\&Action="/jobs/my_trigger.pl"\&Offset=10:00'`
Command line on existing job: mschedctl -c	`mschedctl -c trigger <trigSpec> -o job:<jobID>`	`> mschedctl -c trigger EType=end,AType=mail,Action="Job moab.54 has ended" -o job:moab.54`
Job template in `moab.cfg` : JOBCFG	`JOBCFG[<templateName>] TRIGGER=<trigSpec>`	`JOBCFG[vmcreate] TRIGGER=,EType=end,AType=exec,Action="/tmp/jobEnd.sh"`

Valid event types

- **cancel** on page 747
- **checkpoint** on page 747
- **create** on page 747
- **end** on page 747
- **hold** on page 748
- **modify** on page 748
- **preempt** on page 748
- **start** on page 748

Valid action types

- changeparam
- exec
- internal
- mail

Mail recipient

The job's owner

See **Using a trigger to send email** on page 731 for more information.

Node triggers

Triggers attached to nodes follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. Nodes, like the other objects, have a unique set of trigger rules. The table below details the methods, options, and other notable details that come with attaching triggers to nodes.

Creation methods

Method	Format	Example
Command line on existing node: mschedctl - c	mschedctl -c trigger *<trigSpec>* -o node:*<nodeID>*	`> mschedctl -c trigger EType=fail,AType=exec,Action="/tmp/nodeFailure.sh" -o node:node01`

Method	Format	Example
Node configuration in `moab.cfg`: NODECFG	`NODECFG [<name>] TRIGGER= <trigSpec>`	`NODECFG[node04] TRIGGER=EType=threshold,AType=exec,Action="$HOME/hightemp.py $OID",Threshold=gmetric[TEMP]>70`

Valid event types

- **create** on page 747
- **discover** on page 747
- **end** on page 747
- **fail** on page 748
- **standing** on page 748
- **threshold** on page 748

Valid action types

- changeparam
- exec
- internal
- mail

Thresholds

Node threshold settings	
Valid ETypes	threshold
Valid Threshold types	gmetric

Mail recipient

The user listed first in ADMINCFG[1] (usually `root`)

See **Using a trigger to send email** on page 731 for more information.

Reservation triggers

Triggers attached to reservations follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. Reservations, like the other objects, have a unique set of trigger rules. The table below details the methods, options, and other notable details that come with attaching triggers to reservations.

Creation methods

Method	Format	Example
Command line on reservation creation: mrsvctl -T	`mrsvctl -c -h <host-list> -T <trigSpec>`	`> mrsvctl -c -h node01 -T EType=start,AType=exec, Action="/scripts/node_start.pl"`
Command line on existing reservation: mschedctl - c	`mschedctl -c trigger <trigSpec> -o rsv:<rsvID>`	`> mschedctl -c trigger EType=modify,AType=mail,Action="Reservation system.4 has been modified" -o rsv:system.4`
Standing reservation configuration in `moab.cfg` : SRCFG	`SRCFG [<name>] TRIGGER= <trigSpec>`	`SRCFG[Mail2] TRIGGER=EType=start,Offset=200,AType=exec,Action="/tmp/email.sh"`
Reservation template in `moab.cfg` : RSVPROFILE	`RSVPROFILE [<name>] TRIGGER= <trigSpec>`	`RSVPROFILE[rsvtest] TRIGGER=EType=cancel,AType=exec,Action="$HOME/logdate.pl TEST CANCEL $VPCHOSTLIST $OID $HOSTLIST $ACTIVE"`

Valid event types

- **create** on page 747
- **end** on page 747
- **modify** on page 748
- **standing** on page 748
- **start** on page 748
- **threshold** on page 748

Valid action types

- cancel
- changeparam
- exec
- internal
- jobpreempt
- mail

Thresholds

Node threshold settings	
Valid ETypes	threshold
Valid Threshold types	usage

Mail recipient

The owner of the reservation. If the owner is unknown or not a user, the first user listed first in ADMINCFG (usually `root`).

See **Using a trigger to send email** on page 731 for more information.

Resource manager triggers

Triggers attached to the resource manager follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. The resource manager, like other objects, has a unique set of trigger rules. The table below details the methods, options, and other notable details that come with attaching triggers to RMs.

Creation methods

Method	Format	Example
Command line on existing RM: mschedctl -c	`mschedctl -c trigger <trigSpec> -o rm:<rmID>`	`> mschedctl -c trigger EType=start,AType=exec,Action="/tmp/rmStart.sh" -o rm:torque`

Method	Format	Example
RM configuration in `moab.cfg`: RMCFG	RMCFG [<*name*>] TRIGGER= <*trigSpec*>	`RMCFG[base] TRIGGER=EType=fail,AType=exec,Action="/opt/moab/tools/diagnose_rm.pl $OID"`

Valid event types

- **fail** on page 748
- **threshold** on page 748

Valid action types

- changeparam
- exec
- internal

Scheduler triggers

Triggers attached to the scheduler follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. The scheduler, like the other objects, has a unique set of trigger rules. The table below details the methods, options, and other notable details associated with attaching triggers to the scheduler.

Creation methods

Method	Format	Example
Command line on existing scheduler: mschedctl - c	mschedctl -c trigger <*trigSpec*> -o sched:<*schedID*>	`> mschedctl -c trigger EType=end,AType=exec,Action="/tmp/startRsvs.sh" -o sched:moab`
Scheduler configuration in `moab.cfg` : SCHEDCFG	SCHEDCFG[<*name*>] TRIGGER=<*trigSpec*>	`SCHEDCFG[MyCluster] TRIGGER=EType=fail,AType=mail,Action="scheduler failure detected on $TIME",RearmTime=15:00`

Valid event types

- **create** on page 747
- **end** on page 747
- **fail** on page 748
- **modify** on page 748
- **standing** on page 748
- **start** on page 748

Valid action types

- changeparam
- exec
- internal
- mail

Mail recipient

The user listed first in ADMINCFG (usually `root`)

See **Using a trigger to send email** on page 731 for more information.

Threshold triggers

The following table identifies the object event, and usage types with which the threshold event/action type feature works.

Object type	Event Type	Usage types
Node	Threshold	gmetric
Reservation	Threshold	usage

The following table defines each of the usage types:

Usage type	Description
gmetric	Generic performance metrics configured in Moab (See Enabling Generic Metrics for more information).
usage	The percentage of the resource being used (not idle).

The following table defines each of the threshold trigger comparison operators:

Comparison operator	Value
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to

Examples

Example 3-150: Reservation usage threshold

```
SRCFG[res1] TRIGGER=EType=threshold,AType=mail,Action="More than 75% of reservation
res1 is being used",Threshold=usage>75,FailOffset=1:00
```

When more than 75% of the reservation has been in use for at least a minute, Moab fires a trigger to notify the primary user.

Trigger components

Required trigger components

AType

Action type	Description
cancel	Cancels the object
changeparam	Causes Moab to give a parameter to a new value
exec	Launches an external program or script on the command line when the dependencies are fulfilled. See **Using a trigger to execute a script** on page 733 for more information.
internal	Modifies Moab without using the command line. See **Using a trigger to perform internal Moab actions** on page 733 for more information.

Action type	Description
jobpreempt	Indicates the preempt policy to apply to all jobs currently allocated resources assigned to the trigger's parent reservation
mail	Causes Moab to send mail. See **Using a trigger to send email** on page 731 for more information.

Action

Cancel Action	
Format	NONE
Description	Indicates that Moab should cancel the reservation when the event occurs. No action should be specified.
Example	Etype=threshold,Threshold=usage<10,FailOffset=1:00,AType=cancel *When less than 10% of the reservation has been in use for a minute, Moab cancels it.*

Changeparam Action	
Format	Action="*<STRING>*"
Description	Specifies the parameter to change and its new value (using the same syntax and behavior as the **changeparam** on page 367 command)
Example	Atype=changeparam,Action="JOBCPURGETIME 02:00:00" *Moab maintains detailed job information for two hours after a job has completed.*

Jobpreempt Action	
Format	Action="cancel|checkpoint|requeue|suspend"
Description	Signifies PREEMPTPOLICY to apply to jobs that are running on allocated resources

Jobpreempt Action	
Example	`RSVPROFILE[adm1] TRIGGER=EType=start,Offset=-240,AType=jobpreempt,Action="cancel"` *40 minutes after the reservation* `adm1` *starts, all jobs using the reservation's resources adopt a* **PREEMPTPOLICY** *of* cancel*.*

Mail Action	
Format	`Action="<MESSAGE>"`
Descrip-tion	When **AType**=*mail*, the **Action** parameter contains the message body of the email. This can be configured to include certain variables. See **Using a trigger to send email** on page 731 for details. Mail triggers can be configured to launch for node failures, reservation creation or release, scheduler failures, and even job events. In this way, site administrators can keep track of scheduler events through email. The email comes from *moabadmin*, has a subject of *moab update*, and has a body of whatever you specified in the **Action** attribute. The recipient list depends on the type of object the trigger is attached to. • **Node** - The primary user (first listed in ADMINCFG[1]), typically `root` • **Scheduler** - The primary user • **Job** - The user who owns the job • **Reservation** - The primary user
Example	`NODECFG[DEFAULT] TRIGGER=EType=fail,AType=mail,Action="node $OID will failed.",Offset=05:00:00` *This example sends an email to the primary administrator informing him/her that the node (including the node ID) has failed.*

Exec Action	
Format	`Action="<script>"`
Description	Exec triggers will launch an external program or script when their dependencies are fulfilled. The following example will submit `job.cmd` and then execute `monitor.pl` three minutes after the job is started. See **Using a trigger to execute a script** on page 733 for more information.
Example	`> msub -l trig=EType=start\&AType=exec\&Action="/tmp/monitor.pl" job.cmd\&Offset=03:00`

Internal Action	
Format	`Action="<objectType>:-:<cancel|reserve>"`
Description	A couple different actions are valid depending on what type of object the internal trigger is acting upon. The following list shows the available actions: • Reserve a node • Cancel a job • Cancel a reservation See **Using a trigger to perform internal Moab actions** on page 733 for more information.
Example	`NODECFG[node01] TRIGGER=EType=start,AType=internal,Action="node:-:reserve"` *When* `node01` *starts, it becomes a reservation.* `> msub moab.3 -l 'trig=EType=fail\&AType=internal\&Action="job:-:cancel"` *If* `moab.3` *fails, Moab cancels it.* `> mrsvctl -c -a user==joe -h node50 -T EType=start,AType=internal,Action="reservation:-:cancel",Offset=10:00` *User* `joe`*'s jobs are given a ten-minute window to start, then the reservation cancels.*

EType

Event type	Description
cancel	The event is triggered when the parent object is either canceled or deleted.
checkpoint	Triggers fire when the job is checkpointed. *checkpoint* triggers can only be attached to jobs.
create	Triggers fire when the parent object is created. *create* triggers can be attached to nodes, jobs, reservations, and the scheduler (when attached to the scheduler, triggers fire when Moab starts).
discover	Triggers fire when the node is loaded from a resource manager and Moab cannot recorgnize it nor find it in the checkpoint file.
end	Triggers fire when the parent object ends. *end* triggers can be attached to nodes, jobs, reservations, and the scheduler (when attached to the scheduler, triggers fire when Moab shuts down).

Event type	Description
fail	Triggers fire when the resource manager is in a corrupt or down state for longer than the configured fail time, or when Moab detects a corruption in a node's reservation table. *fail* triggers can be attached to jobs, nodes, resource managers, and the scheduler.
hold	Triggers fire when the job is put on hold. *hold* triggers can only be attached to jobs.
modify	Triggers fire when the parent object is modified. *modify* triggers can be attached to jobs and reservations
preempt	Triggers fire when the job is preempted. *preempt* triggers can only be attached to jobs.
standing	Triggers fire multiple times based on a certain period. They can be used with Period and Offset attributes. *standing* triggers can be attached to nodes and the scheduler.
start	Triggers fire when the parent object or Moab starts. *start* triggers can be attached to jobs, reservations, resource managers, and the scheduler (when Moab starts and at the beginning of Moab's first iteration).
threshold	Triggers fire when a threshold, such as usage or a gmetric comparison, is true. *threshold* triggers can be attached to nodes and reservations. Triggers with **EType**s set to *threshold* must include the Threshold attribute.

Event-modifying trigger components

The following trigger attributes modify the event that causes the trigger to fire.

RearmTime	
Possible Values	`[[HH:]MM:]SS`
Description	The amount of time that must pass before a trigger can fire again. **RearmTime** is enforced from the trigger event time.
Usage Notes	---

Offset	
Possible Values	`[-][[HH:]MM:]SS`

Offset	
Description	The relative time offset from event when trigger can fire
Usage Notes	• Only end triggers can have a negative value for **Offset**. • **Offset** cannot be used with cancel.

Period	
Possible Values	*Minute*, *Hour*, *Day*, *Week*, *Month*, *Infinity*
Description	The period at which the trigger will regularly fire
Usage Notes	---

Threshold	
Possible Values	`Threshold={<metric>[<metricName>]}{> >= < <= ==}<FLOAT>` Where *<metric>* is one of: • **gmetric** • **usage**
Description	When the object meets, drops below, or increases past the configured **Threshold**, the trigger will fire.
Usage Notes	Threshold triggers allow sites to configure triggers to launch based on internal scheduler statistics, such as the usage of a reservation.

FailOffset	
Possible Values	`[[HH:]MM:]SS`
Description	The time that the threshold condition must exist before the trigger fires
Usage Notes	Use with fail triggers to avoid transient triggers.

Action-modifying trigger components

Flags	
Possible Values	`Flags=<flag>[:<flag>]` or `Flags=[<flag>][[<flag>]]` *attacherror* - If the trigger outputs anything to stderr, Moab attaches it as a message to the trigger object. *cleanup* - If the trigger is still running when the parent object completes or is canceled, Moab kills the trigger. *checkpoint* - Moab always checkpoints this trigger. For more information, see **Checkpointing a trigger** on page 736. *objectxmlstdin* - Trigger passes its parent's object XML information into the trigger's stdin. This only works for exec triggers with reservation type parents. *resetonmodify* - The trigger resets if its object is modified, even if **RearmTime** is not set. *user* - The trigger executes under the user ID of the object's owner. If the parent object is the scheduler, you may explicitly specify the user using the format `user+<username>`. For example: `Flags=user+john`.
Description	Specifies various trigger behaviors and actions
Usage Notes	When specifying multiple flags, each flag can be delimited by colons (:) or with square brackets; for example: `Flags=[user][cleanup]` or `Flags=user:cleanup`

BlockTime	
Possible Values	`[[HH:]MM:]SS`
Description	The amount of time Moab will suspend normal operation to wait for trigger execution to finish
Usage Notes	Use caution; Moab will completely stop normal operation until **BlockTime** expires.

ExpireTime	
Possible Values	*<INTEGER>*
Description	The time at which trigger should be terminated if it has not already been activated
Usage Notes	---

Timeout	
Possible Values	`[+|-][[HH:]MM:]SS`
Description	The time allotted to this trigger before it is marked as unsuccessful and its process (if any) killed
Usage Notes	---

MaxRetry	
Possible Values	`MaxRetry=`*<INTEGER>*
Description	The number of times **Action** will be attempted before the trigger is designated a failure
Usage Notes	If **Action** fails, the trigger will restart immediately (up to **MaxRetry** times). If it fails more than **MaxRetry** times, the trigger has failed. This restart ignores **FailOffset** and **RearmTime**.

Organizational trigger components

Name	
Possible Values	`Name=`*<STRING>*
Description	Name of the trigger
Usage Notes	Because Moab uses its own internal ID to distinguish triggers, the **Name** need not be unique. Only the first 16 characters of **Name** are stored by Moab.

Description	
Possible Values	`Description=`*<STRING>*
Description	Description of the trigger
Usage Notes	---

Trigger exit codes

By default Moab considers any non-zero exit code as a failure and marks the trigger as having failed. If a trigger is killed by a signal outside of Moab, Moab treats the signal as the exit code and (in almost all cases) marks the trigger as having failed. Only exec triggers that exit with an exit code of 0 are marked as successful.

Node maintenance example

Example scenario

An administrator wants to create the following setup in Moab:

When a node's temperature exceeds 34°C, Moab reserves it. If the temperature increases to more than 40°C, Moab requeues all jobs on the node. If the node's temperature exceeds 50°C, Moab shuts it down. Moab removes the node's reservation and unsets the variables when the node cools to less than 25°C. The administrator wants to receive an email whenever any of these events occur.

The first trigger reserves the node when its reported temperature exceeds 34°C. Note that the gmetric name in the trigger must match the name of the configured gmetric exactly, including its case (See **Enabling Generic Metrics** on page 575 for more information.).

```
NODECFG[DEFAULT] TRIGGER=Description="ThresholdA",EType=threshold,Threshold=gmetric
[temp]>34,AType=internal,Action="node:-:reserve",RearmTime=30,Offset=2:00,Sets=temp_
rsv
```

The administrator wants the trigger to fire any time a node overheats, so it must be rearmable. It also needs to specify that the node must be over 34°C for at least two minutes for Moab to reserve it. If the trigger succeeds, it will set a variable to be received by the next trigger in order to make them sequential.

The administrator wants to know when this trigger has fired, so another trigger will send an email once the first trigger has fired and the `temp_rsv` variable is set. This one does so via a script:

```
NODECFG[DEFAULT] Trigger=Description="Email on
Reservation",EType=start,AType=exec,Action="$TOOLSDIR/node_temp_emailReserve.pl
$OID",RearmTime=3:00,Requires=temp_rsv
```

The second threshold trigger requeues the node's jobs if the node exceeds 40°C and the `temp_rsv` variable is set. It uses a script to do so. It sets `node_evac` variable when it fires, regardless of whether it succeeds or fails.

```
NODECFG[DEFAULT] Trigger=Description="Threshold B",EType=threshold,Threshold=gmetric
[temp]>40,Atype=exec,Action="$TOOLSDIR/node_evacuate.pl
$OID",RearmTime=3:00,requires=temp_rsv,Sets=node_evac,!node_evac
```

The administrator wants another email to inform him that the node is still overheating and has been evacuated. Another email trigger fires once it receives the `node_evac` variable.

```
NODECFG[DEFAULT] Trigger=Description="Email on
Evacuation",EType=start,AType=exec,Action="$TOOLSDIR/node_temp_emailEvac.pl
$OID",RearmTime=3:00,Requires=node_evac
```

The third threshold trigger uses a script to shut down the node if the temp gmetric exceeds 50 and the `node_evac` variable is set. It sets a `node_shutdown` variable to be received by the notification email.

```
NODECFG[DEFAULT TRIGGER=Description="Threshold C",EType=threshold,Threshold=gmetric
[temp]>50,AType=exec,Action="$TOOLSDIR/node_shutdown.pl
$OID",RearmTime=3:00,Requires=node_evac,Sets=node_shutdown

NODECFG[DEFAULT] Trigger=Description="Email on
Shutdown",EType=start,AType=exec,Action="$TOOLSDIR/node_temp_emailShutdown.pl
$OID",RearmTime=3:00,Requires=node_shutdown
```

The final trigger removes the reservation and unsets the variables once the node's `temp` gmetric is less than 25.

```
NODECFG[DEFAULT] Trigger=Description="Remove
Reservation",EType=threshold,Threshold=gmetric[temp]
<25,AType=exec,Action="opt/moab/bin/mrsvctl -r r:$OID",RearmTime=3:00,Requires=temp_
rsv,unsets=temp_rsv.node_evac.node_shutdown
```

Environment creation example

Example scenario

An administrator wants to create the following setup in Moab:

If a user requests an environment, she must have the permission of her two managers and the administrator. If all three approve, then the environment builds. The user is sent email messages informing her of the environment's end date in case she would like an extension. These are sent 7, 3, and 1 days prior to the environment's ending.

The administrator wants to require his and the managers' approval of any modifications the user makes to her environment so that it cannot be extended without consent.

The first trigger requests manager and administrator approval in response to the user's environment request. So in the event of a reservation's creation, a script is used to send messages to the administrator and manager. The internal variable `OWNER` is used to indicate to the recipients (via the script) which user is requesting the environment.

```
RSVPROFILE[envSetup] TRIGGER=EType=create,AType=exec,Action="envRequest.sh $OWNER"
```

The managers and administrator use an external program to approve or reject the request. On approval, a variable is sent back to Moab (to the reservation specifically). Once all three variables are set, the environment can start. In this example, the variables are called `approval1`, `approval2`, and `approval3`.

```
RSVPROFILE[envSetup]
TRIGGER=EType=start,AType=exec,Action="buildScript",Requires=approval1.approval2.appro
val3
```

As it is configured now, the reservation will continue to reserve the requested resources regardless of whether all three approvals are given. So, in case approval is not given, the next trigger cancels the reservation 7 days after its creation if the three variables are not set.

```
RSVPROFILE[envSetup]
TRIGGER=EType=create,Offset=7:00:00,AType=internal,Action="rsv:-:cancel",Requires=!app
roval1.!approval2.!approval3
```

Every remaining trigger in this series is meant to fire for an approved environment and must require the approval variables. Otherwise these notifications would be sent to users who do not have the environment they requested. The next triggers must be rearmable so that it can fire again if necessary; however, they should be set to just over the amount of time left on the reservation so that it doesn't fire again for the same environment. The notification triggers use the **Offset** attribute to fire at the administrator's requested times (7, 3, and 1 day(s) prior to the environment's end).

```
RSVPROFILE[envSetup] TRIGGER=EType=end,Offset=-
7:00:00,AType=exec,Action="weekNotification.sh",RearmTime=7:00:00:02,Requires=approval
1.approval2.approval3

RSVPROFILE[envSetup] TRIGGER=EType=end,Offset=-
3:00:00,AType=exec,Action="3dayNotification.sh",RearmTime=3:00:00:02,Requires=approval
1.approval2.approval3

RSVPROFILE[envSetup] TRIGGER=EType=end,Offset=-
1:00:00,AType=exec,Action="dayNotification.sh",RearmTime=1:00:00:02,Requires=approval1
.approval2.approval3
```

The next trigger requests administrator and manager approval when the environment is modified. The problem is that the trigger must be rearmable in case of multiple modifications and each time the **RearmTime** on page 748 is reached, Moab will fire the trigger based on the *first* instance of modification. To resolve this issue, this modification trigger requires a *modify* variable. When the reservation is modified, the *modify* variable is set.

```
RSVPROFILE[envSetup]
TRIGGER=EType=modify,AType=exec,Action="modify.sh",RearmTime=1:00:00,Requires=approval
1.approval2.approval3.!modify,Sets=modify
RSVPROFILE[envSetup]
TRIGGER=EType=modify,AType=exec,Action="modificationRequest.sh",RearmTime=5:00,Require
s=approval1.approval2.approval3.modify,Unsets=modify
```

The final triggers notify the user of the end of the environment.

```
RSVPROFILE[envSetup]
TRIGGER=EType=end,AType=exec,Action="end.sh",Requires=approval1.approval2.approval3
```

The same trigger is repeated for the *cancel* **EType** in case the environment ends unexpectedly.

```
RSVPROFILE[envSetup]
TRIGGER=EType=cancel,AType=exec,Action="end.sh",Requires=approval1.approval2.approval3
```

Trigger variables

About trigger variables

Trigger variables are pieces of information that pass from trigger to trigger. They allow triggers to fire based on another trigger's behavior, state, and/or output. A variable can be a required condition for a trigger to fire; for instance, a trigger might be set to launch when a reservation starts, but only if it has received a variable from another trigger indicating that a specific node has started first. Variables give greater flexibility and power to a site administrator who wants to automate certain tasks and system behaviors.

Variables can be used to define under what circumstances the trigger will fire. Many Moab objects have their own variables and each object's variable name space is unique. Triggers can use their own variables or the variables attached to their parent objects. A trigger's variable name space is limited to itself and its parent object. Variables do not have to be unique across all objects.

How-to's

- **Setting and receiving trigger variables** on page 755
- **Externally injecting variables into job triggers** on page 756
- **Exporting variables to parent objects** on page 756
- **Requiring variables from generations of parent objects** on page 757
- **Requesting name space variables** on page 757

References

- **Dependency trigger components** on page 758
- **Internal variables** on page 759

How-to's

Setting and receiving trigger variables

Context

Following is an example of how comparative dependencies can be expressed when creating a trigger.

To set and require variables

1. Create a trigger.

```
EType=start,AType=exec,Action="/tmp/trigger1.sh"
```

2. Use the **Sets** attribute to set a variable if the trigger succeeds. You can precede the variable with "!" to indicate that the variable should be set if the trigger fails. You can specify more than one variable by separating them with a period.

```
AType=exec,Action="/tmp/trigger1.sh",EType=start,Sets=!Var1.Var2
```

The trigger sets variable `Var2` when it succeeds and variable `Var1` when it fails.

3. Set up the recipient trigger(s). Use the Requires attribute to receive the variable(s). Note that preceding the variable with "!" means that the variable must not be set in order for the trigger to fire.

```
AType=exec,Action="/tmp/trigger1.sh",EType=start,Sets=!Var1.Var2
AType=exec,Action="/tmp/trigger2.sh",EType=start,Requires=Var1
AType=exec,Action="/tmp/trigger3.sh",EType=start,Requires=Var2
```

The second trigger will launch if `Var1` has been set (the first trigger failed), and the third trigger will launch if `Var2` is set (the first trigger succeeded).

4. Refine the requirement with comparisons.
   a. Use the following format:
      `<varID>[:<type>[:<varVal>]]`
   b. Change *<varID>* to the variable name.
   c. Use any of the comparisons found on the **Trigger variable comparison types** on page 758 page in place of *<type>*:
   d. Set the value that the variable will be compared against.

```
AType=exec,Action="/tmp/trigger2.sh",EType=start,Requires=Var1:eq:45
AType=exec,Action="/tmp/trigger3.sh",EType=start,Requires=Var2:ne:failure1
```

The first trigger fires if `Var1` exists and has a value of 45. The second trigger fires if `Var2` does not have a string value of `failure1`.

Externally injecting variables into job triggers

Context

Job triggers are able to see the variables in the job object to which it is attached. This means that, for triggers that are attached to job objects, another method for supplying variables exists. Updating the job object's variables effectively updates the variable for the trigger.

To externally inject variables into job triggers

- Use the mjobctl -m command to set a variable to attach to a job.

```
> mjobctl -m var=Flag1=TRUE 1664
```

The variable *Flag1* is set. This will be available to any trigger attached to job `1664`.

Exporting variables to parent objects

To export variables to parent objects

1. When setting a variable, indicate that the variable is to be exported to the parent object by using a caret (^).

```
AType=exec,Action="/tmp/trigger1.sh",EType=start,Sets=Var1.!^Var2
Atype=exec,Action="/tmp/trigger2.sh",EType=start,Requires=Var1
AType=exec,Action="/tmp/trigger3.sh",EType=start,Requires=Var2
```

   `Var2` is exported to the parent object if the trigger fails. It can be used by job and reservation triggers at the same level or by parent objects.

2. Optional: if running a script, you can set a variable as a string to pass up to the parent object.
   a. Set the variable to pass up to the parent object with the caret (^). Use the *exec* **AType** to run a script.

```
AType=exec,Action="/tmp/trigger.sh",EType=start,Sets=^Var1
```

   The trigger sets *Var1* when it completes successfully. Because the trigger launches a script, a string value can be set for *Var1*.

b. Declare the variable's string value on its own line in the trigger stdout.

```
EXITCODE=15
Var1=linux
```

Var1 has the value of `linux` and is passed up to the parent object. This is useful in workflows in which a trigger may depend on the value given by a previous trigger.

To return multiple variables, simply print out one per line.

Requiring variables from generations of parent objects

Context

By default, triggers look for variables to fulfill dependencies in the object to which they are directly attached. If they are attached to a job object, they will also look in the job group, if defined. However, it is not uncommon for objects to have multiple generations of parent objects. If the desired behavior is to search through all parent objects, do the following task.

To require variables from generations of parent objects

- Set the **Requires** attribute in the trigger to the required variable, preceded by a caret (^).

```
EType=start,AType=exec,Action="/tmp/trigger2.sh",Requires=^Var1
```

The trigger searches through the parent objects in which it resides for the variable Var1.

Requesting name space variables

To request a name space variable in a trigger

1. Configure the trigger. If it is attached to a generic system job, verify that it meets all the generic system job trigger requirements.
2. Create an argument list in the `Action` attribute (after the script path and before the closing quotes) and request the desired variable with an asterisk (*) in place of the name space.

   ```
   ...Action="$HOME/myTrig.py $*.IPAddr"...
   ```

   Each applicable name space variable is added to the argument list in the format `<varName>=<val>`.

   For instance, the example above would cause the script to run the following way:

   ```
   > myTrig.py vc1.IPAddr=/tmp/dir1 vc2.IPAddr=/tmp/dir2 vc4.IPAddr=/tmp/dir3
   ```

   Any other arguments provided here without name spaces will not change.
3. Filter which name spaces are passed down to a job trigger by setting `trigns` when you submit the job. Its value is a comma-delimited list of the desired name spaces.

   ```
   msub -l ... -W x="trigns=vc2,vc4"
   ```

   If the new job is applied to the example in step 2, the script's arguments include *vc2.IPAddr* and *vc4.Addr* and exclude *vc1.IPAddr*. The script runs as follows:

```
> myTrig.py vc2.IPAddr=/tmp/dir1 vc4.IPAddr=/tmp/dir2
```

References

Dependency trigger components

Sets	
Possible values	'.' delimited string
Description	Variable values this trigger sets upon success or failure
Usage notes	Preceding the string with an exclamation mark (!) indicates this variable is set upon trigger failure. Preceding the string with a caret (^) indicates this variable is to be exported to the parent object when the trigger completes and satisfies all its set conditions. Used in conjunction with **Requires** on page 758 to create trigger dependencies.

Unsets	
Possible values	'.' delimited string
Description	Variable this trigger destroys upon success or failure.
Usage notes	Preceding the string with an exclamation mark (!) indicates this variable is unset upon trigger failure. Used in conjunction with **Requires** on page 758 to create trigger dependencies.

Requires	
Possible values	'.' delimited string
Description	Variables this trigger requires to be set or not set before it will fire.
Usage notes	Preceding the string with an exclamation mark (!) indicates this variable must not be set. Preceding the string with a caret (^) indicates that the variable may come from a parent object (See **Requiring variables from generations of parent objects** on page 757 for more information.). Used in conjunction with **Sets** on page 758 to create trigger dependencies.

Trigger variable comparison types

The following table describes the valid types of comparisons you can use to express the relationship of a trigger variable to its value:

Type	Comparison	Notes
set	is set (exists)	Default
notset	not set (does not exist)	Same as specifying '!' before a variable
eq	equals	
ne	not equal	
gt	greater than	Integer values only
lt	less than	Integer values only
ge	greater than or equal to	Integer values only
le	less than or equal to	Integer values only

Internal variables

Several internal variables are available for use in trigger scripts. These can be accessed using `$<varName>`.

Internal Variables	
ETYPE	The type of event that signals that the trigger can fire. ETYPE values include cancel, checkpoint, create, end, fail, hold, migrate, preempt, standing, start, and threshold.
OID	The name of the object to which the trigger was attached
OTYPE	The type of object to which the trigger is attached; can be rsv, job, node, vm, or sched
OWNERMAIL	A variable that is populated only if the trigger's parent object has a user associated with it and that user has an email address associated with it
TIME	The time of the trigger launch in the following format: `Wed Mar 10 12:35:12 2012`
USER	The user (when applicable)

Object-specific internal variables

Job Variables	
MASTERHOST	The primary node for the job
HOSTLIST	The entire hostlist of the job

Reservation Variables	
HOSTLIST	The entire hostlist for the reservation
OBJECTXML	The XML representation of an object output is the same that is generated by mdiag -r --xml
OS	The operating system on the first node of the reservation
OWNER	The owner of the reservation

Example 3-151: Internal variable example

```
AType=exec,Action="/tmp/trigger.sh $OID $HOSTLIST",EType=start
```

The object ID (`$OID`) and hostlist (`$HOSTLIST`) will be passed to `/tmp/trigger.sh` as command line arguments when the trigger executes the script. The script can then process this information as needed.

Miscellaneous

User Feedback Overview

The Feedback facility allows a site administrator to provide job performance information to users at job completion time. When a job completes, the program pointed to by the FEEDBACKPROGRAM parameter is called with a number of command line arguments. The site administrator is responsible for creating a program capable of processing and acting upon the contents of the command line. The command line arguments passed are as follows:

1. job id
2. user name
3. user email
4. final job state
5. QoS requested
6. epoch time job was submitted
7. epoch time job started
8. epoch time job completed
9. job XFactor
10. job wallclock limit
11. processors requested
12. memory requested
13. average per task cpu load
14. maximum per task cpu load
15. average per task memory usage
16. maximum per task memory usage
17. messages associated with the job
18. hostlist (comma-delimited)

For many sites, the feedback script is useful as a means of letting users know the accuracy of their wallclock limit estimate, as well as the CPU efficiency, and memory usage pattern of their job. The feedback script may be used as a mechanism to do any of the following:

- email users regarding statistics of all completed jobs
- email users only when certain criteria are met (such as "Job 14991 has just completed which requested 128 MB of memory per task. During execution, it used 253 MB of memory per task potentially conflicting with other jobs. Please improve your resource usage estimates in future jobs.")
- update system databases
- take system actions based on job completion statistics

> Some of these fields may be set to zero if the underlying OS/resource manager does not support the necessary data collection.

Example 3-152:

```
FEEDBACKPROGAM  /opt/moab/tools/fb.pl
```

Enabling High Availability Features

Contact Adaptive Computing before attempting to implement any type of high availability.

High Availability Overview

High availability allows Moab to run on two different machines: a primary and secondary server. The configuration method to achieve this behavior takes advantage of a networked file system to configure two Moab servers with only one operating at a time.

If you use a shared file system for high availability and Moab is configured to use a database, Moab must be an ODBC build, not SQLite.

When configured to run on a networked file system — any networked file system that supports file locking is supported — the first Moab server that starts locks a particular file. The second Moab server waits on that lock and only begins scheduling when it gains control of the lock on the file. This method achieves near instantaneous turnover between failures and eliminates the need for two Moab servers to synchronize information periodically as the two Moab servers access the same database/checkpoint file.

As Moab uses timestamping in the lock file to implement high availability, the clocks on both servers require synchronization; all machines in a cluster must be synchronized to the same time server.

Moab high availability and TORQUE high availability operate independently of each other. If a job is submitted with msub and the primary Moab server is down, msub tries to connect to the fallback Moab server. Once the job is given to TORQUE, if TORQUE can't connect to the primary pbs_server, it tries to connect to the fallback pbs_server. For example:

A job is submitted with msub, but Moab is down on *server01*, so msub contacts Moab running on *server02*.

A job is submitted with msub and Moab hands it off to TORQUE, but pbs_server is down on *server01*, so qsub contacts pbs_server running on *server02*.

When you shut down or restart Moab on both servers, you must run the command twice. A single shutdown (mschedctl -k) or restart (mschedctl -R) command will go to the primary server and kill it, causing the secondary server to fall back and start operating. To kill the secondary server, resubmit the command.

Do not use anything but a plain simple NFS fileshare that is not used by anybody or anything else (i.e., only Moab can use the fileshare).

> Do not use any general-purpose NAS, do not use any parallel file system, and do not use company-wide shared infrastructure to set up Moab high availability using "native" high availability.

Configuring High Availability on a Networked File System

Because the two Moab servers access the same files, configuration is only required in the `moab.cfg` file. The two hosts that run Moab must be configured with the **SERVER** and **FBSERVER** parameters. File lock is turned on using the FLAGS=filelockha flag. Specify the lock file with the **HALOCKFILE** parameter. The following example illustrates a possible configuration:

```
SCHEDCFG[Moab]  SERVER=host1:42559
SCHEDCFG[Moab]  FBSERVER=host2
SCHEDCFG[Moab]  FLAGS=filelockha
SCHEDCFG[Moab]  HALOCKFILE=/opt/moab/.moab_lock
```

Use the HALOCKUPDATETIME parameter to specify how frequently the primary server updates the timestamp on the lock file. Use the HALOCKCHECKTIME parameter to specify how frequently the secondary server checks the timestamp on the lock file.

```
HALOCKCHECKTIME 9
HALOCKUPDATETIME 3
```

In the preceding example, the secondary server checks the lock file for updates every 9 seconds. The **HALOCKUPDATETIME** *parameter is set to 3 seconds, permitting the primary server three opportunities to update the timestamp for each time the secondary server checks the timestamp on the lock file.*

> **FBSERVER** does not take a port number. The primary server's port is used for both the primary server and the fallback server.

Confirming High Availability on a Networked File System

Administrators can run the mdiag -S -v command to view which Moab server is currently scheduling and responding to client requests.

Other High Availability Configuration

Moab has many features to improve the availability of a cluster beyond the ability to automatically relocate to another execution server. The following table describes some of these features.

Feature	Description
AMCFG[] BACKUPHOST	If using the Moab Accounting Manager, you may enable high availability with the accounting manager by specifying a backup server as in the following example: `AMCFG[mam] BACKUPHOST=headnode2`
JOBACTIONONNODEFAILURE	If a node allocated to an active job fails, it is possible for the job to continue running indefinitely even though the output it produces is of no value. Setting this parameter allows the scheduler to automatically preempt these jobs when a node failure is detected, possibly allowing the job to run elsewhere and also allowing other allocated nodes to be used by other jobs.
SCHEDCFG[] RECOVERYACTION	If a catastrophic failure event occurs (SIGSEGV or SIGILL signal is triggered), Moab can be configured to automatically restart, trap the failure, ignore the failure, or behave in the default manner for the specified signal. These actions are specified using the values *RESTART*, *TRAP*, *IGNORE*, or *DIE*, as in the following example: `SCHEDCFG[bas] MODE=NORMAL RECOVERYACTION=RESTART`

Malleable Jobs

Malleable jobs are jobs that can be adjusted in terms of resources and duration required, and which allow the scheduler to maximize job responsiveness by selecting a job's resource shape or footprint prior to job execution. Once a job has started, however, its resource footprint is fixed until job completion.

To enable malleable jobs, the underlying resource manager must support dynamic modification of resource requirements prior to execution (i.e., TORQUE) and the jobs must be submitted using the TRL (task request list) resource manager extension string. With the **TRL** attribute specified, Moab will attempt to select a start time and resource footprint to minimize job completion time and maximize overall effective system utilization (i.e., *<AverageJobEfficiency> * <AverageSystemUtilization>*).

Example 3-153:

With the following job submission, Moab will execute the job in one of the following configurations: 1 node for 1 hour, 2 nodes for 30 minutes, or 4 nodes for 15 minutes.

```
> qsub -l nodes=1,trl=1@3600:2@1800:4@900 testjob.cmd
job 72436.orion submitted
```

Identity Managers

- Identity Manager Overview
- Basic Configuration
- Importing Credential Fairness Policies
- Identity Manager Data Format
- Identity Manager Conflicts
- Refreshing Identity Manager Data

The Moab identity manager interface can be used to coordinate global and local information regarding users, groups, accounts, and classes associated with compute resources. The identity manager interface may also be used to allow Moab to automatically and dynamically create and modify user accounts and credential attributes according to current workload needs.

> Only one identity manager can be configured at a time.

Identity Manager Overview

Moab allows sites extensive flexibility when it comes to defining credential access, attributes, and relationships. In most cases, use of the USERCFG, GROUPCFG, ACCOUNTCFG, CLASSCFG, and QOSCFG parameters is adequate to specify the needed configuration. However, in certain cases such as the following, this approach may not be ideal or even adequate:

- Environments with very large user sets
- Environments with very dynamic credential configurations in terms of fairshare targets, priorities, service access constraints, and credential relationships
- Grid environments with external credential mapping information services
- Enterprise environments with fairness policies based on multi-cluster usage

Moab addresses these and similar issues through the use of an identity manager. An identity manager is configured with the IDCFG parameter and allows Moab to exchange information with an external identity management service. As with Moab resource manager interfaces, this service can be a full commercial package designed for this purpose, or something far simpler such as a web service, text file, or database.

Basic Configuration

Configuring an identity manager in basic read-only mode can be accomplished by simply setting the **SERVER** attribute. If Moab is to interact with the identity manager in read/write mode, some additional configuration may be required.

BLOCKCREDLIST	
Format	One or more comma-delimited object types from the following list: *acct*, *group*, or *user*
Details	If specified, Moab will block all jobs associated with credentials not explicitly reported in the most recent identity manager update. If the credential appears on subsequent updates, resource access will be immediately restored. Jobs will only be blocked if fairshare is enabled. This can be accomplished by setting the FSPOLICY parameter to any value such as in the following example: `FSPOLICY DEDICATEDPS`
Example	`IDCFG[test01] BLOCKCREDLIST=acct,user,groups` *Moab will block any jobs associated with accounts, users, or groups not in the most recent identity manager update.*

CREATECRED	
Format	*<BOOLEAN>* (default is *FALSE*)
Details	Specifies whether Moab should create credentials reported by the identity manager that have not yet been locally discovered or loaded via the resource manager. By default, Moab will only load information for credentials which have been discovered outside of the identity manager.
Example	`IDCFG[test01] CREATECRED=TRUE` *Moab will create credentials from* `test01` *that have not been previously loaded.*

REFRESHPERIOD	
Format	*minute*, *hour*, *day*, or *infinity* (default is *infinity*).
Details	If specified, Moab refreshes identity manager information once every specified iteration. If *infinity* is specified, the information is updated only at Moab start up.
Example	`IDCFG[test01] REFRESHPERIOD=hour` *Moab queries the identity manager every hour.*

RESETCREDLIST	
Format	One or more comma-delimited object types from the following list: *acct*, *group*, or *user*.
Details	If specified, Moab will reset the account access list and fairshare cap and target for all credentials of the specified type(s) regardless of whether they are included in the current info manager report. Moab will then load information for the specified credentials.
Example	`IDCFG[test01] RESETCREDLIST=group` *Moab will reset the account access list and fairshare target for all groups.*

SERVER	
Format	*<URL>*
Details	Specifies the protocol/interface to use to contact the identity manager.
Example	`IDCFG[test01] SERVER=exec://$HOME/example.pl` *Moab will use* `example.pl` *to communicate with the identity manager.*

UPDATEREFRESHONFAILURE	
Format	*<BOOLEAN>* (default is *FALSE*)
Details	When an **IDCFG** script fails, it retries almost immediately and continuously until it succeeds. When **UPDATEREFRESHONFAILURE** is set to *TRUE*, a failed script does not attempt to rerun immediately, but instead follows the specified **REFRESHPERIOD** schedule. When set to *TRUE*, **UPDATEREFRESHONFAILURE** updates the script execution timestamp, even if the script does not end successfully.
Example	`IDCFG[info] SERVER=exec:///home/tshaw/test/1447/bad_script.pl REFRESHPERIOD=hour UPDATEREFRESHONFAILURE=TRUE`

Importing Credential Fairness Policies

One common use for an identity manager is to import fairness data from a global external information service. As an example, assume a site needed to coordinate Moab group level fairshare targets with an allocation database that constrains total allocations available to any given group. To enable this, a configuration like the following might be used:

```
IDCFG[alloc] SERVER=exec://$TOOLSDIR/idquery.pl
...
```

The `tools/idquery.pl` script could be set up to query a local database and report its results to Moab. Each iteration, Moab will then import this information, adjust its internal configuration, and immediately respect the new fairness policies.

Identity Manager Data Format

When an identity manager outputs credential information either through an `exec` or `file` based interface, the data should be organized in the following format:
<CREDTYPE>`:`*<CREDID> <ATTR>*`=`*<VALUE>*

where

- *<CREDTYPE>* is one of `user`, `group`, `account`, `class`, or `qos`.
- *<CREDID>* is the name of the credential.
- *<ATTR>* is one of `adminlevel`, `alist`, `chargerate`, comment, emailaddress, `fstarget`, `globalfstarget`, `globalfsusage`, maxjob, maxmem, maxnode, maxpe, maxproc, maxps, maxwc, `plist`, `priority`, `qlist`, or `role`. Multi-dimensional policies work here as well.
- *<VALUE>* is the value for the specified attribute.

To clear a comment, set its value to `""`; for example: `comment=""`.

Example 3-154:

The following output may be generated by an `exec` based identity manager:

```
group:financial fstarget=16.3 alist=project2
group:marketing fstarget=2.5
group:engineering fstarget=36.7
group:dm fstarget=42.5
user:jason adminlevel=3
account:sales maxnode=128 maxjob=8,16
```

The following example limits user `bob` to 8 `matlab` generic resources.

```
user:bob MAXGRES[matlab]=8
```

To specify unlimited use of generic resources, set the value to `-1`.

Identity Manager Conflicts

When local credential configuration (as specified via `moab.cfg`) conflicts with identity manager configuration, the identity manager value takes precedence and the local values are overwritten.

Refreshing Identity Manager Data

By default, Moab only loads identity manager information once when it is first started up. If the identity manager data is dynamic, then you may want Moab to periodically update its information. To do this, set the **REFRESHPERIOD** attribute of the **IDCFG** parameter. Legal values are documented in the following table:

Value	Description
minute	Update identity information once per minute
hour	Update identity information once per hour
day	Update identity information once per day
infinity	Update identity information only at start-up (default)

Example 3-155:

```
IDCFG[hq] SERVER=exec://$TOOLSDIR/updatepolicy.sh REFRESHPERIOD=hour
```

Job credential feasibility is evaluated at job submission and start time.

Related topics

- Credential Overview
- Usage Limits/Throttling Policies

Generic System Jobs

Generic system jobs are system jobs with a trigger. They are useful for specifying steps in a workflow.

- Creating a Generic System Job
  - The Trigger
- Workflows Using Job Template Dependencies
  - Inheriting Resources in Workflows

Creating a Generic System Job

Generic system jobs are specified via a job template. The template can be selectable and you must use the **GENERICSYSJOB** attribute to let Moab know that this job template describes a generic system job and to specify a trigger, as shown in the following example:

```
JOBCFG[gen]
GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/genericTrig.py",Timeout=5:00
```

The Trigger

The generic system job's trigger that meets certain criteria. This trigger must have a timeout, an `Atype=Exec`, and the `EType` must equal "start". The timeout of the trigger will be used as the walltime for the job. The trigger will begin when the system job begins and the job will be considered completed when the trigger completes. The job will have the same completion code as the trigger. The walltime on the job template is not applicable in this case since the timeout of the trigger will be the walltime.

If the trigger fails, an error message will be attached to all of the job's parent VCs. You can view this in the `--xml` output of the VC query. The message includes the location of STDIN, STDOUT, and STDERR files. For example:

```
mvcctl -q ALL --xml

<Data>
<vc CREATETIME="1320184350" DESCRIPTION="Moab.1"
  FLAGS="DESTROYOBJECTS,DESTROYWHENEMPTY,HASSTARTED,WORKFLOW"
  JOBS="Moab.1" NAME="vc1" OWNER="user:frank">
<ACL aff="positive" cmp="%=" name="frank" type="USER"></ACL>
<MESSAGES>
<message COUNT="1" CTIME="1320184362"
     DATA="Trigger 10 failed on job Moab.1.setup- STDIN:
/tmp/ByLLl2wv/spool/vm.py.ieWPPS5 STDOUT:
/tmp/ByLLl2wv/spool/vm.py.oDMIXAW STDERR /tmp/ByLLl2wv/spool/vm.py.e2jD5iN"
     EXPIRETIME="1322776362" OWNER="frank" PRIORITY="0"
     TYPE="other" index="0"></message>
</MESSAGES>
<Variables>
<Variable name="VMID">vm1</Variable>
<Variable name="HV">TRUE</Variable>
</Variables>
</vc>
</Data>
```

You can specify other triggers on a generic system job using the **TRIGGER** attribute and delimiting them with semicolons. For example:

```
JOBCFG[gen]  GENERICSYSJOB=<genericSystemJobTriggerSpecs>
JOBCFG[gen]  TRIGGER=<triggerSpecs>;TRIGGER=<triggerSpecs>
```

Workflows Using Job Template Dependencies

To create workflows, use the following format:

```
JOBCFG[gen]  TEMPLATEDEPEND=AFTERANY:otherTemplate
```

This will create a job based on the template `otherTemplate`. The generic job will run after the otherTemplate job has finished. Afterany in the example means after all other jobs have completed, regardless of success.

Inheriting Resources in Workflows

The *INHERITRES* flag can be used to cause the same resources in one step of a workflow to be passed to the next step:

```
JOBCFG[gen]  TEMPLATEDEPEND=AFTERANY:otherTemplate
JOBCFG[otherTemplate]  INHERITRES=TRUE
```

This example forces the job based on `otherTemplate` to have the same resource requirements as its parent. When the `otherTemplate` job is finished, the INHERITRES flag will cause the parent to run on the same resources as the child.

The job that finishes first will pass its allocation up.

Any variables on the original job will be passed to the other jobs in the workflow. Variables can be added by other jobs in the workflow via the sets attribute in the generic system job's trigger. Other triggers must then request that variable name in the command line options.

You will need to set the carat (^) in order for the variable to be sent up to the job group.

If you set the variable, you need to set it in the STDOUT of the trigger script. See the example below:

```
JOBCFG[W1] GENERICSYSJOB=...,action='$HOME/W1.py $ipaddress' TEMPLATEDEPEND=AFTER:W2
JOBCFG[W2] TRIGGER=...,action='$HOME/W2.py',sets=^ipaddress
```

If a variable value is not set in STDOUT, it will be set to TRUE.

To set the variable to a specific value, the `W2.py` script must set the value in its STDOUT:

```
print "ipaddress=10.10.10.1" #This will be parsed by Moab and set as the value of the
"ipaddress" variable
```

Example 3-156:

To create a VM with a workflow using job template dependencies and generic system jobs, use the following format:

```
#The job template that is "gate" to the workflow
JOBCFG[CreateVMWithSoftware] TEMPLATEDEPEND=AFTEROK:InstallSoftware SELECT=TRUE

JOBCFG[InstallSoftware]
GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/setupSoftware.py
$IPAddr",Timeout=30:00
JOBCFG[InstallSoftware]  INHERITRES=TRUE
JOBCFG[InstallSoftware]  TEMPLATEDEPEND=AFTEROK:CreateVM

JOBCFG[CreateVM]  GENERICSYSJOB=EType=start,AType=exec,Action=$HOME/installVM.py
$HOSTLIST",Timeout=1:00:00,sets=^IPAddr
JOBCFG[CreateVM]  INHERITRES=TRUE
```

The user will then submit the job requesting what they need in the VM:

```
msub -l walltime=2:00:00,template=CreateVMWithSoftware,nodes=1:ppn=4,mem=1024
ActualWorkload.py
```

*The job will have the `CreateVMWithSoftware` template applied to it and will create the `InstallSoftware` job. The `InstallSoftware` job, because of **INHERITRES**, will have the same resource request (4 procs, 1GB of memory). This job then has its template applied to it which will do the same thing in creating the `CreateVM` job. The `CreateVM job` will then run, the trigger script will return the IP address of the new VM and pass its allocation up to the `InstallSoftware` job. The `InstallSoftware` job will use the IPAddr variable to find the VM and install the software. It will then return its resources up to the parent job, which will run the actual workload.*

Database Configuration

Moab supports connecting to a database via native SQLite3, and it can also connect to other databases using the ODBC driver. These optional external databases store some additional information that the MongoDB database does not and allow you to query them directly using SQL. These databases are slower, however, and only SQLite3, which does not allow external queries, is supported.

The SQLite3 connection is for storing statistics. Consider reviewing the SQLite web page Appropriate Uses for SQLite for information regarding the suitability of using SQLite3 on your system.

While the ODBC connection is useful for storing statistics, it also stores events, nodes, and jobs. You can further configure Moab to store checkpoint information to a database rather than to the flat text file (`.moab.ck`) if you set the CHECKPOINTWITHDATABASE parameter to *TRUE*.

Connecting to an external database makes Moab more searchable, allowing you to run queries for statistics and events rather than using regular expressions to draw the information from the Moab flat files.

SQLite3

Moab supports connecting to a database via native SQLite3. Database installation and configuration occurs automatically during normal Moab installation (configure, make install). If you did not follow the normal process to install Moab and need to install the database, do the following to manually install and configure Moab database support:

1. Create the database file `moab.db` in your moab home directory by running the following command from the root of your unzipped Moab build directory:
   `perl buildutils/install.sqlite3.pl` *<moab-home-directory>*
   - Verify that the command worked by running `lib/sqlite3` *<moab-home-directory>*`/moab.db`; at the resulting prompt, type `.tables` and press **ENTER**. You should see several tables such as `mcheckpoint` listed. Exit from this program with the .quit command.
   - The `perl buildutils/install.sqlite3.pl` *<moab-home-directory>* command may fail if your operating system cannot find the SQLite3 libraries. Also, Moab fails if unable to identify the libraries. To temporarily force the libraries to be found, run the following command:
     `export LD_LIBRARY_PATH=`*<location where libraries were copied>*

2. In the `moab.cfg` file in the `etc/` folder of the `home` directory, add the following line:

```
USEDATABASE  INTERNAL
```

To verify that Moab is running with SQLite3 support, start Moab and run the mdiag -S -v command. If there are no database-related error messages displayed, then Moab should be successfully connected to a database.

> `moabd` is a safe and recommended method of starting Moab if things are not installed in their default locations.

Connecting to a MySQL Database with an ODBC Driver

This documentation shows how to set up and configure Moab to connect to a MySQL database using the MySQL ODBC driver. This document assumes the necessary MySQL and ODBC drivers have already been installed and configured.

To set up and configure Moab to connect to a MySQL database using the MySQL ODBC driver, do the following:

This solution has been tested and works with these versions:

- libmyodbc - 5.1.5
- MySQL 5.1

1. Download and install the ODBC version of Moab. Install and configure Moab as normal but add the following in the Moab configuration file (`moab.cfg`):

```
USEDATABASE             ODBC
# Turn on stat profiling
USERCFG[DEFAULT]        ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]       ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]         ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]       ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]     ENABLEPROFILING=TRUE
NODECFG[DEFAULT]        ENABLEPROFILING=TRUE
```

2. Create the database in MySQL using the MySQL database dump contained in the `moab-db.sql` file. This file is located in the `contrib/sql` directory in the root of the binaries.

   - Run the following command:

```
mysql -u root -p < moab-db-mysql-create.sql
```

3. Configure the MySQL and ODBC driver. The `odbcinst.ini` file must be contained in `/etc`.

Run the following command to find the MySQL ODBC client driver. You could also query the `libmyodbc` package that was installed.

```
[root]# updatedb
[root]# locate libmyodbc
```

```
[MySQL]
Description = ODBC for MySQL
Driver = /usr/lib/odbc/libmyodbc.so
```

4. Configure Moab to use the MySQL ODBC driver. Moab uses an ODBC datastore file to connect to MySQL using ODBC. This file must be located in the Moab home directory (`/opt/moab` by default) and be named `dsninfo.dsn`, which is used by Moab. You need to have the following data in both `/etc/odbc.ini` and *$MOABHOMEDIR*`/dsninfo.dsn`:

```
[ODBC]
Driver = MySQL
USER = <username>
PASSWORD = <password>
Server = localhost
Database = Moab
Port = 3306
```

The user should have read/write privileges on the Moab database.

The preceding example file tells ODBC to use the MySQL driver, username `<username>`, password `<password>`, and to connect to MySQL running on the localhost on port `3306`. ODBC uses this information to connect to the database called `Moab`.

5. Test the ODBC to MySQL connection by running the isql command, which reads the `/etc/odbc.ini` file:

```
$ isql -v ODBC
+---------------------------------------+
| Connected! |
| |
| sql-statement |
| help [tablename] |
| quit |
| |
+---------------------------------------+
SQL> show tables;
+-----------------------------------------------------------------+
| Tables_in_Moab |
+-----------------------------------------------------------------+
| EventType |
| Events |
| GeneralStats |
| GenericMetrics |
| Moab |
| NodeStats |
| NodeStatsGenericResources |
| ObjectType |
| mcheckpoint |
+-----------------------------------------------------------------+
SQLRowCount returns 10
10 rows fetched
SQL>
```

If you encounter any errors using the isql command, there was a problem setting up the ODBC to MySQL connection. Try the following debugging steps to resolve the issue:

a. The `odbcinst.ini` and `odbc.ini` files are usually assumed to be located in `/etc`, but that is not always true. Use the `odbcinst -j` command to determine the assumed location of the files in your configuration.

```
[root#] odbcinst -j
unixODBC 2.2.12
DRIVERS............: /etc/unixODBC/odbcinst.ini
SYSTEM DATA SOURCES: /etc/unixODBC/odbc.ini
USER DATA SOURCES..: /home/adaptive/.odbc.ini
```

b. Because `odbcinst.ini` and `odbc.ini` are expected in `/etc/unixODBC`, not `/etc`, move them from `/etc` to `/etc/unixODBC`.

c. Use the `strace` command to determine where `isql` expects the `odbc.ini` and `odbcinst.ini` files. Note the location in which `isql` expects these files.

```
$ strace isql -v ODBC
```

6. With the ODBC driver configured, the database created, and Moab configured to use the database, start Moab for it to begin storing information in the created database.

> `> moabd` is a safe and recommended method of starting Moab if things are not installed in their default locations.

Related topics

- Importing Statistics to the Moab Database

Connecting to a PostgreSQL Database with an ODBC Driver

This documentation shows how to set up and configure Moab to connect to a PostgreSQL database using the ODBC driver. This document assumes the necessary ODBC drivers have already been installed and configured.

> Occasionally vacuuming your PostgresSQL database could improve Moab performance. See the PostgresSQL documentation for information on how to vacuum your database.

To set up and configure Moab to connect to a PostgreSQL database using the ODBC driver, do the following:

> This solution has been tested and works with the following file version:
>
> - odbc-postgresql - 1:08.03.0200-1.2

1. Configure the PostgreSQL and ODBC driver. `odbcinst.ini` file must be contained in `/etc`.

   > Run the following commands to find the PostgreSQL ODBC client driver and setup file. You could also query the `libodbcpsql` package that was installed.
   >
   > ```
   > [root]# updatedb
   > [root]# locate psqlodbc
   > [root]# locate libodbcpsql
   > ```

   ```
   [PostgreSQL]
   Description = PostgreSQL ODBC driver
   Driver = /usr/lib/odbc/psqlodbca.so
   Setup = /usr/lib/odbc/libodbcpsqlS.so
   Debug = 0
   CommLog = 1
   UsageCount = 2
   ```

2. Configure Moab to use the PostgreSQL ODBC driver. Moab uses an ODBC datastore file to connect to PostgreSQL using ODBC. This file must be located in the Moab home directory (`/opt/moab` by default) and be named `dsninfo.dsn`, which is used by Moab. If the following content, which follows the standard ODBC driver file syntax, is not already included in the `/etc/odbc.ini` file, make sure that you include it. Also, include the same content in the `dsninfo.dsn` file.

   ```
   [ODBC]
   Driver = PostgreSQL
   Description = PostgreSQL Data Source
   Servername = localhost
   Port = 5432
   Protocol = 8.4
   UserName = postgres
   Password = moab
   Database = Moab
   ```

The user should have read/write privileges on the Moab database.

The preceding example file tells ODBC to use the PostgreSQL driver, `postgres` user, `moab` password, and to connect to PostgreSQL running on the localhost on port 5432. ODBC uses this information and connects to the database called `Moab`.

3. Test the ODBC to PostgreSQL connection by running the isql command, which reads the `/etc/odbc.ini` file. If connected, you should be able to run the help command.

   If you encounter any errors using the isql command, there was a problem setting up the ODBC to MySQL connection. Try the following debugging steps to resolve the issue:

   a. The `odbcinst.ini` and `odbc.ini` files are usually assumed to be located in `/etc`, but that is not always true. Use the `odbcinst -j` command to determine the assumed location of the files in your configuration.

   ```
   [root#] odbcinst -j
   unixODBC 2.2.12
   DRIVERS............: /etc/unixODBC/odbcinst.ini
   SYSTEM DATA SOURCES: /etc/unixODBC/odbc.ini
   USER DATA SOURCES..: /home/adaptive/.odbc.ini
   ```

   b. Because `odbcinst.ini` and `odbc.ini` are expected in `/etc/unixODBC`, not `/etc`, move them from `/etc` to `/etc/unixODBC`.

   c. Use the `strace` command to determine where `isql` expects the `odbc.ini` and `odbcinst.ini` files. Note the location in which `isql` expects these files.

   ```
   $ strace isql -v ODBC
   ```

4. Create the database in PostgreSQL using the `moab-db-postgresql.sh` setup script contained in the `contrib/sql` directory at the root of the binary.

   - Run the script and provide the DB username that will attach to the Moab database (you must supply a DB username or the script will exit). The default admin user is `postgres`, but you can make a new user at this time:

   ```
   > ./moab-db-postgresql.sh postgres
   Create db user "postgres" in postgreSQL? (y/n)>
   ```

   - The script asks if you want to create the DB user you specified in postgreSQL. If the DB user already exists, answer 'n'. Otherwise, the DB user is created and it asks for the new user's password.
   - The script then creates the database "Moab".
   - Finally, as the DB user you provided, the script imports the DB schema from `moab-db-postgresql-create.sql` into the Moab database.

5. Download and install the ODBC version of Moab. Install and configure Moab as normal but add the following in the Moab configuration file (`moab.cfg`):

```
USEDATABASE              ODBC
# Turn on stat profiling
USERCFG[DEFAULT]         ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]        ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]          ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]        ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]      ENABLEPROFILING=TRUE
NODECFG[DEFAULT]         ENABLEPROFILING=TRUE
```

6. With the ODBC driver configured, the database created, and Moab configured to use the database, start Moab for it to begin storing information in the created database.

> > moabd is a safe and recommended method of starting Moab if things are not installed in their default locations.

Related topics

- Importing Statistics to the Moab Database

Connecting to an Oracle Database with an ODBC Driver

Context

This documentation shows how to set up and configure Moab to connect to an Oracle database using the ODBC driver.

To connect to an Oracle database with an ODBC driver

1. Install and configure the Oracle Instant Client with ODBC supporting libraries. For instructions, see **Installing the Oracle Instant Client** on page 785.
2. Open your Moab configuration file (`$MOABHOMEDIR/moab.cfg`) and add the following lines to the end of the file.

```
USEDATABASE ODBC

# Turn on stat profiling
USERCFG[DEFAULT]         ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]        ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]          ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]        ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]      ENABLEPROFILING=TRUE
NODECFG[DEFAULT]         ENABLEPROFILING=TRUE
```

3. Configure the Oracle ODBC Driver. The `odbcinst.ini` file must be contained in `/etc`.

```
[root]# vim /etc/odbcinst.ini
```

Run the following command to find the Oracle Instant Client driver. You could also query the Oracle Instant Client package that was installed.

```
[root]# updatedb && locate libsqora
```

Add the following text to the file.

```
[Oracle 11g ODBC driver]
Description     = Oracle ODBC driver for Oracle 11g
Driver          = /usr/lib/oracle/11.2/client64/lib/libsqora.so.11.1
Setup           =
FileUsage       =
CPTimeout       =
CPReuse         =
Driver Logging  = 7

[ODBC]
Trace = Yes
TraceFile = /tmp/odbc.log
ForceTrace = Yes
Pooling = No
DEBUG = 1
```

`Driver Logging` is set high (level 7) so that you can debug during the installation and configuration process if necessary. You can decrease the setting or remove the directive once you finish the process.

To configure the location of the ODBC log (`/tmp/odbc.log`), set the `TraceFile` attribute shown in the example above. See "unixODBC without the GUI" on the unixODBC website for more information.

4. Because the driver installed in step 1 is a shared library, run `ldd` to verify that it and all of its dependencies are installed and working.

```
[root]# ldd /usr/lib/oracle/11.2/client64/lib/libsqora.so.11.1
            linux-vdso.so.1 =>  (0x00007fff631ff000)
            libdl.so.2 => /lib64/libdl.so.2 (0x00007f8afbe83000)
            libm.so.6 => /lib64/libm.so.6 (0x00007f8afbbff000)
            libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f8afb9e1000)
            libnsl.so.1 => /lib64/libnsl.so.1 (0x00007f8afb7c8000)
            libclntsh.so.11.1 =>
/usr/lib/oracle/11.2/client64/lib/libclntsh.so.11.1 (0x00007f8af8e59000)
            libodbcinst.so.1 => not found
            libc.so.6 => /lib64/libc.so.6 (0x00007f8af8ac5000)
            /lib64/ld-linux-x86-64.so.2 (0x0000003bdb000000)
            libnnz11.so => /usr/lib/oracle/11.2/client64/lib/libnnz11.so
(0x00007f8af86f8000)
            libaio.so.1 => /lib64/libaio.so.1 (0x00007f8af84f6000)
```

If the command returns `libodbcinst.so.1 => not found`, create a symbolic link from `/usr/lib64/libodbcinst.so.1` to `/usr/lib64/libodbcinst.so.2`. This is a known Red Hat issue. See Red Hat Bugzilla for more information.

```
[root]# locate libodbcinst

/usr/local/lib/libodbcinst.so.2

[root]# cd /usr/lib64
[root]# ln -s libodbcinst.so.2 libodbcinst.so.1
```

Rerun `ldd`. It should load `libsqora.so.11.1` without error, as shown in the `ldd` example above.

If the `ldd` command returns a warning like this: "ldd: warning: you do not have execution permission for `/usr/lib/oracle/11.2/client64/lib/libsqora.so.11.1'", run the following command:

```
[root]# chmod 755 /usr/lib/oracle/11.2/client64/lib/lib*
```

Rerun `ldd`. It should load `libsqora.so.11.1` without error, as shown in the `ldd` example above.

5. Configure Moab to use the Oracle ODBC driver. This example assumes that a Moab user exists and has been granted read and write privileges to the MOAB database instance referred to on the **Installing the Oracle Instant Client** on page 785 page.

```
[root]# vim $MOABHOMEDIR/dsninfo.dsn
```

Add the following lines the file, but change `ServerName`, `UserName`, and `Password` to suit your own system. `ServerName` is the name of the Oracle database instance. `Username` and `Password` are the credentials used to connect to that instance.

```
[ODBC]
Application Attributes = T
Attributes = W
BatchAutocommitMode = IfAllSuccessful
BindAsFLOAT = F
CloseCursor = F
DisableDPM = F
DisableMTS = T
Driver = Oracle 11g ODBC driver
DSN = ODBC
EXECSchemaOpt =
EXECSyntax = T
Failover = T
FailoverDelay = 10
FailoverRetryCount = 10
FetchBufferSize = 64000
ForceWCHAR = F
Lobs = T
Longs = T
MaxLargeData = 0
MetadataIdDefault = F
QueryTimeout = T
ResultSets = T
ServerName = MOAB
SQLGetData extensions = F
Translation DLL =
Translation Option = 0
DisableRULEHint = T
UserID = moab
Password = moab
StatementCache=F
CacheBufferSize=20
UseOCIDescribeAny=F
MaxTokenSize=8192
```

6. Add the contents of the `dsninfo.dsn` file to `/etc/odbc.ini`. Because the contents of `dsninfo.dsn` are required in both files, use the following command to concatenate the contents of `dsninfo.dsn` to `/etc/odbc.ini`. If the `odbc.ini` file already has content, verify that there are no conflicts.

```
[root]# cat $MOABHOMDIR/dsninfo.dsn >> /etc/odbc.ini
```

7. Create a directory to store the `tnsnames.ora` file you will create in the next step.

```
[root]# mkdir /etc/oracle
```

8. Create the `tnsnames.ora` file. The `ServerName` in `$MOABHOMEDIR/dsninfo.dsn` tells the Oracle ODBC driver what `tnsnames.ora` entry to use (`MOAB`). The `MOAB` `tnsnames` entry tells the Oracle ODBC driver to connect to server `adaptive-oracle` on the local domain (`ac`) on port 1561 using TCP and to connect to the Oracle instance named `MOAB` (The SID is the unique name of the instance).

```
[root]# cat >/etc/oracle/tnsnames.ora <<EOL
MOAB =
   (DESCRIPTION =
     (ADDRESS_LIST =
       (ADDRESS = (PROTOCOL = TCP)(HOST = adaptive-oracle)(PORT = 1561))
     )
     (CONNECT_DATA =
       (SID = MOAB)
     )
   )
EOL
```

9. Create a profile script (`oracle-instant-client.sh`) to be invoked by the operating system at startup. This script will set the `ORACLE_HOME`, `TWO_TASK`, and `TNS_ADMIN` environment variables required by Oracle and will amend the `LD_LIBRARY_PATH` to include required Oracle client libraries in the library search path.

```
[root]# cat >/etc/profile.d/oracle-instant-client.sh <<EOL
# Set ORACLE_HOME to the directory where the bin and lib directories are located
for the oracle client
export ORACLE_HOME=/usr/lib/oracle/11.2/client64

# No need to add ORACLE_HOME to the linker search path. oracle-instant-client.conf
in
# /etc/ld.so.conf.d should already contain /usr/lib/oracle/11.2/client64.
# Alternately, you can set it here by uncommenting the following line:
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib

# Define the default location where Oracle should look for the server
export TWO_TASK=//adaptive-oracle:1561/listener

# Define where to find the tnsnames.ora file
export TNS_ADMIN=/etc/oracle
EOL
```

10. Source the `oracle-instant-client.sh` script and verify that each environment variable is set correctly.

```
[root]# source /etc/profile.d/oracle-instant-client.sh
[root]# echo $ORACLE_HOME
[root]# echo $LD_LIBRARY_PATH
[root]# echo $TWO_TASK
[root]# echo $TNS_ADMIN
```

11. Modify either the Moab startup script (`/etc/init.d/moab`) – recommended – or the `moabd` script (`/opt/moab/sbin/moabd`) to source `oracle-instant-client.sh`.

   - Moab startup script (recommended): the following example suggests a location to source the `oracle-instant-client.sh` script within the Moab startup script.

```
...

# Export all environment variables required by the Oracle Instant Client
. /etc/profile.d/oracle-instant-client.sh

export MOABHOMEDIR=/opt/moab

...
```

- moabd shell script: the following example will resemble the moabd script in /opt/moab/sbin. Note that the moabd script is not invoked by the Moab startup script; The Moab startup script invokes the Moab binary (/opt/moab/sbin/moab) by default.

```
#!/bin/sh
#
#   Copyright (C) 2012 by Adaptive Computing Enterprises, Inc. All Rights
Reserved.
#

# Export all environment variables required by the Oracle Instant Client
. /etc/profile.d/oracle-instant-client.sh

MOABHOMEDIR="/opt/moab" LD_LIBRARY_PATH="/opt/moab/lib:$LD_LIBRARY_PATH" moab
"$@"
```

12. Verify the Oracle ODBC driver is working.

```
isql -v ODBC
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
```

If you encounter any errors using the isql command, there was a problem setting up the ODBC to Oracle connection. Try the following debugging steps to resolve the issue:

a. The odbcinst.ini and odbc.ini files are usually assumed to be located in /etc, but that is not always true. Use the odbcinst -j command to determine the assumed location of the files in your configuration.

```
[root#] odbcinst -j
unixODBC 2.2.12
DRIVERS............: /etc/unixODBC/odbcinst.ini
SYSTEM DATA SOURCES: /etc/unixODBC/odbc.ini
USER DATA SOURCES..: /home/adaptive/.odbc.ini
```

b. Because odbcinst.ini and odbc.ini are expected in /etc/unixODBC, not /etc, move them from /etc to /etc/unixODBC.

c. Use the strace command to determine where isql expects the odbc.ini and odbcinst.ini files. Note the location in which isql expects these files.

```
$ strace isql -v ODBC
```

13. If you have not already done so, create the database tables in Oracle using the moab-db-oracle-create.sql script located in the contrib/sql directory in the root of the binaries. This example assumes that you are logged into the MOAB database instance (referred to on the **Installing the Oracle Instant Client** on page 785 page) as Moab user with read and write privileges.

```
SQL> @./contrib/sql/moab-db-oracle-create.sql
```

14. Verify that the database schema installed correctly by listing the tables. Your results should look like this:

```
SQL> select table_name from all_tables where owner = 'MOAB';
+-------------------------------+
| TABLE_NAME                    |
+-------------------------------+
| TRIGGERS                      |
| MOAB                          |
| OBJECTTYPE                    |
| VCS                           |
| EVENTTYPE                     |
| JOBHISTORY                    |
| MCHECKPOINT                   |
| NODES                         |
| EVENTS                        |
| NODESTATSGENERICRESOURCES     |
| JOBS                          |
| RESERVATIONS                  |
| GENERICMETRICS                |
| REQUESTS                      |
| GENERALSTATS                  |
| NODESTATS                     |
+-------------------------------+
SQLRowCount returns -1
16 rows fetched
```

15. Restart Moab.

```
[root]# mschedctl -R
```

16. Verify Moab is correctly configured to write to the Oracle database by doing each of the following steps:

    a. Tail the `moab.log` file for ODBC errors.

    ```
    # Check the $MOABHOMEDIR/log/moab.log file for ODBC errors. You should see a few
    hits even if there are no errors.
    [root]# tail -f $MOABHOMEDIR/log/moab.log | grep -i odbc
    ```

    b. Log in to the Moab Oracle database.

    In the first example below, `isql` will search `/etc/odbc.ini` for "[ODBC]". unixODBC will then use the Oracle 11g ODBC driver defined in `/etc/odbcinst.ini` to establish a connection. The `ServerName` in `/etc/odbc.ini` tells the Oracle driver to reference the `MOAB` `tnsnames` entry in `/etc/oracle/tnsnames.ora` for connection parameters.

    The second example uses `sqlplus` and a connect string to connect.

    Try both connection methods.

    ```
    # Log in to Oracle. Try both isql and sqlplus64 clients.
    [root]# isql -v ODBC
    ```

    ```
    [root]# sqlplus64 moab/moab@adaptive-oracle:1561/MOAB
    ```

    c. Select some data from one or more of the tables (Nodes, Events, and the like) to verify that data is being stored in the Moab Oracle instance.

```
# sqlplus64 moab/moab@adaptive-oracle:1561/MOAB

SQL*Plus: Release 11.2.0.4.0 Production on Fri Oct 4 14:59:02 2013

Copyright (c) 1982, 2013, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Release 11.2.0.1.0 - 64bit Production

SQL> select table_name from user_tables;

TABLE_NAME
------------------------------
JOBS
REQUESTS
RESERVATIONS
VCS
EVENTTYPE
GENERALSTATS
GENERICMETRICS
NODESTATS
NODESTATSGENERICRESOURCES
EVENTS
JOBHISTORY
MCHECKPOINT
NODES
TRIGGERS
MOAB
OBJECTTYPE

16 rows selected.
```

Related topics

- **Installing the Oracle Instant Client** on page 785
- **Connecting to a MySQL Database with an ODBC Driver** on page 773
- **Connecting to a PostgreSQL Database with an ODBC Driver** on page 776
- **Database Configuration** on page 772

Installing the Oracle Instant Client

Context

The following procedure demonstrates how to install the correct ODBC drivers for your Oracle database. This guide is a prerequisite for the **Connecting to an Oracle Database with an ODBC Driver** on page 778 task. Each step must be performed as root.

To install the Oracle Instant Client

1. Go to the "Install Client Downloads" page on the Oracle website. Choose the link that matches your system type (for instance, Instant Client for Linux x86-64). Choose **Accept License Agreement** at the top of the page and download the following RPM or zip files for your target version (such as 11.2):

   The process of connecting Oracle to Moab Workload Manager has been tested on Oracle Instant Client version 11.2. The process may work with other versions, but they are not supported.

- Basic (`oracle-instantclient11.2-basic-11.2.0.4.0-1.x86_64.rpm`)
- SQL Plus (`oracle-instantclient11.2-sqlplus-11.2.0.4.0-1.x86_64.rpm`)
- ODBC (`oracle-instantclient11.2-odbc-11.2.0.4.0-1.x86_64.rpm`)

2. Install the packages. This example installs the RPMs.

```
[root]# rpm -i ./oracle-instantclient11.2-basic-11.2.0.4.0-1.x86_64.rpm
[root]# rpm -i ./oracle-instantclient11.2-sqlplus-11.2.0.4.0-1.x86_64.rpm
[root]# rpm -i ./oracle-instantclient11.2-odbc-11.2.0.4.0-1.x86_64.rpm
```

3. Create a configuration file in `/etc/ld.so.conf.d` to add the Oracle client libraries to the `LD_LIBRARY_PATH`.

   To confirm where the RPMs installed the libraries, run `rpm -qlp` *<rpmFileName>*.

```
[root]# cat >/etc/ld.so.conf.d/oracle-instant-client.conf <<EOL
/usr/lib/oracle/11.2/client64/lib
EOL
```

> If you installed Oracle Instant Client from a repository, run `repoquery -ql` *<rpmName>* instead.

   Rebuild the `LD_LIBRARY_PATH`.

```
[root]# ldconfig
```

4. Connect to the database using `sqlplus`. If you used RPMs to install the client, the 32-bit and 64-bit clients are already in your PATH.

```
[root]# sqlplus64 moab/moab@adaptive-oracle:1561/MOAB

SQL*Plus: Release 11.2.0.4.0 Production on Mon Sep 30 14:35:10 2013

Copyright (c) 1982, 2013, Oracle.  All rights reserved.

Connected to:
Oracle Database 11g Release 11.2.0.1.0 - 64bit Production
```

The 64-bit `sqlplus` client was used to connect to a 64-bit 11g instance called MOAB, which is hosted on `adaptive-oracle.ac`.

5. Verify that you are logged in to the correct database.

```
SQL> select name from v$database
  2  ;
NAME
---------
MOAB
```

6. Create the database in Oracle using the `moab-db-oracle-create.sh` script located in the `contrib/sql` directory in the root of the binaries.

> Useful comments are at the top of the script. Read the comments before running the script.

```
[root]# ./moab-db-oracle-create.sh
```

7. Display all of user MOAB's tables.

```
SQL> select table_name from all_tables where owner = 'MOAB';

TABLE_NAME
------------------------------
TRIGGERS
MOAB
OBJECTTYPE
VCS
EVENTTYPE
JOBHISTORY
MCHECKPOINT
NODES
EVENTS
NODESTATSGENERICRESOURCES
JOBS
RESERVATIONS
GENERICMETRICS
REQUESTS
GENERALSTATS
NODESTATS

16 rows selected.
SQL>
```

8. Generate a script to describe all of user MOAB's tables. Cut and paste the following into a terminal that is *not* logged in to SQLPlus.

```
[root]# cat > /tmp/generateDescribe.sql <<EOL
SET HEADING OFF
SET FEEDBACK OFF
SET ECHO OFF
SET PAGESIZE 0
SPOOL /tmp/describeAllUserTables.sql
select 'desc '||owner||'.'||table_name||';' from all_tables where owner = 'MOAB';
SPOOL OFF
EOL
```

9. Run describeAllUserTables.sql.

```
[root]# SQL> start /tmp/describeAllUserTables.sql
```

Related topics

- **Connecting to an Oracle Database with an ODBC Driver** on page 778
- **Database Configuration** on page 772

Migrating Your Database to Newer Versions of Moab

Sometimes when upgrading from an older version of Moab to a newer version, you must update your database schema. If the schema Moab expects to operate against is different from the actual schema of the database Moab is connected to, Moab might not be able to use the database properly and data might be lost.

When upgrading the Moab database schema from an old version, you must perform each version upgrade in order. You cannot skip versions. For example, to migrate from version 6.1 to version 8.0, you must follow the steps in **Migrating from Moab 6.1 to Moab 7.0** on page 789, **Migrating from Moab 7.0 to**

Moab 7.1 on page 789, **Migrating from Moab 7.1 to Moab 7.2** on page 789, **Migrating from Moab 7.2 to Moab 7.2.6** on page 788, **Migrating from Moab 7.2.6 to Moab 7.5** on page 788, and then **Migrating from Moab 7.5 to Moab 8.0** on page 788.

If you are upgrading your database to the 8.0 schema from 7.2.x where "x" is 5 or lower, you must complete the instructions for the following sections in order:

- **Migrating from Moab 7.2 to Moab 7.2.6** on page 788
- **Migrating from Moab 7.2.6 to Moab 7.5** on page 788.
- **Migrating from Moab 7.5 to Moab 8.0** on page 788

Migrating from Moab 7.5 to Moab 8.0

In Moab Workload Manager 8.0, column names that have become reserved words in newer versions of MySQL, PostgreSQL, and Oracle were renamed to eliminate the need to quote column names in SQL statements. Also, a few additional columns were added to existing tables to support Moab's Green feature. To upgrade your database with these changes, use the `moab-db-<database>-upgrade8_0.sql` file located in the `contrib/sql` directory in the root of the binaries. For example, to migrate your MySQL database from the 7.5 (or later) schema, run the following:

```
[root@]# mysql -u root -D <database name> -p < moab-db-mysql-upgrade8_0.sql
```

The database name is usually "Moab".

Similar migration scripts exist for Oracle and PostgreSQL.

Migrating from Moab 7.2.6 to Moab 7.5

In Moab Workload Manager 7.5, column names that are reserved words in databases supported by Adaptive Computing were renamed to eliminate the need to quote column names in SQL statements. To upgrade your database with these changes, use the `moab-db-<database>-upgrade7_5.sql` file located in the `contrib/sql` directory in the root of the binaries. For example, to migrate your MySQL database from the 7.2.6 (or later) schema, run the following:

```
[root@]# mysql -u root -D <database name> -p < moab-db-mysql-upgrade7_5.sql
```

The database name is usually "Moab".

Similar migration scripts exist for Oracle and PostgreSQL.

Migrating from Moab 7.2 to Moab 7.2.6

In Moab Workload Manager 7.2.6, several columns were extended and the primary key on the Triggers table changed. To upgrade your database with these changes, use the `moab-db-<database>-upgrade7_2_6.sql` file located in the `contrib/sql` directory in the root of the binaries. For example, to migrate your MySQL database from the 7.2.x (pre-7.2.6) schema to the 7.2.6 schema, run the following:

```
[root@]# mysql -u root -D <database name> -p < moab-db-mysql-upgrade7_2_6.sql
```

The database name is usually "Moab".

Similar migration scripts exist for Oracle and PostgreSQL.

The 7.2.6 database upgrade is compatible with all earlier versions of 7.2.

Migrating from Moab 7.1 to Moab 7.2

In Moab 7.2, several events in the event table related to the Accounting Manager were renamed. To upgrade your database with these changes, use the `moab-db-<database>-upgrade7_2.sql` file located in the `contrib/sql` directory in the root of the binaries. For example, to migrate your MySQL database from the 7.1 schema to the 7.2 schema, run the following:

```
[root@]# mysql -u root -D <database name> -p < moab-db-mysql-upgrade7_2.sql
```

The database name is usually "Moab".

Similar migration scripts exist for Oracle and PostgreSQL.

Migrating from Moab 7.0 to Moab 7.1

In Moab 7.1, Offset was renamed TriggerOffset in the Triggers table. To upgrade your database with these changes, use the `moab-db-<database>-upgrade7_1.sql` file located in the root of the binaries. For example, to migrate your MySQL database from the 7.0 schema to the 7.1 schema, run the following:

```
[root@]# mysql -u root -D <database name> -p < moab-db-mysql-upgrade7_1.sql
```

The database name is usually "Moab".

Similar migration scripts exist for Oracle and PostgreSQL.

Migrating from Moab 6.1 to Moab 7.0

In Moab 7.0, the Moab table has been removed from the database, and MoabInfo and JobHistory tables have been added to it. To upgrade your database with these changes, use the `moab-db-mysql-upgrade6_1.sql` file located in the `contrib/sql` directory in the root of the binaries. This is done by running the following command:

```
[root@]# mysql -u root -D <database name> -p < moab-db-mysql-upgrade6_1.sql
```

The database name is usually "Moab".

Your MySQL database is updated for Moab 7.0.

Migrating from Moab 6.0 to Moab 6.1

An Events table has been added to the database in Moab 6.1. Update the `contrib/sql/moab-db.sql` file with the following table:

```
CREATE TABLE Events (
ID INTEGER,
ObjectType INTEGER,
EventType INTEGER,
EventTime INTEGER UNSIGNED,
ObjectName VARCHAR(64),
Name VARCHAR(64),
Description TEXT,
PRIMARY KEY (ID)
);
```

Use the mdiag -e `--xml` command in the following format to query the events table.

```
mdiag -e [-w <starttime>|<endtime>|<eventtypes>|<oidlist>|<eidlist>|<objectlist>] --
xml
```

The table is then displayed with all specified events configured with the RECORDEVENTLIST parameter.

> If the command could return a large of data, redirect the output. `mdiag -e --xml > outputfile`

Migrating from Moab 5.4 to Moab 6.0

The ODBC database schema has been updated for Moab 6.0. When updating Moab to version 6.0, the changes below must be applied to the database for database functionality to work. Below are the SQL statements required to update the schema for Moab 6.0.

> These changes are only necessary for an ODBC database. An SQLite database does not require an update.

```
ALTER TABLE Events ADD COLUMN Name VARCHAR(64);
ALTER TABLE Events MODIFY Description TEXT;

CREATE TABLE Nodes (
  ID VARCHAR(64),
  State VARCHAR(64),
  OperatingSystem VARCHAR(64),
  ConfiguredProcessors INTEGER UNSIGNED,
  AvailableProcessors INTEGER UNSIGNED,
  ConfiguredMemory INTEGER UNSIGNED,
  AvailableMemory INTEGER UNSIGNED,
  Architecture VARCHAR(64),
  AvailGres VARCHAR(64),
  ConfigGres VARCHAR(64),
  AvailClasses VARCHAR(64),
  ConfigClasses VARCHAR(64),
  ChargeRate DOUBLE,
  DynamicPriority DOUBLE,
  EnableProfiling INTEGER UNSIGNED,
  Features VARCHAR(64),
  GMetric VARCHAR(64),
  HopCount INTEGER UNSIGNED,
  HypervisorType VARCHAR(64),
  IsDeleted INTEGER UNSIGNED,
  IsDynamic INTEGER UNSIGNED,
  JobList VARCHAR(64),
  LastUpdateTime INTEGER UNSIGNED,
  LoadAvg DOUBLE,
  MaxLoad DOUBLE,
  MaxJob INTEGER UNSIGNED,
  MaxJobPerUser INTEGER UNSIGNED,
  MaxProc INTEGER UNSIGNED,
  MaxProcPerUser INTEGER UNSIGNED,
  OldMessages VARCHAR(64),
  NetworkAddress VARCHAR(64),
  NodeSubstate VARCHAR(64),
  Operations VARCHAR(64),
  OSList VARCHAR(64),
  Owner VARCHAR(64),
  ResOvercommitFactor VARCHAR(64),
  Partition VARCHAR(64),
  PowerIsEnabled INTEGER UNSIGNED,
  PowerPolicy VARCHAR(64),
  PowerSelectState VARCHAR(64),
  PowerState VARCHAR(64),
  Priority INTEGER UNSIGNED,
  PriorityFunction VARCHAR(64),
  ProcessorSpeed INTEGER UNSIGNED,
  ProvisioningData VARCHAR(64),
  AvailableDisk INTEGER UNSIGNED,
  AvailableSwap INTEGER UNSIGNED,
  ConfiguredDisk INTEGER UNSIGNED,
  ConfiguredSwap INTEGER UNSIGNED,
  ReservationCount INTEGER UNSIGNED,
  ReservationList VARCHAR(64),
  ResourceManagerList VARCHAR(64),
  Size INTEGER UNSIGNED,
  Speed DOUBLE,
  SpeedWeight DOUBLE,
  TotalNodeActiveTime INTEGER UNSIGNED,
  LastModifyTime INTEGER UNSIGNED,
  TotalTimeTracked INTEGER UNSIGNED,
```

```
  TotalNodeUpTime INTEGER UNSIGNED,
  TaskCount INTEGER UNSIGNED,
  VMOSList VARCHAR(64),
  PRIMARY KEY (ID)
);



CREATE TABLE Jobs (
  ID VARCHAR(64),
  SourceRMJobID VARCHAR(64),
  DestinationRMJobID VARCHAR(64),
  GridJobID VARCHAR(64),
  AName VARCHAR(64),
  User VARCHAR(64),
  Account VARCHAR(64),
  Class VARCHAR(64),
  QOS VARCHAR(64),
  OwnerGroup VARCHAR(64),
  JobGroup VARCHAR(64),
  State VARCHAR(64),
  EState VARCHAR(64),
  SubState VARCHAR(64),
  UserPriority INTEGER UNSIGNED,
  SystemPriority INTEGER UNSIGNED,
  CurrentStartPriority INTEGER UNSIGNED,
  RunPriority INTEGER UNSIGNED,
  PerPartitionPriority TEXT,
  SubmitTime INTEGER UNSIGNED,
  QueueTime INTEGER UNSIGNED,
  StartTime INTEGER UNSIGNED,
  CompletionTime INTEGER UNSIGNED,
  CompletionCode INTEGER,
  UsedWalltime INTEGER UNSIGNED,
  RequestedMinWalltime INTEGER UNSIGNED,
  RequestedMaxWalltime INTEGER UNSIGNED,
  CPULimit INTEGER UNSIGNED,
  SuspendTime INTEGER UNSIGNED,
  HoldTime INTEGER UNSIGNED,
  ProcessorCount INTEGER,
  RequestedNodes INTEGER,
  ActivePartition VARCHAR(64),
  SpecPAL VARCHAR(64),
  DestinationRM VARCHAR(64),
  SourceRM VARCHAR(64),
  Flags TEXT,
  MinPreemptTime INTEGER UNSIGNED,
  Dependencies TEXT,
  RequestedHostList TEXT,
  ExcludedHostList TEXT,
  MasterHostName VARCHAR(64),
  GenericAttributes TEXT,
  Holds TEXT,
  Cost DOUBLE,
  Description TEXT,
  Messages TEXT,
  NotificationAddress TEXT,
  StartCount INTEGER UNSIGNED,
  BypassCount INTEGER UNSIGNED,
  CommandFile TEXT,
  Arguments TEXT,
  RMSubmitLanguage TEXT,
```

```
  StdIn TEXT,
  StdOut TEXT,
  StdErr TEXT,
  RMOutput TEXT,
  RMError TEXT,
  InitialWorkingDirectory TEXT,
  UMask INTEGER UNSIGNED,
  RsvStartTime INTEGER UNSIGNED,
  BlockReason TEXT,
  BlockMsg TEXT,
  PSDedicated DOUBLE,
  PSUtilized DOUBLE,
  PRIMARY KEY (ID)
);

CREATE TABLE Requests (
  JobID VARCHAR(64),
  RIndex INTEGER UNSIGNED,
  AllocNodeList VARCHAR(1024),
  AllocPartition VARCHAR(64),
  PartitionIndex INTEGER UNSIGNED,
  NodeAccessPolicy VARCHAR(64),
  PreferredFeatures TEXT,
  RequestedApp VARCHAR(64),
  RequestedArch VARCHAR(64),
  ReqOS VARCHAR(64),
  ReqNodeSet VARCHAR(64),
  ReqPartition VARCHAR(64),
  MinNodeCount INTEGER UNSIGNED,
  MinTaskCount INTEGER UNSIGNED,
  TaskCount INTEGER UNSIGNED,
  TasksPerNode INTEGER UNSIGNED,
  DiskPerTask INTEGER UNSIGNED,
  MemPerTask INTEGER UNSIGNED,
  ProcsPerTask INTEGER UNSIGNED,
  SwapPerTask INTEGER UNSIGNED,
  NodeDisk INTEGER UNSIGNED,
  NodeFeatures TEXT,
  NodeMemory INTEGER UNSIGNED,
  NodeSwap INTEGER UNSIGNED,
  NodeProcs INTEGER UNSIGNED,
  GenericResources TEXT,
  ConfiguredGenericResources TEXT,
  PRIMARY KEY (JobID,RIndex)
);

INSERT INTO ObjectType (Name,ID) VALUES ("Rsv",13);
INSERT INTO ObjectType (Name,ID) VALUES ("RM",14);
INSERT INTO ObjectType (Name,ID) VALUES ("Sched",15);
INSERT INTO ObjectType (Name,ID) VALUES ("SRsv",16);
INSERT INTO ObjectType (Name,ID) VALUES ("Sys",17);
INSERT INTO ObjectType (Name,ID) VALUES ("TNode",18);
INSERT INTO ObjectType (Name,ID) VALUES ("Trig",19);
INSERT INTO ObjectType (Name,ID) VALUES ("User",20);
INSERT INTO ObjectType (Name,ID) VALUES ("CJob",23);
INSERT INTO ObjectType (Name,ID) VALUES ("GRes",30);
INSERT INTO ObjectType (Name,ID) VALUES ("Gmetric",31);
INSERT INTO ObjectType (Name,ID) VALUES ("Stats",39);
INSERT INTO ObjectType (Name,ID) VALUES ("TJob",42);
INSERT INTO ObjectType (Name,ID) VALUES ("Paction",43);
INSERT INTO ObjectType (Name,ID) VALUES ("VM",45);
INSERT INTO ObjectType (Name,ID) VALUES ("JGroup",48);
```

```
INSERT INTO EventType (Name,ID) VALUES ("TRIGTHRESHOLD",41);
INSERT INTO EventType (Name,ID) VALUES ("VMCREATE",42);
INSERT INTO EventType (Name,ID) VALUES ("VMDESTROY",43);
INSERT INTO EventType (Name,ID) VALUES ("VMMIGRATE",44);
INSERT INTO EventType (Name,ID) VALUES ("VMPOWERON",45);
INSERT INTO EventType (Name,ID) VALUES ("VMPOWEROFF",46);
INSERT INTO EventType (Name,ID) VALUES ("NODEMODIFY",47);
INSERT INTO EventType (Name,ID) VALUES ("NODEPOWEROFF",48);
INSERT INTO EventType (Name,ID) VALUES ("NODEPOWERON",49);
INSERT INTO EventType (Name,ID) VALUES ("NODEPROVISION",50);
INSERT INTO EventType (Name,ID) VALUES ("ALLSCHEDCOMMAND",51);
INSERT INTO EventType (Name,ID) VALUES ("AMCANCEL",52);
INSERT INTO EventType (Name,ID) VALUES ("AMDEBIT",53);
INSERT INTO EventType (Name,ID) VALUES ("AMQUOTE",54);
INSERT INTO EventType (Name,ID) VALUES ("AMRESERVE",55);
INSERT INTO EventType (Name,ID) VALUES ("RMPOLLEND",56);
INSERT INTO EventType (Name,ID) VALUES ("RMPOLLSTART",57);
INSERT INTO EventType (Name,ID) VALUES ("SCHEDCYCLEEND",58);
INSERT INTO EventType (Name,ID) VALUES ("SCHEDCYCLESTART",59);
INSERT INTO EventType (Name,ID) VALUES ("JOBCHECKPOINT",60);

ALTER TABLE GeneralStats ADD COLUMN TotalConfiguredProcCount INTEGER;
```

Importing Statistics from stats/DAY.* to the Moab Database

The `contrib/stat_converter` folder contains the files to build `mstat_converter`, an executable that reads file-based statistics in a Moab stats directory and dumps them into a database. It also reads the Moab checkpoint file (`.moab.ck`) and dumps that to the database as well. It uses the `$MOABHOMEDIR/moab.cfg` file to connect to the appropriate database and reads the statistics files from `$MOABHOMEDIR/stats`.

To run, execute the program `mstat_converter` with no arguments.

The statistics converter program does not clear the database before converting. However, if there are statistics in the database and the statistics files from the same period, the converter overwrites the database information with the information from the statistics files.

Accelerators

Moab can integrate with the TORQUE resource manager to discover, report, schedule, and submit workload to various accelerator architectures (such as NVIDIA GPUs or Intel® Xeon Phi™ co-processor architecture) for parallel processing. See the topics below for specific information.

- Scheduling GPUs
  - Using GPUs with NUMA
  - NVIDIA GPUs

Scheduling GPUs

In TORQUE 2.5.4 and later, users can request GPUs on a node at job submission by specifying a nodes resource request, using the qsub -l option. The number of GPUs a node has must be specified in the nodes file. The GPU is then reported in the output of pbsnodes:

```
napali
state = free
np = 2
ntype = cluster
status = rectime=1288888871,varattr=,jobs=,state=free,netload=1606207294,gres=tom:!
/home/dbeer/dev/scripts/dynamic_
resc.sh,loadave=0.10,ncpus=2,physmem=3091140kb,availmem=32788032348kb,
totmem=34653576492kb,idletime=4983,nusers=3,nsessions=14,sessions=3136 1805 2380 2428
1161 3174 3184
3191 3209 3228 3272 3333 20560 32371,uname=Linux napali 2.6.32-25-generic #45-Ubuntu
SMP Sat Oct 16 19:52:42
UTC 2010 x86_64,opsys=linux
mom_service_port = 15002
mom_manager_port = 15003
gpus = 1
```

The `$PBS_GPUFILE` has been created to include GPU awareness. The GPU appears as a separate line in `$PBS_GPUFILE` and follows this syntax:

```
<hostname>-gpu<index>
```

If a job were submitted to run on a server called "napali" (the submit command would look something like: qsub test.sh -l nodes=1:ppn=2:gpus=1), the `$PBS_GPUFILE` would contain:

```
napali-gpu0
```

It is left up to the job's owner to make sure that the job executes properly on the GPU. By default, TORQUE treats GPUs exactly the same as ppn (which corresponds to CPUs).

Related topics

- Using GPUs with NUMA
- NVIDIA GPUs

Using GPUs with NUMA

The pbs_server requires awareness of how the MOM is reporting nodes since there is only one MOM daemon and multiple MOM nodes. Configure the `server_priv/nodes` file with the `num_node_boards` and `numa_gpu_node_str` attributes. The attribute `num_node_boards` tells pbs_server how many NUMA nodes are reported by the MOM. If each NUMA node has the same number of GPUs, add the total number of GPUs to the nodes file. Following is an example of how to configure the nodes file with `num_node_boards`:

```
numahost gpus=12 num_node_boards=6
```

This line in the nodes file tells pbs_server there is a host named numahost and that it has 12 GPUs and 6 nodes. The pbs_server divides the value of GPUs (12) by the value for `num_node_boards` (6) and determines there are 2 GPUs per NUMA node.

In this example, the NUMA system is uniform in its configuration of GPUs per node board, but a system does not have to be configured with the same number of GPUs per node board. For systems with non-uniform GPU distributions, use the attribute `numa_gpu_node_str` to let pbs_server know where GPUs are located in the cluster.

If there are equal numbers of GPUs on each NUMA node, you can specify them with a string. For example, if there are 3 NUMA nodes and the first has 0 GPUs, the second has 3, and the third has 5, you would add this to the nodes file entry:

```
numa_gpu_node_str=0,3,5
```

In this configuration, pbs_server knows it has three MOM nodes and the nodes have 0, 3s, and 5 GPUs respectively. Note that the attribute gpus is not used. The gpus attribute is ignored because the number of GPUs per node is specifically given.

In TORQUE 3.0.2 or later, qsub supports the mapping of `-l gpus=X` to `-l gres=gpus:X`. This allows users who are using NUMA systems to make requests such as `-l ncpus=20,gpus=5` (or `-l ncpus=20:gpus=5`)indicating they are not concerned with the GPUs in relation to the NUMA nodes they request; they only want a total of 20 cores and 5 GPUs.

Related topics

- Scheduling GPUs
- NVIDIA GPUs

NVIDIA GPUs

The `pbs_mom` file can now query for GPU hardware information and report status to the pbs_server. gpustatus will appear in pbsnodes output. New commands allow for setting GPU modes and for resetting GPU ECC error counts.

This feature is only available in TORQUE 2.5.6, 3.0.2, and later.

This document assumes that you have installed the NVIDIA CUDA ToolKit and the NVIDIA development drivers on a compute node with an NVIDIA GPU. (Both can be downloaded from http://developer.nvidia.com/category/zone/cuda-zone).

You will want to download the latest version if you run into problems compiling.

If the pbs_server does not have GPUs, it only needs to be configured with `--enable-nvidia-gpus`. All other systems that have NVIDIA GPUs will need:

- --enable-nvidia-gpus
- --with-nvml-include=DIR (include path for nvml.h)

  `nvml.h` is only found in the NVIDIA CUDA ToolKit.

- --with-nvml-lib=DIR (*lib path for libnvidia-ml)

Systems that have NVIDIA GPUs require the following:

Server

```
./configure --with-debug --enable-nvidia-gpus
```

Compute nodes (with NVIDIA GPUs)

```
./configure --with-debug --enable-nvidia-gpus --with-nvml-lib=/usr/lib64 --with-nvml-
include=/cuda/NVML
```

If all of the compute nodes have the same hardware and software configuration, you can choose to compile on one compute node and then run make packages.

```
> make packages
Building ./torque-package-clients-linux-x86_64.sh ...
Building ./torque-package-mom-linux-x86_64.sh ...
Building ./torque-package-server-linux-x86_64.sh ...
Building ./torque-package-gui-linux-x86_64.sh ...
Building ./torque-package-devel-linux-x86_64.sh ...
Done.
```

The package files are self-extracting packages that can be copied and executed on your production machines. (Use `--help` for options.)

When updating, it is good practice to stop the pbs_server and make a backup of the TORQUE home directory. You will also want to backup the output of qmgr -c 'p s'. The update will only overwrite the binaries.

If you move GPU cards to different slots, you must restart pbs_server in order for TORQUE to recognize the drivers as the same ones in different locations rather than 2 new, additional drivers.

For further details, see these topics:

- **TORQUE configuration** on page 797
- **GPU modes for NVIDIA 260.x driver** on page 798
- **GPU Modes for NVIDIA 270.x driver** on page 798
- **gpu_status** on page 799
- **New NVIDIA GPU support** on page 799

TORQUE configuration

There are three configuration (./configure) options available for use with Nvidia GPGPUs:

- --enable-nvidia-gpus
- --with-nvml-lib=DIR
- --with-nvml-include=DIR

`--enable-nvidia-gpus` is used to enable the new features for the Nvidia GPGPUs. By default, the pbs_moms use the `nvidia_smi` command to interface with the Nvidia GPUs.

```
./configure --enable-nvidia-gpus
```

To use the NVML (NVIDIA Management Library) API instead of nvidia-smi, configure TORQUE using `--with-nvml-lib=DIR` and `--with-nvml-include=DIR`. These commands specify the location of the `libnvidia-ml` library and the location of the `nvml.h` include file.

```
./configure -with-nvml-lib=/usr/lib
--with-nvml-include=/usr/local/cuda/Tools/NVML
server_priv/nodes:
node001 gpus=1
node002 gpus=4
…
pbsnodes -a
node001
    …
    gpus = 1
...
```

By default, when TORQUE is configured with `--enable-nvidia-gpus` the `$TORQUE_HOME/nodes` file is automatically updated with the correct GPU count for each MOM node. See the TORQUE documentation on qgpumode for additional information.

GPU modes for NVIDIA 260.x driver

- 0 – Default - Shared mode available for multiple processes
- 1 – Exclusive - Only one COMPUTE thread is allowed to run on the GPU
- 2 – Prohibited - No COMPUTE contexts are allowed to run on the GPU

GPU Modes for NVIDIA 270.x driver

- 0 – Default - Shared mode available for multiple processes
- 1 – Exclusive Thread - Only one COMPUTE thread is allowed to run on the GPU (v260 exclusive)
- 2 – Prohibited - No COMPUTE contexts are allowed to run on the GPU
- 3 – Exclusive Process - Only one COMPUTE process is allowed to run on the GPU

gpu_status

```
root@gpu:~# pbsnodes gpu
gpu
...
    gpus = 2
    gpu_status = gpu[1]=gpu_id=0:6:0;gpu_product_name=Tesla
    C2050;gpu_display=Disabled;gpu_pci_device_id=6D110DE;gpu_pci_location_id=0:6:0;
    gpu_fan_speed=54 %;gpu_memory_total=2687 Mb;gpu_memory_used=74
Mb;gpu_mode=Default;gpu_state=Unallocated;gpu_utilization=96
%;gpu_memory_utilization=10
%;gpu_ecc_mode=Enabled;gpu_single_bit_ecc_errors=0;gpu_double_bit_ecc_errors=
0;gpu_temperature=88 C,gpu[0]=gpu_id=0:5:0;gpu_product_name=Tesla
C2050;gpu_display=Enabled;gpu_pci_device_id=6D110DE;gpu_pci_location_id=0:5:0;
gpu_fan_speed=66 %;gpu_memory_total=2687 Mb;gpu_memory_used=136
Mb;gpu_mode=Default;gpu_state=Unallocated;gpu_utilization=96
%;gpu_memory_utilization=10
%;gpu_ecc_mode=Enabled;gpu_single_bit_ecc_errors=0;
gpu_double_bit_ecc_errors=0;gpu_temperature=86 C,driver_ver=270.41.06,timestamp=Wed
May 4 13:00:35
2011
```

New NVIDIA GPU support

qsub allows specifying required compute mode when requesting GPUs. If no GPU mode is requested, it will default to "exclusive" for Nvidia driver version 260 or "exclusive_thread" for NVIDIA driver version 270 and above.

- qsub -l nodes=1:ppn=1:gpus=1
- qsub -l nodes=1:gpus=1
- qsub -l nodes=1:gpus=1:exclusive_thread
- qsub -l nodes=1:gpus=1:exclusive_process
- qsub -l nodes=1:gpus=1:reseterr
- qsub -l nodes=1:gpus=1:reseterr:exclusive_thread (exclusive_thread:reseterr)
- qsub -l nodes=1:gpus=1:reseterr:exclusive_process

Related topics

- **Scheduling GPUs** on page 795
- **Using GPUs with NUMA** on page 795

GPU Metrics

GPU metrics can be collected for nodes that:

- Have one or more GPUs.
- Run TORQUE 2.5.x or later.
- Use NVIDIA drivers v260.x or v270.x.

GPU metric tracking must be enabled in `moab.cfg`:

```
RMCFG[torque]  flags=RECORDGPUMETRICS
```

There is one GPU metric for all GPU devices within a node (gpu_timestamp) and nine GPU metrics for each GPU device within a node. If the maximum GPU devices within a node is 4, you must increase the MAXGMETRIC value in `moab.cfg` by (maxgpudevices x gpumetrics) + 1. In this case, the formula is (4 x 9) + 1 = 37, so whatever the **MAXGMETRIC** value is, it must be increased by 37. This way, when enabling GPU metrics recording, Moab has enough GMETRIC types to accommodate the GPU metrics.

GPU Metrics Map

The GPU metric names map is as follows (where *X* is the GPU number):

Metric name as returned by pbsnodes	GMETRIC name as stored in Moab	Metric output
timestamp	gpu_timestamp The gpu_timestamp metric is global to all GPUs on the node and indicates the last time the driver collected information on the GPUs.	The time data was collected in epoch time
gpu_fan_speed	gpuX_fan	The current fan speed as a percentage
gpu_memory_total	gpuX_mem	The total GPU memory in megabytes
gpu_memory_used	gpuX_usedmem	The total used GPU memory in megabytes
gpu_utilization	gpuX_util	The GPU capability currently in use as a percentage
gpu_memory_utilization	gpuX_memutil	The GPU memory currently in use as a percentage
gpu_ecc_mode	gpuX_ecc	Whether ECC is enabled or disabled
gpu_single_bit_ecc_errors	gpuX_ecc1err	The total number of EEC single-bit errors since the last counter reset

Metric name as returned by pbsnodes	GMETRIC name as stored in Moab	Metric output
gpu_double_bit_ecc_errors	gpuX_ecc2err	The total number of EEC double-bit errors since the last counter reset
gpu_temperature	gpuX_temp	The GPU current temperature in Celsius

Example 3-157: GPU example

```
$ mdiag -n -v --xml

<Data>
<node AGRES="GPUS=2;"
AVLCLASS="[test 8][batch 8]"
CFGCLASS="[test 8][batch 8]"
GMETRIC="gpu1_fan:59.00,gpu1_mem:2687.00,gpu1_usedmem:74.00,gpu1_util:94.00,gpu1_
memutil:9.00,gpu1_ecc:0.00,gpu1_ecc1err:0.00,gpu1_ecc2err:0.00,gpu1_temp:89.00,gpu0_
fan:70.00,gpu0_mem:2687.00,gpu0_usedmem:136.00,gpu0_util:94.00,gpu0_memutil:9.00,gpu0_
ecc:0.00,gpu0_ecc1err:0.00,gpu0_ecc2err:0.00,gpu0_temp:89.00,gpu_
timestamp:1304526680.00"
GRES="GPUS=2;"
LASTUPDATETIME="1304526518" LOAD="1.050000"
MAXJOB="0" MAXJOBPERUSER="0" MAXLOAD="0.000000" NODEID="gpu"
NODEINDEX="0" NODESTATE="Idle" OS="linux" OSLIST="linux"
PARTITION="makai" PRIORITY="0" PROCSPEED="0" RADISK="1"
RAMEM="5978" RAPROC="7" RASWAP="22722" RCDISK="1" RCMEM="5978"
RCPROC="8" RCSWAP="23493" RMACCESSLIST="makai" SPEED="1.000000"
STATMODIFYTIME="1304525679" STATTOTALTIME="315649"
STATUPTIME="315649"></node>
</Data>
```

Intel® Xeon Phi™ Coprocessor Configuration

Intel Many-Integrated Cores (MIC) architecture configuration

If you use an Intel Many-Integrated Cores (MIC) architecture-based product (e.g., Intel Xeon Phi™) in your cluster for parallel processing, you must configure TORQUE to detect them.

Prerequisites

- TORQUE 4.2 or later
- If you set up TORQUE using auto-detection and intend to get the MIC-based device status report, you must build pbs_mom on a system that has the lower-level API libraries for the MIC-based device(s) installed. Additionally, every MOM built with `--enable-mics` and running on a compute node must already have the lower-level API libraries installed on the node. Note that the library is called `coi_host`. You must obtain the API libraries from Intel.

Setup Options

There are two ways to configure MIC-based devices with TORQUE: (1) manually and (2) by auto-detection.

Manual configuration

- Add `mics=X` to the nodes file for the appropriate nodes.

```
napali np=12 mics=2
```

Auto-detect

When you use auto-detection, pbs_mom discovers the MIC-based devices and reports them to pbs_server.

- At build time, add `--enable-mics` to the configure line.

```
./configure --enable-mics <other configure options>
```

Validating the configuration

TORQUE

pbsnodes

Example 3-158: pbsnodes output

```
slesmic
       state = free
       np = 100
       ntype = cluster
       status =
rectime=1347634381,varattr=,jobs=,state=free,netload=7442004852,gres=,loadave=0.00,ncp
us=32,physmem=65925692kb,availmem=66531344kb,totmem=68028984kb,idletime=59059,nusers=2
,nsessions=8,sessions=4387 4391 4392 4436 4439 4443 4459 100395,uname=Linux slesmic
3.0.13-0.27-default #1 SMP Wed Feb 15 13:33:49 UTC 2012 (d73692b) x86_64,opsys=linux
       mom_service_port = 15002
       mom_manager_port = 15003
       mics = 2
       mic_status = mic[1]=mic_id=8796;num_cores=61;num_threads=244;physmem=8065748992;free_
physmem=7854972928;swap=0;free_swap=0;max_frequency=1090;isa=COI_ISA_
KNC;load=0.000000;normalized_load=0.000000;,mic[0]=mic_id=8796;num_cores=61;num_
threads=244;physmem=8065748992;free_physmem=7872712704;swap=0;free_swap=0;max_
frequency=1090;isa=COI_ISA_KNC;load=0.540000;normalized_load=0.008852;

rhmic.ac
       state = free
       np = 100
       ntype = cluster
       status =
rectime=1347634381,varattr=,jobs=,state=free,netload=3006171583,gres=,loadave=0.00,ncp
us=32,physmem=65918268kb,availmem=66901588kb,totmem=67982644kb,idletime=59477,nusers=2
,nsessions=2,sessions=3401 29320,uname=Linux rhmic.ac 2.6.32-220.el6.x86_64 #1 SMP Tue
Dec 6 19:48:22 GMT 2011 x86_64,opsys=linux
       mom_service_port = 15002
       mom_manager_port = 15003
       mics = 1
       mic_status = mic[0]=mic_id=8796;num_cores=61;num_threads=244;physmem=8065748992;free_
physmem=7872032768;swap=0;free_swap=0;max_frequency=1090;isa=COI_ISA_
KNC;load=0.540000;normalized_load=0.008852;<mic_status>;
```

Moab

mdiag -n -v

Example 3-159: mdiag -n -v output

```
$ mdiag -n -v
compute node summary
Name                    State    Procs      Memory         Disk          Swap
Speed   Opsys   Arch Par   Load Classes                        Features

hola                     Idle    4:4      8002:8002        1:1       10236:13723
1.00   linux      - hol   0.24 [batch]                        -
GRES=MICS:2,
-----                     ---    4:4      8002:8002        1:1       10236:13723

Total Nodes: 1  (Active: 0  Idle: 1  Down: 0)
```

checknode -v

Example 3-160: checknode output

```
$ checknode slesmic
node slesmic

State:      Idle  (in current state for 00:00:16)
Configured Resources: PROCS: 100  MEM: 62G  SWAP: 64G  DISK: 1M  MICS: 2
Utilized   Resources: SWAP: 1581M
Dedicated  Resources: ---
Generic Metrics:    mic1_mic_id=8796.00,mic1_num_cores=61.00,mic1_num_
threads=244.00,mic1_physmem=8065748992.00,mic1_free_physmem=7854972928.00,mic1_
swap=0.00,mic1_free_swap=0.00,mic1_max_frequency=1090.00,mic1_load=0.12,mic1_
normalized_load=0.00,mic0_mic_id=8796.00,mic0_num_cores=61.00,mic0_num_
threads=244.00,mic0_physmem=8065748992.00,mic0_free_physmem=7872679936.00,mic0_
swap=0.00,mic0_free_swap=0.00,mic0_max_frequency=1090.00
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   0.000
Classes:    [batch]
RM[napali]* TYPE=PBS
EffNodeAccessPolicy: SHARED

Total Time: 3:45:43  Up: 3:45:43 (100.00%)  Active: 00:00:00 (0.00%)

Reservations:
  ---
```

Job submission

Syntax

Example 3-161: Request MIC-based device(s) in qsub

```
qsub .... -l nodes=X:mics=Y
```

Because these resources are delimited with a colon, this command requests a job with X nodes and Y mics per task. If you run the same command and delimit the resources with a comma (`qsub .... -l nodes=X,mics=Y`), you request a job with X nodes and Y mics per job.

qstat -f

Example 3-162: qstat -f output

```
Job Id: 5271.napali
Job_Name = STDIN
Job_Owner = dbeer@napali
job_state = Q
queue = batch
server = napali
Checkpoint = u
ctime = Fri Sep 14 08:56:33 2012
Error_Path = napali:/home/dbeer/dev/private-torque/trunk/STDIN.e5271
Hold_Types = n
Join_Path = oe
Keep_Files = n
Mail_Points = a
mtime = Fri Sep 14 08:56:33 2012
Output_Path = napali:/home/dbeer/dev/private-torque/trunk/STDIN.o5271
Priority = 0
qtime = Fri Sep 14 08:56:33 2012
Rerunable = True
Resource_List.neednodes = 1:mics=1
Resource_List.nodect = 1
Resource_List.nodes = 1:mics=1
substate = 10
Variable_List = PBS_O_QUEUE=batch,PBS_O_HOME=/home/dbeer,
       PBS_O_LOGNAME=dbeer,
       PBS_O_PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
       in:/usr/games,PBS_O_MAIL=/var/mail/dbeer,PBS_O_SHELL=/bin/bash,
       PBS_O_LANG=en_US.UTF-8,
       PBS_O_SUBMIT_FILTER=/usr/local/sbin/torque_submitfilter,
       PBS_O_WORKDIR=/home/dbeer/dev/private-torque/trunk,PBS_O_HOST=napali,
       PBS_O_SERVER=napali
euser = dbeer
egroup = company
queue_rank = 3
queue_type = E
etime = Fri Sep 14 08:56:33 2012
submit_args = -l nodes=1:mics=1
fault_tolerant = False
job_radix = 0
submit_host = napali
```

checkjob -v

Example 3-163: checkjob -v output

```
dthompson@mahalo:~/dev/moab-test/trunk$ checkjob -v 2
job 2 (RM job '2.mahalo')

AName: STDIN
State: Idle
Creds:  user:dthompson  group:dthompson  class:batch
WallTime:   00:00:00 of 1:00:00
SubmitTime: Thu Sep 13 17:06:06
(Time Queued  Total: 00:00:24  Eligible: 00:00:02)

TemplateSets:  DEFAULT
Total Requested Tasks: 1

Req[0]  TaskCount: 1  Partition: ALL
Dedicated Resources Per Task: PROCS: 1  MICS: 1

...
```

Intel® Xeon Phi™ Co-processor Metrics

Intel Many-Integrated Cores (MIC) architecture-based device (e.g., Intel Xeon Phi™) metrics can be collected for nodes that:

- Have one or more MIC-based devices.
- Run TORQUE 4.2.x or later.
- Run Moab 7.2 or later.

MIC-based device metric tracking must be enabled in `moab.cfg`:

```
RMCFG[torque]  flags=RECORDMICMETRICS
```

There are 11 metrics for each MIC-based device within a node. If the maximum MIC-based devices within a node is 4, you must increase the MAXGMETRIC value in `moab.cfg` by (*maxmicdevices* x *micmetrics*). In this case, the formula is (4 x 11) = 44, so whatever the **MAXGMETRIC** value is, it must be increased by 44. This way, when enabling MIC-based device metrics recording, Moab has enough GMETRIC types to accommodate the additional metrics.

MIC-based Metrics Map

The MIC-based metric names map is as follows (where *X* is the MIC-based device number):

Metric name as returned by pbsnodes	GMETRIC name as stored in Moab	Metric output
mic_id	micX_mic_id	The ID of the MIC-based device
num_cores	micX_num_cores	The number of cores in the MIC-based device
num_threads	micX_num_threads	The number of hardware threads on the MIC-based device
physmem	micX_physmem	The total physical memory in the MIC-based device
free_physmem	micX_free_physmem	The available physical memory in the MIC-based device
swap	micX_swap	The total swap space on the MIC-based device
free_swap	micX_free_swap	The unused swap space on the MIC-based device
max_frequency	micX_max_frequency	The maximum frequency speed of any core in the MIC-based device
isa	micX_isa	The hardware interface type of the MIC-based device
load	micX_load	The total current load of the MIC-based device
normalized_load	micX_normalized_load	The normalized load of the MIC-based device (total load divided by number of cores in the MIC-based device)

Preemption

About preemption

Sites possess workloads of varying importance, and users may want to run jobs with higher priorities before jobs with lower priorities. This can be done by using preemption. Preemption is simply the process by which a higher-priority job can take the place of a lower-priority job. You can also use preemption for optimistic scheduling and development job support.

This section explains how to configure and use preemption. **Simple example of preemption** on page 827 offers a basic introduction and contains examples to help you get started using preemption. The other sections offer more explanation and information about what you can do with preemption and contain some best practices that will help you avoid the need for troubleshooting in the future.

While this section does not explain every possible preemption configuration, it does prescribe the best practices for setting up and using preemption with your system. It is recommended that you follow the established instructions contained in this section.

> Preemption does not work with dynamic provisioning.

> Neither *SPANEVENLY* nor *DELAY* values of the NODESETPLUS parameter will work with multi-req jobs or preemption.

> Do not allow preemption with interactive jobs unless **PREEMPTPOLICY** is set to *CANCEL*. (For more information, see **Canceling jobs with preemption** on page 809.)

Tasks associated with preemption:

The following sections include information about each type of preemption, their different usage benefits, and any configurations and settings needed to use them.

- **Canceling jobs with preemption** on page 809
- **Checkpointing jobs with preemption** on page 812
- **Requeueing jobs with preemption** on page 813
- **Suspending jobs with preemption** on page 816
- **Using owner preemption** on page 819
- **Using QoS preemption** on page 823

Preemption references:

These sections contain information that you can use as references for the preemption tasks.

- **Manual preemption commands** on page 824
- **Preemption flags** on page 825
- **PREEMPTPOLICY types** on page 827
- **Simple example of preemption** on page 827
- **Testing and troubleshooting preemption** on page 831

Related topics

- **Optimizing Scheduling Behavior – Backfill and Node Sets** on page 508

How-to's

Canceling jobs with preemption

Context

CANCEL is one of the **PREEMPTPOLICY** types (for more information, see **PREEMPTPOLICY types** on page 827). The *CANCEL* attribute cancels active jobs, regardless of any **JOBFLAGS** (such as *REQUEUEABLE* or *SUSPENDABLE*). (For more information, see **Job Flags** on page 153.)

For information about *PREEPMPTEE* and *PREEMPTOR* flags, see **Preemption flags** on page 825

> You should not allow preemption with interactive jobs unless **PREEMPTPOLICY** is set to *CANCEL*.

The following outlines some benefits of using *CANCEL* and also lists some things you should be aware of if you choose to use it.

Advantages:

This attribute is the easiest to configure and use.

Cautions:

Canceled jobs are not automatically restarted or requeued. Users must resubmit canceled jobs.

To preempt jobs using CANCEL

1. Make the following configurations to the `moab.cfg` file:
    a. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)
    b. Make sure that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*, which is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 831).
    c. Set **PREEMPTPOLICY** to *CANCEL* (for more information, see **PREEMPTPOLICY types** on page 827).
    d. Make sure that the *PREEMPTEE* job has a lower priority than the *PREEMPTOR* job (for more information, see **Preemption flags** on page 825).

    For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY CANCEL

QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

2. Submit a job to the preemptee QoS (`test1`). For example:

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test1 -l procs=128
```

(Optional) Examine the following output for showq:

```
Moab.7
[john@g06]# showq

active jobs------------------------
JOBID      USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.7     john        Running    128       00:01:59      Thu Nov 10 12:28:44

1 active job      128 of 128 processors in use by local jobs (100.00%)
                  2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID      USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID      USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

3. Now submit a job to the preemptor QoS (`test2`). For example:

```
[john@g06]$ echo sleep 120 | msub -l procs=128,walltime=120 -l qos=test2
```

(Optional) Examine the following output for showq:

```
Moab.8
[john@g06]# showq

active jobs------------------------
JOBID      USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.7     john        Canceling  128       00:01:56      Thu Nov 10 12:28:44
Moab.8     john        Running    128       00:02:00      Thu Nov 10 12:28:48

2 active jobs 128 of 128 processors in use by local jobs (100.00%)
              2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID      USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID      USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total jobs: 2
```

Note that `test1` is canceled when `test2` is submitted.

(Optional) Examine the checkjob outputs for these two jobs:

```
[john@g06]$ checkjob Moab.9
job Moab.9

State: Removed
Completion Code: -1 Time: Thu Nov 10 12:28:48
Creds: user:john group:john qos:test1
WallTime: 00:00:02 of 00:02:00
SubmitTime: Thu Nov 10 12:28:44
(Time Queued Total: 00:00:07 Eligible: 00:00:00)

Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses
NodeCount: 2

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.UFe8sQ

StartCount: 1
Flags: GLOBALQUEUE,PROCSPECIFIED
Attr: PREEMPTEE
StartPriority: 100
```

Note that the preempted job has been removed.

```
[john@g06]$ checkjob Moab.10
job Moab.10

State: Running
Creds: user:john group:john qos:test2
WallTime: 00:00:00 of 00:02:00
SubmitTime: Thu Nov 10 12:36:31
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Thu Nov 10 12:28:48
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.CZavjU

StartCount: 1
Flags: HASPREEMPTED,PREEMPTOR,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 10000
Reservation 'Moab.10' (-00:00:07 -> 00:01:53 Duration: 00:02:00)
```

Related topics

- **Suspending jobs with preemption** on page 816
- **Checkpointing jobs with preemption** on page 812

- **Requeueing jobs with preemption** on page 813
- **Preemption flags** on page 825
- **About preemption** on page 807
- **PREEMPTPOLICY types** on page 827
- **Testing and troubleshooting preemption** on page 831

Checkpointing jobs with preemption

Context

CHECKPOINT is one of the **PREEMPTPOLICY** types (for more information, see **PREEMPTPOLICY types** on page 827). For systems that allow checkpointing, the *CHECKPOINT* attribute allows a job to save its current state and either terminate or continue running. A checkpointed job may restart at any time and resume execution from its most recent checkpoint.

You can tune checkpointing behavior on a per-resource manager-basis by setting the **CHECKPOINTSIG** and **CHECKPOINTTIMEOUT** attributes of the **RMCFG** parameter.

For information about *PREEPMPTEE* and *PREEMPTOR* flags, see **Preemption flags** on page 825

The following outlines some benefits of using *CHECKPOINT* and also lists some things you should be aware of if you choose to use it.

Advantages:

This attribute allows you to restart a job from its last checkpoint.

Cautions:

Jobs tend to take longer to complete when you use *CHECKPOINT*.

To preempt jobs using CHECKPOINT

Make the following configurations to the `moab.cfg` file:

1. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)(This locks the job on a node and keeps trying to preempt.)
2. Make sure that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*, which is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 831).
3. Set **PREEMPTPOLICY** to *CHECKPOINT* (for more information, see **PREEMPTPOLICY types** on page 827).
4. Make sure that the *PREEMPTEE* job has a lower priority than the *PREEMPTOR* job (for more information, see **Preemption flags** on page 825).

For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY CHECKPOINT

QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

Related topics

- **Suspending jobs with preemption** on page 816
- **Requeueing jobs with preemption** on page 813
- **Canceling jobs with preemption** on page 809
- **Preemption flags** on page 825
- **About preemption** on page 807
- **PREEMPTPOLICY types** on page 827
- **Testing and troubleshooting preemption** on page 831

Requeueing jobs with preemption

Context

REQUEUE is one of the **PREEMPTPOLICY** types (for more information, see **PREEMPTPOLICY types** on page 827). The *REQUEUE* value terminates active jobs and returns them to the job queue in an idle state.

For information about *PREEMPMPTEE* and *PREEMPTOR* flags, see **Preemption flags** on page 825

The following outlines some benefits of using *REQUEUE* and also lists some things you should be aware of if you choose to use it.

Advantages:

- Jobs are automatically resubmitted into the job queue.

Cautions:

- A job gets resubmitted in the job queue at the same priority it had when Moab originally started it (i.e., the job does not jump ahead in the queue).
- Jobs start over from the beginning.

> You must mark a job as *RESTARTABLE* if you want it to requeue. If you do not, the job will be canceled when it is preempted.
>
> If supported by the resource manager, you can set the *RESTARTABLE* job flag when submitting the job by using the msub -r option. Otherwise, use the **JOBFLAGS** attribute of the associated class or QoS credential, as in this example:
>
> ```
> CLASSCFG[low] JOBFLAGS=RESTARTABLE
> ```
>
> For more information, see **Job Flags** on page 153.

To preempt jobs using REQUEUE

1. Make the following configurations to the `moab.cfg` file:
   a. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)

b. Make sure that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*, which is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 831).

c. Set PREEMPTPOLICY to *REQUEUE* (for more information, see **PREEMPTPOLICY types** on page 827).

d. Make sure that the *PREEMPTEE* job has a lower priority than the *PREEMPTOR* job (for more information, see **Preemption flags** on page 825).

For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY REQUEUE

QOSCFG[test1] QFLAGS=PREEMPTEE JOBFLAGS=RESTARTABLE MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

2. Submit a job to the preemptee QoS (`test1`). For example:

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test1 -l procs=128
```

(Optional) Examine the following output for showq:

```
Moab.1
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING      STARTTIME
Moab.1    john        Running    128       00:09:59       Wed Nov 9 15:56:33

1 active job     128 of 128 processors in use by local jobs (100.00%)
                 2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

3. Now submit a job to the preemptor QoS (`test2`). For example:

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test2 -l procs=128
```

(Optional) Examine the following output for showq and checkjob:

```
Moab.2
[john@g06]# showq

active jobs------------------------
JOBID      USERNAME      STATE      PROCS    REMAINING      STARTTIME
Moab.2     john          Running    128      00:09:59       Wed Nov  9 15:56:47

1 active job 128 of 128 processors in use by local jobs (100.00%)
             2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID      USERNAME     STATE       PROCS    WCLIMIT        QUEUETIME
Moab.1     john         Idle        128      00:10:00       Wed Nov  9 15:56:33

1 eligible job

blocked jobs-----------------------
JOBID      USERNAME     STATE       PROCS    WCLIMIT        QUEUETIME

0 blocked jobs

Total jobs: 2
```

```
[john@g06]# checkjob Moab.2
job Moab.2

State: Running
Creds: user:john group:john qos:test2
WallTime: 00:02:04 of 00:10:00
SubmitTime: Wed Nov  9 15:56:46
(Time Queued Total: 00:00:01 Eligible: 00:00:00)

StartTime: Wed Nov  9 15:56:47
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses
NodeCount: 2

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.ELoX5Q

StartCount: 1
Flags: HASPREEMPTED,PREEMPTOR,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 10000
Reservation 'Moab.2' (-00:02:21 -> 00:07:39 Duration: 00:10:00)
```

Related topics

- **Suspending jobs with preemption** on page 816
- **Checkpointing jobs with preemption** on page 812
- **Canceling jobs with preemption** on page 809
- **Preemption flags** on page 825
- **About preemption** on page 807

- **PREEMPTPOLICY types** on page 827
- **Testing and troubleshooting preemption** on page 831

Suspending jobs with preemption

Context

SUSPEND is one of the **PREEMPTPOLICY** types (for more information, see **PREEMPTPOLICY types** on page 827). The *SUSPEND* attribute causes active jobs to stop executing, but to remain in memory on the allocated compute nodes.

For information about *PREEMPMPTEE* and *PREEMPTOR* flags, see **Preemption flags** on page 825

The following outlines some benefits of using *SUSPEND* and also lists some things you should be aware of if you choose to use it.

Advantages:

- The job remains in memory on the allocated compute nodes.
- Using *SUSPEND* frees up processor resources.
- The job can restart where it left off before it was suspended.

Cautions:

- There is a possibility that having multiple suspended jobs on a compute node will crash the swap.
- Moab tracks only *requested* memory of active jobs (not *used* memory). The swap can crash if the job uses a lot of memory and Moab starts other jobs.
- Suspended jobs do not relinquish their licenses.

> You must mark a job as *SUSPENDABLE* if you want it to suspend. If you do not, the job will be requeued or canceled when it is preempted.
>
> If supported by the resource manager, you can set the job *SUSPENDABLE* flag when submitting the job by using the msub -r option. Otherwise, use the **JOBFLAGS** attribute of the associated class or QoS credential, as in this example:
>
> ```
> CLASSCFG[low] JOBFLAGS=SUSPENDABLE
> ```
>
> For more information, see **Job Flags** on page 153.

To preempt jobs using SUSPEND

When you use *SUSPEND*, you must increase your **JOBRETRYTIME**. By default, **JOBRETRYTIME** is set to 60 seconds, but when you use *SUSPEND*, it is recommended that you increase the time to 300 seconds (5 minutes).

1. Make the following configurations to the `moab.cfg` file:

   a. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)

   b. Make sure that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*, which is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 831).

   c. Set **PREEMPTPOLICY** to *SUSPEND* (for more information, see **PREEMPTPOLICY types** on page 827).

   d. For the *PREEMPTEE* job, set **JOBFLAGS**=*RESTARTABLE,SUSPENDABLE*.

   e. Make sure that the *PREEMPTEE* job has a lower priority than the *PREEMPTOR* job (for more information, see **Preemption flags** on page 825).

   For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY SUSPEND

QOSCFG[test1] QFLAGS=PREEMPTEE JOBFLAGS=RESTARTABLE,SUSPENDABLE MEMBERULIST=john
PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

2. Submit a job to the preemptee QoS (`test1`). For example:

```
[john@g06]$ echo sleep 120 | msub -l procs=128,walltime=120 -l qos=test1
```

   (Optional) Examine the output for showq:

```
Moab.7
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.7    john        Running    128       00:01:59      Thu Nov 10 12:28:44

1 active job      128 of 128 processors in use by local jobs (100.00%)
                  2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

3. Now submit a job to the preemptor QoS (`test2`). For example:

```
[john@g06]$ echo sleep 120 | msub -l procs=128,walltime=120 -l qos=test2
```

(Optional) Examine the output for showq:

```
Moab.8
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.7    john        Suspended  128       00:01:56      Thu Nov 10 12:28:44
Moab.8    john        Running    128       00:02:00      Thu Nov 10 12:28:48

2 active jobs 128 of 128 processors in use by local jobs (100.00%)
              2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total jobs: 2
```

Note that when a job is suspended, it stays in the output of showq. This is normal behavior for a suspended job. Moab should only suspend a job once.

4. (Optional) Examine the checkjob outputs for these two jobs.

```
[john@g06]$ checkjob Moab.9
job Moab.9

State: Suspended
Creds: user:john group:john qos:test1
WallTime: 00:00:02 of 00:02:00
SubmitTime: Thu Nov 10 12:36:29
(Time Queued Total: 00:00:07 Eligible: 00:00:00)

Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses
NodeCount: 2

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.UFe8sQ

StartCount: 1
Flags: RESTARTABLE,SUSPENDABLE,PREEMPTEE,GLOBALQUEUE,PROCSPECIFIED
Attr: PREEMPTEE
StartPriority: 100
job cannot be resumed: preemption required but job is conditional preemptor with no
targets
BLOCK MSG: non-idle state 'Running' (recorded at last scheduling iteration)
```

```
[john@g06]$ checkjob Moab.10
job Moab.10

State: Running
Creds: user:john group:john qos:test2
WallTime: 00:00:00 of 00:02:00
SubmitTime: Thu Nov 10 12:36:31
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Thu Nov 10 12:36:31
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.CZavjU

StartCount: 1
Flags: HASPREEMPTED,PREEMPTOR,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 10000
Reservation 'Moab.10' (-00:00:07 -> 00:01:53 Duration: 00:02:00)
```

Occasionally, Moab will keep a job from restarting, holding it in a suspended state for a long period of time, if it thinks the job cannot restart. For example, if a job could write to I/O before it was suspended, and now it cannot, Moab would realize the job is unable to start and would leave it in a suspended state.

Related topics

- **Checkpointing jobs with preemption** on page 812
- **Requeueing jobs with preemption** on page 813
- **Canceling jobs with preemption** on page 809
- **Preemption flags** on page 825
- **About preemption** on page 807
- **PREEMPTPOLICY types** on page 827
- **Testing and troubleshooting preemption** on page 831

Using owner preemption

Context

Owner preemption allows jobs submitted by a reservation owner to preempt jobs submitted by other users (for more information, see **Configuring and Managing Reservations** on page 461).

Owner preemption is enabled with the **OWNERPREEMPT** reservation flag.

For information about *PREEMPTEE* and *PREEMPTOR* flags, see **Preemption flags** on page 825

To enable owner preemption

1. Make the following configurations to the `moab.cfg` file:

    a. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)

    b. Make sure that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*, which is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 831).

    c. Set the **PREEMPTPOLICY** type (for more information, see **PREEMPTPOLICY types** on page 827).

    d. Set the *OWNERPREEMPT* flag.

    > Optionally, if you want the owner preemption to override any PREEMPTMINTIME settings for *PREEMPTEE* jobs, you can set the **OWNERPREEMPTIGNOREMINTIME** flag as well.

    e. Specify an owner.

    > If the non-owner job does not have a *RESTARTABLE* or *REQUEUEABLE* flag set, the job will cancel.

    For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY <policy>

SRCFG[myrez] FLAGS=OWNERPREEMPT HOSTLIST=node01
SRCFG[myrez] OWNER=USER:john
SRCFG[myrez] USERLIST=jane,john PERIOD=INFINITY

QOSCFG[test1] QFLAGS=PREEMPTEE JOBFLAGS=restartable MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

2. Submit a job to a user who is not the owner (in this example, `jane`).

```
[jane@g06]$ echo sleep 600 | msub -l walltime=600 -l procs=64
```

    (Optional) Examine the following output for showq and checkjob for `jane`'s job:

```
Moab.1
[jane@g06]$ showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS      REMAINING     STARTTIME
Moab.1    jane        Running    64         00:09:57      Mon Nov 14 12:07:52

1 active job     64 of 64 processors in use by local jobs (100.00%)
                 1 of 1 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

```
root@g06]# checkjob Moab.1
job Moab.1

State: Running
Creds: user:jane group:jane
WallTime: 00:01:02 of 00:10:00
SubmitTime: Mon Nov 14 12:07:52
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Mon Nov 14 12:07:52
Total Requested Tasks: 64

Req[0] TaskCount: 64 Partition: FLEXlm
NodeCount: 1

Allocated Nodes:
[node01:64]


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.FoZfIU

StartCount: 1
Flags: GLOBALQUEUE,PROCSPECIFIED
StartPriority: 1
Reservation 'Moab.1' (-00:01:24 -> 00:08:36 Duration: 00:10:00)
```

3. Now submit a job for the owner (in this example, `john`).

```
[john@g06]$ echo sleep 600 | msub -l walltime=600 -l procs=50
```

```
[john@g06]$ echo sleep 600 | msub -l walltime=600 -l procs=50
```

(Optional) Examine the following output for showq and checkjob for `john`'s job:

```
Moab.2
[john@g06]$ showq

active jobs------------------------
JOBID      USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.1     jane        Canceling  64        00:07:43      Mon Nov 14 12:07:52
Moab.2     john        Running    50        00:09:59      Mon Nov 14 12:10:08

2 active jobs    64 of 64 processors in use by local jobs (100.00%)
                 1 of 1 nodes active (100.00%)

eligible jobs----------------------
JOBID      USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID      USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total jobs: 2
```

Note that jane's job is canceled once john's job is submitted.

```
[john@g06]$ checkjob Moab.2
job Moab.2

State: Running
Creds: user:john group:john
WallTime: 00:00:31 of 00:10:00
SubmitTime: Mon Nov 14 12:10:08
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Mon Nov 14 12:10:08
Total Requested Tasks: 50

Req[0] TaskCount: 50 Partition: FLEXlm
NodeCount: 1

Allocated Nodes:
[node01:50]


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.jf1N4a

StartCount: 1
Flags: HASPREEMPTED,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 1
Reservation 'Moab.2' (-00:00:48 -> 00:09:12 Duration: 00:10:00)
```

Note the new HASPREEMPTED flag.

(Optional) Now look at the showq for jane's job (after):

```
[root@g06]# checkjob Moab.1
job Moab.1

State: Removed
Completion Code: -1 Time: Mon Nov 14 12:10:08
Creds: user:jane group:jane
WallTime: 00:02:47 of 00:10:00
SubmitTime: Mon Nov 14 12:07:52
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

Total Requested Tasks: 64

Req[0] TaskCount: 64 Partition: FLEXlm
NodeCount: 1

Allocated Nodes:
[node01:64]


IWD: /opt/native
Executable: /opt/native/spool/moab.job.FoZfIU

Execution Partition: FLEXlm
Flags: GLOBALQUEUE,PROCSPECIFIED
StartPriority: 0
```

Note that the state is now Removed.

Related topics

- **Preemption flags** on page 825
- **About preemption** on page 807
- **PREEMPTPOLICY types** on page 827
- **Testing and troubleshooting preemption** on page 831

Using QoS preemption

Context

This section breaks down how to configure the `moab.cfg` file to set up preemption with QoS. Using QoS, you can specify preemption rules and control access to preemption privileges by using the QFLAGS *PREEMPTEE* and *PREEMPTOR* credentials. For information about the *PREEMPTEE* and *PREEMPTOR* flags, see **Preemption flags** on page 825.

QoS-based preemption only occurs when the following three conditions are satisfied:

- The preemptor job has the *PREEMPTOR* attribute set.
- The preemptee job has the *PREEMPTEE* attribute set.
- The preemptor job has a higher priority than the preemptee job.

To configure moab.cfg for QoS preemption

1. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)

2. Make sure that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*, which is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 831.
3. If it is not already, set **NODEACCESSPOLICY** to *SHARED*.
4. Set the **PREEMPTPOLICY** policy type (for more information, see **PREEMPTPOLICY types** on page 827).
5. Set up **QFLAGS** to mark jobs as *PREEMPTEE* (a lower-priority job that can be preempted by a higher-priority job), or as *PREEMPTOR* (a higher-priority job that can preempt a lower-priority job). As in the example:

   ```
   QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=<user> PRIORITY=100
   QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=<user> PRIORITY=10000
   ```

   For more information, see **Preemption flags** on page 825.
6. Make sure that the *PREEMPTEE* job has a lower priority than the *PREEMPTOR* job. As in the example:

   ```
   QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=<user> PRIORITY=100
   QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=<user> PRIORITY=10000
   ```

For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY <policy>

QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=<user> PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=<user> PRIORITY=10000
```

Related topics

- **About preemption** on page 807
- **Preemption Management** on page 505
- **Preemption flags** on page 825
- **PREEMPTPOLICY types** on page 827
- **Simple example of preemption** on page 827
- **Testing and troubleshooting preemption** on page 831

References

Manual preemption commands

You can use the **mjobctl** command to manually preempt jobs. The command can modify a job's execution state in the following ways:

Action	Flag	Details
Cancel	-c	Terminate job; remove from queue
Checkpoint	-C	Terminate and checkpoint job leaving job in queue
Requeue	-R	Terminate job; leave in queue
Resume	-r	Resume suspended job
Start (execute)	-x	Start idle job
Suspend	-s	Suspend active job

In general, users are allowed to suspend or terminate jobs they own. Administrators are allowed to suspend, terminate, resume, and execute any queued jobs.

Related topics

- **About preemption** on page 807
- **Testing and troubleshooting preemption** on page 831

Preemption flags

Using QoS, you can specify preemption rules and control access to preemption privileges. This allows you to increase system throughput, improve job response time for specific classes of jobs, or enable various political policies. You enable all policies by specifying some QoS credentials with the QFLAGS *PREEMPTEE*, and others with *PREEMPTOR*.

PREEMPTEE	
Description	Indicates that the job can be preempted by a higher-priority job.
Use	Use for lower-priority jobs that can be preempted.
Notes	This may delay some node actions. When reprovisioning, the system job may expire before the provision action occurs; while the action will still occur, the job will not show it.
Example	`QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=<user> PRIORITY=100`

PREEMPTOR	
Description	Indicates that the job should take priority and preempt any *PREEMPTEE* jobs.
Use	Use for jobs that need to take precedence over lower-priority jobs.
Notes	*PREEMPTOR* jobs, either queued or running, must have a higher priority than *PREEMPTEE* jobs. When you configure job as a *PREEMPTOR*, you should also increase its priority (for details, see **PREEMPTPRIOJOBSELECTWEIGHT** and **PREEMPTRTIMEWEIGHT**).
Example	`QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=<user> PRIORITY=10000`

Additional preemptor and preemptee information

> Preemptor priority plays a big role in preemption. Generally, you should assign the preemptor job a higher priority value than any other queued jobs so that it will move to (or near to) the top of the eligible queue.

You can set the **RESERVATIONPOLICY** parameter to *NEVER*. With this configuration, preemptee jobs can start whenever idle resources become available. These jobs will be allowed to run until a preemptor job arrives, at which point the preemptee jobs are preempted, freeing the resource. This configuration allows near immediate resource access for the preemptor jobs. Using this approach, a cluster can maintain near 100% system utilization while still delivering excellent turnaround time to the most important jobs.

In environments where job checkpointing or job suspension incur significant overhead, you might want to constrain the rate at which job preemption is allowed. You can use the **JOBPREEMPTMINACTIVETIME** parameter to throttle job preemption. In essence, this parameter prevents a newly started or newly resumed job from being eligible for preemption until it has executed for a specified amount of time. Conversely, you can exclude jobs from preemption after they have run for a certain amount of time by using the **JOBPREEMPTMAXACTIVETIME** parameter.

Related topics

- **About preemption** on page 807
- **Using QoS preemption** on page 823
- **Manual preemption commands** on page 824
- **PREEMPTPOLICY types** on page 827
- **Testing and troubleshooting preemption** on page 831

PREEMPTPOLICY types

You can use the **PREEMPTPOLICY** parameter to control how the scheduler preempts a job. This parameter enforces preemption using one of the following methods:

PREEMPTPOLICY type	Description
SUSPEND	Causes active jobs to stop executing, but to remain in memory on the allocated compute nodes.
CHECKPOINT	Saves the current job state and either terminates or continues running the job. A check-pointed job may restart at any time and resume execution from its most recent checkpoint.
REQUEUE	Terminates active jobs and returns them to the job queue in an idle state.
CANCEL	Cancels active jobs.

Each of these methods varies in the level of disruption to the job, *SUSPEND* being the least disruptive and *CANCEL* being the most disruptive.

Moab uses preemption escalation to free up resources. So for example, if the PREEMPTPOLICY is set to *SUSPEND*, Moab uses this method if it is available; however, Moab will escalate it to something potentially more disruptive if necessary to preempt and free up resources.

Related topics

- **Suspending jobs with preemption** on page 816
- **Checkpointing jobs with preemption** on page 812
- **Requeueing jobs with preemption** on page 813
- **Canceling jobs with preemption** on page 809
- **About preemption** on page 807
- **Preemption flags** on page 825

Simple example of preemption

This section illustrates the process of setting up preemption on your system from beginning to end and contains examples of what actions to take and what you should see as you go.

Example scenario

For this basic setup example, we will have a user who can submit to either a "test1" or "test2" QoS. This example will use a *REQUEUE* preemption type.

We will go through three parts to set up this preemption:

- Configuring the `moab.cfg` file
- Submitting a job to the *PREEMPTEE* QoS
- Submitting a job to the *PREEMPTOR* QoS

Okay, let's get started!

Configuring moab.cfg

First, you will need to make some configurations to the `moab.cfg` file.

1. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)
2. Make sure that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*, which is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 831).
3. Set the **PREEMPTPOLICY** type. In this example, **PREEMPTPOLICY** is set to *REQUEUE*. For more information, see **PREEMPTPOLICY types** on page 827.
4. Set up QFLAGS to mark jobs as *PREEMPTEE* (a lower-priority job that can be preempted by a higher-priority job), or as *PREEMPTOR* (a higher-priority job that can preempt a lower-priority job). For more information, see **Preemption flags** on page 825.

   For this example, we also set **JOBFLAGS**=*RESTARTABLE* (because this example uses *REQUEUE*). For more information, see **Requeueing jobs with preemption** on page 813.

5. Make sure that the *PREEMPTEE* job has a lower priority than the *PREEMPTOR* job.

Here is an example of how that would all look in a `moab.cfg` file (text marked red for emphasis).

```
GUARANTEEDPREEMPTION TRUE
#should not be JOBNODEMATCHPOLICY EXACTNODE as it causes problems when starting jobs

PREEMPTPOLICY REQUEUE

QOSCFG[test1] QFLAGS=PREEMPTEE JOBFLAGS=RESTARTABLE MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=1000
```

Now you can submit a job to the preemptee QoS (*test1*).

Submitting a job to the preemptee QoS

Let's submit a job to the preemptee QoS (*test1*), requesting all processor cores in the cluster:

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test1 -l procs=128
```

Take a look at the showq and checkjob output:

```
Moab.1
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.1    john        Running    128       00:09:59      Wed Nov 9 15:56:33

1 active job     128 of 128 processors in use by local jobs (100.00%)
                 2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

```
[john@g06]# checkjob Moab.1
job Moab.1

State: Running
Creds: user:john group:john qos:test1
WallTime: 00:00:00 of 00:10:00
SubmitTime: Wed Nov 9 15:56:33
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Wed Nov 9 15:56:33
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses

Allocated Nodes:
node[01-02]*64

IWD: /opt/native/
SubmitDir: /opt/native/
Executable: /opt/native/spool/moab.job.zOyf1N

StartCount: 1
Flags: RESTARTABLE,PREEMPTEE,GLOBALQUEUE,PROCSPECIFIED
Attr: PREEMPTEE
StartPriority: 100
Reservation 'Moab.1' (-00:00:03 -> 00:09:57 Duration: 00:10:00
```

Submitting a job to the preemptor QoS

Now we will submit a preemptor QoS job (*test2*) to preempt the first job (*test1*):

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test2 -l procs=128
```

Examine the following output for showq and checkjob:

```
Moab.2
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS      REMAINING     STARTTIME
Moab.2    john        Running    128        00:09:59      Wed Nov 9 15:56:47

1 active job 128 of 128 processors in use by local jobs (100.00%)
             2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME
Moab.1    john        Idle       128        00:10:00      Wed Nov 9 15:56:33

1 eligible job

blocked jobs-----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total jobs: 2
```

Note that the preemptor job (Moab.2) moved to Running, while the preemptee job (Moab.1) was requeued.

```
[john@g06]# checkjob Moab.2
job Moab.2

State: Running
Creds: user:john group:john qos:test2
WallTime: 00:02:04 of 00:10:00
SubmitTime: Wed Nov 9 15:56:46
(Time Queued Total: 00:00:01 Eligible: 00:00:00)

StartTime: Wed Nov 9 15:56:47
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses
NodeCount: 2

Allocated Nodes:
node[01-02]*64


IWD: /opt/native/
SubmitDir: /opt/native/
Executable: /opt/native/spool/moab.job.ELoX5Q

StartCount: 1
Flags: HASPREEMPTED,PREEMPTOR,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 10000
Reservation 'Moab.2' (-00:02:21 -> 00:07:39 Duration: 00:10:00)
```

Note the flag, HASPREEMPTED. HASPREEMPTED is set when the PREEMPTOR job has preempted the PREEMPTEE job. Also note that the preemptor job priority plays a very big role in preemption. Generally, you should assign the preemptor a higher priority than any other queued jobs so that it will move to (or near to) the top of the eligible queue.

Related topics

- **About preemption** on page 807
- **Preemption flags** on page 825

- **PREEMPTPOLICY types** on page 827
- **Manual preemption commands** on page 824
- **Testing and troubleshooting preemption** on page 831

Testing and troubleshooting preemption

There are multiple steps associated with setting up a working preemption policy. With preemption, issues arise because it appears that Moab is not allowing preemptor jobs to preempt preemptee jobs in the right way. To diagnose this, use the following checklist:

	Verify that preemptor jobs are marked with the *PREEMPTOR* flag. (Verify with checkjob <JOBID> | grep Flags.)
	Verify that preemptee jobs are marked with the *PREEMPTEE* flag. (Verify with checkjob <JOBID> | grep Flags.)
	Verify that the start priority of the preemptor job is higher than the priority of the preemptee job. (Verify with checkjob <JOBID> | grep Priority.)
	Verify that the resources allocated to the preemptee job match those requested by the preemptor job.
	Verify that the preemptor job is within the 32-preemptee limit.
	Verify that there are no policies preventing preemption from occurring. (Verify with checkjob -v -n <NODEID> <JOBID>.)
	Verify that the **PREEMPTPOLICY** parameter is properly set. (See **PREEMPTPOLICY types** on page 827.)
	Verify that the preemptee job is properly marked as restartable, suspendable, or checkpointable. (Verify with checkjob <JOBID> | grep Flags.)
	Verify that **GUARANTEEDPREEMPTION** is set to *TRUE*.
	Verify that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*. Moab does not currently consider *EXACTNODE* when it handles preemption, resulting in unexpected behavior when *EXACTNODE* is set in an environment with preemption.
	Verify that **NODEACCESSPOLICY** is *not* set to *SINGLEUSER*. (*SHARED* is recommended.)
	Verify that **BACKFILLPOLICY** is set to *FIRSTFIT*.
	Verify that the resource manager is properly responding to preemption requests. (Use **mdiag -R**.)

> If there is a resource manager level race condition, verify that Moab is properly holding target resources. (Verify with **mdiag -S** and set **RESERVATIONRETRYTIME** if needed.)

Related topics

- **About preemption** on page 807
- **Quality of Service (QoS) Facilities** on page 499
- **Managing QoS Access** on page 506
- **JOBMAXPREEMPTPERITERATION** on page 958
- **Trigger components** on page 744
- **Checkpoint/Restart Facilities** on page 528
- **ENABLEFSVIOLATIONPREEMPTION** on page 929
- **PREEMPTPRIOJOBSELECTWEIGHT** on page 997
- **PREEMPTSEARCHDEPTH** on page 998
- **USAGEEXECUTIONTIMEWEIGHT** on page 1042 (control priority of suspended jobs)
- **IGNOREPREEMPTEEPRIORITY** on page 950 (relative job priority is ignored in preemption decisions)
- **DISABLESAMECREDPREEMPTION** on page 924 (jobs cannot preempt other jobs with the same credential)
- **PREEMPTRTIMEWEIGHT** on page 997 (add remaining time of jobs to preemption calculation)

Job templates

About job templates

A Moab job template is a set of pre-configured settings, attributes, and resources that Moab applies to jobs that match certain criteria or to which you manually apply it. They perform three primary functions:

1. They generically match and categorize jobs.
2. They set arbitrary default or forced attributes for certain jobs.
3. They generate workflows that create and maintain user-requested services in a cloud environment.

You can use job templates in many aspects of schedulingPeer-Based Grid usage policies. Job templates are defined using the **JOBCFG** on page 954 configuration parameter.

Two methods exist for applying job templates to jobs. You can use the **JOBMATCHCFG** on page 957 parameter to mark a template that contains the criteria a job must meet for eligibility and another template as the one to be applied to the job if it is eligible. This allows you to automate the use of templates. For example, to force all interactive jobs to run on a certain set of nodes, you can set one template (the criteria template) to have the *interactive* flag, then give the other template the desired host list. You can also apply a template directly to a job at submission if that ability is enabled for that template.

Job template how-to's

Job template references

How-to's

Creating job templates

Context

Job templates are created in the Moab configure file using the **JOBCFG** on page 954 parameter.

To create a job template

1. Open `moab.cfg`. Add the **JOBCFG** parameter and give the new job template a unique name.

```
JOBCFG[newtemplate]
```

2. Configure any desired attributes (see **Job template extension attributes** on page 837.). Some of the important attributes include:

   - **FLAGS** on page 839 - Lets you specify any job flags that should be applied.

   ```
   JOBCFG[newtemplate] FLAGS=SUSPENDABLE
   ```

   When Moab applies `newtemplate` to a job, the job is marked as suspendable.

   - **SELECT** on page 845 - Lets you apply the template directly at job submission.

   ```
   JOBCFG[newtemplate] FLAGS=SUSPENDABLE SELECT=TRUE
   ```

   When you submit a job via msub, you can specify that your job has `newtemplate` applied to it. When Moab applies the template to a job, that job is marked as suspendable.

   - **TEMPLATEDEPEND** on page 847 - Lets you create dependencies when you create a job template workflow (see **Creating workflows with job templates** on page 836.).

```
JOBCFG[newtemplate] FLAGS=SUSPENDABLE SELECT=TRUE TEMPLATEDEPEND=AFTER:job1.pre
```

When Moab applies `newtemplate` to a job, the job cannot run until job `job1.pre` has finished running; the job is also marked as suspendable. You can specify that Moab apply this template to a job as you submit it.

3. If you want to automate job template application, see **Applying templates based on job attributes** on page 834 for instructions. If you want to apply the template manually on job submission, see **Requesting job templates directly** on page 835 for instructions.

Related topics

- **Job template extension attributes** on page 837
- **Job template examples** on page 849

Viewing job templates

Context

You can view a job template by running the mdiag -j command.

To view a job template

- Run the mdiag -j command with the *policy* flag. Moab returns a list of job templates configured in `moab.cfg`.

```
> mdiag -j --flags=policy --blocking
```

Applying templates based on job attributes

Context

The **JOBMATCHCFG** on page 957 parameter allows you to establish relationships between a number of job templates. JMAX and JMIN function as filters to determine whether a job is eligible for a subsequent template to be applied to the job. If a job is eligible, JDEF and JSET templates apply attributes to the job. See **Job template extension attributes** on page 837 for more information about the **JOBMATCHCFG** attributes. The table on that page indicates which job template types are compatible with which job template extension attributes.

> **JSET**s and **JDEF**s have only been tested using msub as the job submission command.

To apply a job template based on job attributes

1. In the Moab configuration file, create a job template with a set of criteria that a job must meet in order for Moab to apply the template. In the following example, Moab will apply a template to all interactive jobs, so the first template sets the *interactive* flag.

```
JOBCFG[inter.min] FLAGS=interactive
```

2. Create the job template that Moab should apply to the job if it meets the requirements set in the first template. In this example, Moab ignores all configured policies, so the second template sets the *ignpolicies* flag.

```
JOBCFG[inter.set] FLAGS=ignpolicies
```

3. Use the **JOBMATCHCFG** parameter and its **JMAX** or **JMIN** (specify the template specifying maximum or minimum requirements) and **JDEF** or **JSET** (specify the template to be applied) attributes to demonstrate the relationship between the two templates (See **Job template matching attributes** on page 849 for more information.). In this case, all interactive jobs ignore policies; in other words, if a submitted job has at least the *inter.min* template settings, Moab applies the *inter.set* template settings to the job.

```
JOBMATCHCFG[interactive] JMIN=inter.min JSET=inter.set
```

> *Moab applies the inter.set template to all jobs with the interactive flag set, causing them to ignore Moab's configured policies.*

4. To control which job template is applied to a job that matches multiple templates, use **FLAGS**=*BREAK*. Job templates are processed in the order they are listed in the configuration file and using the *BREAK* flag causes Moab to stop evaluating **JOBMATCHCFG** entries that occur after the current match.

```
JOBMATCHCFG[small] JMIN=small.min JMAX=small.max JSET.set=small.set FLAGS=BREAK
JOBMATCHCFG[large] JMIN=large.min JMAX=large.max JSET=large.set
```

> *In this case, the large template would not be applied when a job matches both the small and large templates. The small template matches first, and because of **FLAGS**=BREAK, Moab stops evaluating further **JOBMATCHFG** entries for the job.*

Related topics

- **Requesting job templates directly** on page 835
- **Job template examples** on page 849

Requesting job templates directly

Context

When a job template has its **SELECT** on page 845 attribute set to *TRUE*, you can request that template directly on job submission.

To directly request job templates

1. Set the **SELECT** attribute on the template in `moab.cfg`.

```
JOBCFG[medium.set] NODESET=ONEOF:FEATURE:fast,slow SELECT=true
```

2. Submit a job with a resource list (msub -l), requesting the template using the format `template=<templateName>`.

```
> msub -l template=medium.set
```

Moab creates a job with the *medium.set* job template created in step 1.

> Attributes set in the template are evaluated as if they were part of the job submission. They are still subject to all of the same ACLs and policies.

Related topics

- **Applying templates based on job attributes** on page 834

Creating workflows with job templates

Context

Moab can create workflows from individual jobs using job templates.

To build a workflow with job templates

1. Create the jobs in the workflow using the **JOBCFG** on page 954 parameter (See **Creating job templates** on page 833 for more information.). It might be useful to add the **PURGEONSUCCESSONLY** on page 157 flag to your setup or destroy jobs; it will allow you to restart the jobs easily if they fail. Specify the order in which they should run with the **TEMPLATEDEPEND** on page 847 attribute. Please see the Job dependency syntax table for a list of valid dependency options.

```
JOBCFG[setup.pre]   TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/setup.pre.sh
JOBCFG[setup.pre2]  TEMPLATEDEPEND=AFTER:setup.pre SELECT=TRUE
EXEC=/nfs/tools/setup.pre2.sh
JOBCFG[engineering] TEMPLATEDEPEND=AFTER:setup.pre2
```

When Moab applies the engineering template to a qualifying job, the job will not run until template job setup.pre and then setup.pre2 are created from the specified **EXEC** *strings and finish running.*

> The Moab naming convention for jobs created with job templates is `<moabId>.<templateName>`. By default, when Moab submits jobs to only one resource manager, the job IDs are synchronized with the resource manager's job IDs. You can use the parameter **USEMOABJOBID** on page 1045 so that a template-created job is easily associated with its parent job (such as `moab.1`, `moab.1.setup.pre`).

2. Create the job template that will act as the criteria a job must meet for Moab to apply the *engineering* template. In this situation, the job must be submitted with the account name *engineering*.

```
JOBCFG[engineering.match] ACCOUNT=engineering
```

3. Create the **JOBMATCHCFG** on page 957 configuration to tell Moab that when a job matches the *engineering.match* template, it should apply the *engineering* template.

```
JOBMATCHCFG[engineering.job] JMIN=engineering.match JSET=engineering
```

Related topics

- **Job template extension attributes** on page 837
- **Job template workflow examples** on page 850
- **Creating job templates** on page 833

References

Job template extension attributes

When creating a job template, you can use any attribute acceptable within the WIKI workload query data format. In addition, job templates can use any of the extension attributes in the following table. Note that the Template type (**JMIN**, **JMAX**, **JDEF**, **JSET**) row indicates compatibility with the associated attribute (See **Applying templates based on job attributes** on page 834 for more information.).

> Attributes set in a template are evaluated as if they were part of the original job submission. Their jobs are still subject to all the same ACLs and policies.

ACCOUNT	
Format	<ACCOUNT>[,<ACCOUNT>]...
Template type	**JMIN** **JDEF** **JSET**
Description	Account credentials associated with job. This is used for job template matching.
Example	`JOBCFG[public] FLAGS=preemptee` `JOBCFG[public.min] ACCOUNT=public_acct` `JOBMATCHCFG[public] JMIN=public.min JSET=public`

CLASS	
Format	<CLASS>[,<CLASS>]...
Template type	**JMIN** **JDEF** **JSET**
Description	Class credentials associated with job. This is used for job template matching.
Example	`JOBCFG[night] FLAGS=preemptor` `JOBCFG[night.min] CLASS=night_class` `JOBMATCHCFG[night] JMIN=night.min JSET=night`

CPUCLOCK	
Format	<STRING>
Template type	JMIN JMAX JSET
Description	CPU clock frequency for all CPUs of a job. For more information, see **CPUCLOCK** on page 621. The job template extension overrides the job script.
Example	`JOBCFG[slow] SELECT=TRUE cpuclock=1400` `JOBCFG[fast] SELECT=TRUE cpuclock=3200` `JOBCFG[cpu.min] CPUCLOCK=1000` `JOBCFG[cpu.max] CPUCLOCK=2000` `JOBCFG[cpu.set] CPUCLOCK=1500` `JOBMATCHCFG[cpu] JMIN=cpu.min JMAX=cpu.max JSET=cpu.set`

CPULIMIT	
Format	[[[DD:]HH:]MM:]SS
Template type	JMIN JMAX JDEF JSET
Description	Maximum amount of CPU time used by all processes in the job.
Example	`JOBCFG[job.min] CPULIMIT=1:00:00:00` `JOBCFG[job.max] CPULIMIT=2:00:00:00`

DESCRIPTION	
Format	<STRING>
Template type	JMAX JDEF
Description	Description of the job. When you run the checkjob command, the description appears as Reason.
Example	`JOBCFG[webdb] DESCRIPTION="Template job"`

DPROCS

Format	<INTEGER>
Template type	JMIN JMAX JSET
Description	Number of processors dedicated per task. The default is 1.
Example	`JOBCFG[job.min] DPROCS=2` `JOBCFG[job.max] DPROCS=4`

EXEC

Format	<STRING>
Template type	JSET
Description	Specifies what the job runs, regardless of what the user set.
Example	`JOBCFG[setup.pre] EXEC=nfs/tools/setup.pre.sh`

FLAGS

Format	<JOBFLAG>[,<JOBFLAG>]...
Template type	JMIN JDEF JSET
Description	One or more legal job flag values.
Example	`JOBCFG[webdb]` `FLAGS=NORMSTART`

GNAME

Format	<STRING>

GNAME	
Template type	JDEF JSET
Description	Group credential associated with job.
Example	`JOBCFG[webserv] GNAME=service` For matching the group, see the GROUP attribute.

GRES	
Format	<genericResource>[:<COUNT>][,<genericResource>[:<COUNT>]]...
Template type	JMAX JDEF
Description	Consumable generic attributes associated with individual nodes or the special pseudo-node global, which provides shared cluster (floating) consumable resources. Use the NODECFG parameter to configure such resources.
Example	`JOBCFG[gres.set] GRES=abaqus:2` *In this example, the gres.set template applies two Abaqus licenses per task to a matched job.*

GROUP	
Format	<GROUP>[,<GROUP>]...
Template type	JMIN
Description	Group credentials associated with job. This is used for job template matching.
Example	`JOBCFG[webserv] GROUP=service` For information about setting the group, see the GNAME attribute.

MEM	
Format	<INTEGER>
Template type	JMIN JMAX JDEF JSET
Description	Maximum amount of physical memory per task used by the job in megabytes. You can optionally specify other units with your integer (`300kb` or `2gb`, for example). See **Requesting Resources** on page 2237 for more information.
Example	`JOBCFG[smalljobs] MEM=25`

NODEACCESSPOLICY	
Format	One of the following: SHARED, SHAREDONLY, SINGLEJOB, SINGLETASK , SINGLEUSER, or UNIQUEUSER
Template type	JDEF JSET
Description	Specifies how node resources will be shared by a job. See the **Node Access Policies** on page 407 for more information.
Example	`JOBCFG[serverapp] NODEACCESSPOLICY=SINGLEJOB`

NODERANGE	
Format	<MIN>[,<MAX>]
Template type	JMAX JDEF
Description	Minimum and maximum nodes allowed to be allocated to job.
Example	`JOBCFG[vizserver] NODERANGE=1,16`

NODES	
Format	<INTEGER>
Template type	JMIN JMAX JSET
Description	Number of nodes required by the job. The default is 1. See Node Definition for more information.
Example	`JOBCFG[job.min] NODES=2` `JOBCFG[job.max] NODES=4`

NODESET	
Format	<STRING>
Template type	JSET
Description	See **Node Set Overview** on page 515 for more information.
Example	`JOBCFG[medium.set]` `NODESET=ONEOF:FEATURE:fast,slow`

PARTITION	
Format	<PARTITION>[:<PARTITION>]...
Template type	JMIN JDEF JSET
Description	Specifies the partition (or partitions) in which a job must run.
Example	`JOBCFG[meis] PARTITION=math:geology`

PREF	
Format	<FEATURE>[,<FEATURE>]...
Template type	**JDEF** **JSET**
Description	Specifies which node features are preferred by the job and should be allocated if available. See PREF for more information.
Example	`JOBCFG[meis] PREF=bigmem`

PRIORITY	
Format	<INTEGER>
Template type	**JMAX** **JDEF**
Description	Relative job priority. **PRIORITY** works only as a default setting and not as an override (**JSET**) setting.
Example	`JOBCFG[meis] PRIORITY=25000`

PROCRANGE	
Format	<MIN>[,<MAX>]
Template type	JDEF JSET
Description	Minimum and maximum processors allowed to be allocated to job.
Example	`JOBCFG[meis] PROCRANGE=2,64`

QOS	
Format	<QOS>[,<QOS>]...
Template type	JMIN JDEF JSET
Description	QoS credentials associated with job. This is used for job template matching.
Example	`JOBCFG[admin] RFEATURES=bigmem` `JOBCFG[admin.min] QOS=admin_qos` `JOBMATCHCFG[admin] JMIN=admin.min JSET=admin`

RARCH	
Format	<STRING>
Template type	JSET
Description	Architecture required by job.
Example	`JOBCFG[servapp]` `RARCH=i386`

RFEATURES	
Format	<FEATURE>[,<FEATURE>]...
Template type	JMIN JDEF JSET
Description	List of features required by job.
Example	`JOBCFG[servapp]` `RFEATURES=fast,bigmem`

RM	
Format	<STRING>
Template type	JDEF JSET
Description	Destination resource manager to be associated with job.
Example	`JOBCFG[webdb] RM=slurm`

ROPSYS	
Format	<STRING>
Template type	JDEF JSET
Description	Operating system required by job.
Example	`JOBCFG[test.set] ROPSYS=windows`

SELECT	
Format	<BOOLEAN> : TRUE | FALSE
Description	Job template can be directly requested by job at submission.
Example	`JOBCFG[servapp] SELECT=TRUE`

SOFTWARE	
Format	<RESTYPE>[{+|:}<COUNT>][@<TIMEFRAME>]
Template type	JDEF JSET

SOFTWARE	
Description	Indicates generic resources required by the job. See SOFTWARE for more information.
Example	`JOBCFG[servapp] SOFTWARE=matlab:2`

SYSTEMJOBTYPE	
Template type	JMIN
Description	System job type (ex. vmcreate).
Example	`JOBCFG[vmcreate.min] SYSTEMJOBTYPE=vmcreate` `JOBCFG[vmcreate.set]` `TRIGGER=atype=reserve,action="00:05:00",etype=end` `JOBMATCHCFG[vmcreate] JMIN=vmcreate.min JSET=vmcreate.set`

TASKS	
Format	<INTEGER>
Template type	JMIN JMAX JSET
Description	Number of tasks required by job. The default is 1. See Task Definition for more information.
Example	`JOBCFG[job.min] TASKS=4` `JOBCFG[job.max] TASKS=8`

TASKPERNODE	
Format	<INTEGER>
Template type	JMIN JMAX JDEF

TASKPERNODE	
Description	Exact number of tasks required per node. The default is 0. **TASKPERNODE** works only as a default setting and not as an override (**JSET**) setting.
Example	`JOBCFG[job.min] TASKPERNODE=2` `JOBCFG[job.max] TASKPERNODE=4`

TEMPLATEDEPEND	
Format	<TYPE>:<TEMPLATE_NAME>
Description	Create another job from the *<TEMPLATE_NAME>* job template, on which any jobs using this template will depend. This is used for dynamically creating workflows. See Job Dependencies for more information.
Example	`JOBCFG[engineering] TEMPLATEDEPEND=AFTER:setup.pre` `JOBCFG[setup.pre] SELECT=TRUE EXEC=/tools/setup.pre.sh`

UNAME	
Format	<STRING>
Default	**JDEF** **JSET**
Description	User credential associated with job.
Example	`JOBCFG[webserv] UNAME=service` For matching the user, see the USER attribute.

USER	
Format	<USER>[,<USER>]...

USER	
Template type	JMIN JMAX
Description	User credentials associated with job.
Example	`JOBCFG[webserv] USER=service` For setting the user, see the UNAME attribute.

VARIABLE	
Format	<NAME>[:<VAL>]
Template type	JMIN JSET
Description	Variables attached to the job template.
Example	`JOBCFG[this] VARIABLE=var1:1 VARIABLE=var2:1` Variables are set upon successful completion of the job.

WCLIMIT	
Format	[[HH:]MM:]SS
Template type	JMIN JMAX JDEF JSET
Description	Walltime required by job. The default is 8640000 (100 days).
Example	`JOBCFG[job.min] WCLIMIT=2:00:00` `JOBCFG[job.max] WCLIMIT=12:00:00`

Related topics

- **Job template examples** on page 849
- **Creating job templates** on page 833

Job template matching attributes

The **JOBMATCHCFG** on page 957 parameter allows you to establish relationships between a number of job templates. The table in **Job template extension attributes** on page 837 indicates which job template types are compatible with which job template extension attributes. The following types of templates can be specified with the **JOBMATCHCFG** parameter:

Attribute	Description
JMAX	A potential job is rejected if it has matching attributes set or has resource requests that exceed those specified in this template. For **JMAX**, a job template can specify only positive non-zero numbers as maximum limits for generic resources. If a job requests a generic resource that is not limited by the template, then the template can still be used.
JMIN	A potential job is rejected if it does not have matching attributes set or has resource requests that do not meet or exceed those specified in this template.
JDEF	A matching job has the specified attributes set as defaults but all values can be overridden by the user if the matching attribute is explicitly set at job submission time.
JSET	A matching job has the specified attributes forced to these values and these values override any values specified by the submitter at job submission time.
JSTAT	A matching job has its usage statistics reported into this template.

Related topics

- **Job template extension attributes** on page 837
- **Job template examples** on page 849
- **Applying templates based on job attributes** on page 834

Job template examples

Job templates can be used for a wide range of purposes including enabling automated learning, setting up custom application environments, imposing special account constraints, and applying group default settings. The following examples highlight some of these uses:

Example 3-164: Setting up application-specific environments

```
JOBCFG[xxx] EXEC=*app* JOBPROLOG=/usr/local/appprolog.x
```

Example 3-165: Applying job preferences and defaults

```
JOBCFG[xxx] CLASS=appq EXEC=*app* PREF=clearspeed
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT] PRIORITYF=5.0*PREF
```

Example 3-166: Applying resource constraints to fuzzy collections

In the following example, a job template match is set up. Using the **JOBMATCHCFG** on page 957 parameter, Moab is configured to apply all attributes of the *inter.set* job template to all jobs that match the constraints of the *inter.min* job template. In this example, all interactive jobs are assigned the *ignpolicies* flag that allows them to ignore active, idle, system, and partition level policies. Interactive jobs are also locked into the test standing reservation and thus only allowed to run on the associated nodes.

```
# limit all users to a total of two non-interactive jobs
USERCFG[DEFAULT] MAXJOB=2
SRCFG[test] DESCRIPTION="compute pool for interactive and short duration jobs"
SRCFG[test] JOBATTRLIST=INTERACTIVE
SRCFG[test] MAXTIME=1:00:00
SRCFG[test] HOSTLIST=R:atl[16-63]
JOBCFG[inter.min] FLAGS=interactive
JOBCFG[inter.set] FLAGS=ignpolicies
JOBMATCHCFG[interactive] JMIN=inter.min JSET=inter.set
```

Example 3-167: Resource manager templates

In the following example, interactive jobs are not allowed to enter through this resource manager and any job that does route in from this resource manager interface has the *preemptee* flag set.

```
JOBCFG[no_inter] FLAGS=interactive
JOBCFG[preempt_job] FLAGS=preemptee
RMCFG[gridA.in] MAX.JOB=no_inter SET.JOB=preempt_job
```

Related topics

- **Job template extension attributes** on page 837
- **Job template workflow examples** on page 850
- **Creating job templates** on page 833

Job template workflow examples

Example 3-168: A workflow with multiple dependencies

In this example the job will depend on the completion of two other jobs Moab creates. Both jobs execute at the same time.

```
# Engineering2
JOBCFG[engineering2] TEMPLATEDEPEND=AFTER:engineering2.pre2
TEMPLATEDEPEND=AFTER:engineering2.pre
JOBCFG[engineering2.pre2] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/engineering2.pre2.sh
JOBCFG[engineering2.pre] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/engineering2.pre.sh
JOBCFG[engineering2.match] ACCOUNT=engineering2
JOBMATCHCFG[engineering2.job] JMIN=engineering2.match JSET=engineering2
```

Example 3-169: Jobs that run after the submission job

Three additional jobs are created that depend on the submitted job.

```
# Workflow 2
JOBCFG[workflow2] TEMPLATEDEPEND=BEFORE:workflow2.post1
TEMPLATEDEPEND=BEFORE:workflow2.post2 TEMPLATEDEPEND=BEFORE:workflow2.post3
JOBCFG[workflow2.post1] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow2.post1.sh
JOBCFG[workflow2.post2] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow2.post2.sh
JOBCFG[workflow2.post3] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow2.post3.sh
JOBCFG[workflow2.match] ACCOUNT=workflow2
JOBMATCHCFG[workflow2.job] JMIN=workflow2.match JSET=workflow2
```

Example 3-170: A complex workflow

A complex workflow that handles failures.

```
# Workflow 4
JOBCFG[workflow4.step1] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step1.sh TEMPLATEDEPEND=BEFOREFAIL:workflow4.fail1
JOBCFG[workflow4.fail1] TASKS=1 WCLIMIT=00:00:30 SELECT=TRUE
EXEC=/usr/tools/workflow.fail.1.sh TEMPLATEDEPEND=BEFOREANY:workflow4.fail2
JOBCFG[workflow4.fail2] TASKS=1 WCLIMIT=00:00:30 SELECT=TRUE
EXEC=/usr/tools/workflow.fail.2.sh
# Submission job
JOBCFG[workflow4.step2] TEMPLATEDEPEND=AFTEROK:workflow4.step1
TEMPLATEDEPEND=BEFOREOK:workflow4.step3.1 TEMPLATEDEPEND=BEFOREOK:workflow4.step3.2
JOBCFG[workflow4.step3.1] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step3.1.sh
JOBCFG[workflow4.step3.2] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step3.2.sh TEMPLATEDEPEND=BEFOREOK:workflow4.step4
JOBCFG[workflow4.step4] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step4.sh
JOBCFG[workflow4.step4] TEMPLATEDEPEND=BEFOREOK:workflow4.step5.1
TEMPLATEDEPEND=BEFOREOK:workflow4.step5.2 TEMPLATEDEPEND=BEFORENOTOK:workflow4.step5.3
JOBCFG[workflow4.step5.1] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step5.1.sh
JOBCFG[workflow4.step5.2] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step5.2.sh
JOBCFG[workflow4.step5.3] TASKS=1 WCLIMIT=00:00:30 SELECT=TRUE
EXEC=/usr/tools/workflow.step5.3.sh
JOBCFG[workflow4.match] ACCOUNT=workflow4
```

Related topics

- **Creating workflows with job templates** on page 836
- **Applying templates based on job attributes** on page 834
- **Job template examples** on page 849
- **Job template extension attributes** on page 837

Moab Workload Manager for Grids

Cluster Consolidation and Sovereign Grids is a video tutorial of a session offered at Moab Con that offers further details for understanding cluster consolidation and sovereign grids.

Moab Grid Scheduler allows sites to establish relationships among multiple clusters. There are three types of relationships you can implement within the grid: (1) centralized management, (2) hierarchal management, and (3) localized management. These relationships provide access to additional resources, improve load-balancing, provide single system images, and offer other benefits. The grid interface is flexible allowing sites to establish the needed relationship.

Grid Basics

- Submitting Jobs to the Grid
- Viewing Jobs and Resources

Grid Overview

A grid enables you to exchange workload and resource status information and to distribute jobs and data among clusters in an established relationship. In addition, you can use resource reservations to mask reported resources, coordinate requests for consumable resources, and quality of service guarantees.

In a grid, some servers running Moab are a source for jobs (that is, where users, portals, and other systems submit jobs), while other servers running Moab are a destination for these jobs (that is, where the jobs execute). Thus, jobs originate from a source server and move to a destination server. For a source server to make an intelligent decision, though, resource availability information must flow from a destination server to that source server.

Because you can manage workload on both the source and destination side of a grid relationship, you have a high degree of control over exactly when, how, and where to execute workload.

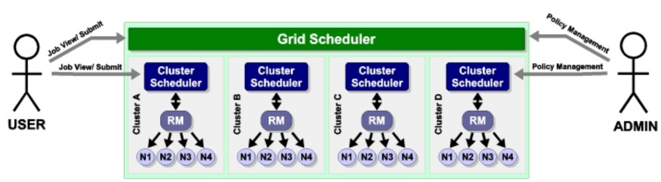
Grid Benefits

Moab's peer-to-peer capabilities can be used for multiple purposes, including any of the following:

- manage access to external shared resources
- enable cluster monitoring information services
- enable massive-scalability clusters
- enable distributed grid computing

Of these, the most common use is the creation of grids to join multiple centrally managed, partially autonomous, or fully autonomous clusters. The purpose of this section is to highlight the most common uses of grid technology and provide references to sections which further detail their configuration and management. Other sections cover the standard aspects of grid creation including configuring peer relationships, enabling data staging, credential management, usage policies, and other factors.

Image 3-11: Jobs submitted to grid scheduler then cluster schedulers



Management-Scalability

Much like a massive-scalability cluster, a massive-scalability grid allows organizations to overcome scalability limitations in resource managers, networks, message passing libraries, security middleware, file systems, and other forms of software and hardware infrastructure. Moab does this by allowing a single large set of resources to be broken into multiple smaller, more manageable clusters, and then virtually re-assembling them using Moab. Moab becomes responsible for integrating the seams between the clusters and presenting a single-system image back to the end-users, administrators, and managers.

Jobs cannot span clusters.

Resource Access

In some cases, the primary motivation for creating a grid is to aggregate resources of different types into a single system. This aggregation allows for multi-step jobs to run a portion of the job on one architecture, and a portion on another.

A common example of a multi-architecture parameter-sweep job would be a batch regression test suite which requires a portion of the tests running on Redhat 7.2, a portion on SUSE 9.1, a portion on Myrinet nodes, and a portion on Infiniband nodes. While it would be very difficult to create and manage a single cluster which simultaneously provided all of these configurations, Moab can be used to create and manage a single grid which spans multiple clusters as needed.

Load-Balancing

While grids often have additional motivations, it is rare to have a grid created where increased total system utilization is not an objective. By aggregating the total pool of jobs requesting resources and increasing the pool of resources available to each job, Moab is able to improve overall system utilization, sometimes significantly. The biggest difficulty in managing multiple clusters is preventing inter-cluster policies and the cost of migration from overwhelming the benefits of decreased fragmentation losses. Even though remote resources may be available for immediate usage, migration costs can occur in the form of credential, job, or data staging and impose a noticeable loss in responsiveness on grid workload.

Moab provides tools to allow these costs to be monitored and managed and both cluster and grid level performance to be reported.

Single System Image (SSI)

Another common benefit of grids is the simplicity associated with a single system image-based resource pool. This simplicity generally increases productivity for end-users, administrators, and managers.

An SSI environment tends to increase the efficiency of end-users by minimizing human errors associated with porting a request from a known system to a less known system. Additionally, the single point of access grid reduces human overhead associated with monitoring and managing workload within multiple independent systems.

For system administrators, a single system image can reduce overhead, training time, and diagnostic time associated with managing a cluster. Furthermore, with Moab's peer-to-peer technology, no additional software layer is required to enable the grid and no new tools must be learned. No additional layers means no additional failure points, and that is good for everyone involved.

Managers benefit from SSI by being able to pursue organization mission objectives globally in a more coordinated and unified manner. They are also able to monitor progress toward those objectives and effectiveness of resources in general.

High Availability

A final benefit of grids is their ability to decrease the impact of failures. Grids add another layer of high availability to the cluster-level high availability. For some organizations, this benefit is a primary motivation, pulling together additional resources to allow workload to continue to be processed even in the event that some nodes, or even an entire cluster, become unavailable. Whether the resource unavailability is based on node failures, network failures, systems middleware, systems maintenance, or other factors, a properly configured grid can reroute priority workload throughout the grid to execute on other compatible resources.

With grids, there are a number of important factors in high availability that should be considered:

- enabling highly available job submission/job management interfaces
- avoiding network failures with redundant routes to compute resources
- handling partial failures
- dynamically restarting failed jobs

Grid Relationships

There are three types of relationships you can implement within the grid:

- Centralized Management (Master/Slave)
- Centralized/Localized Management (Hierarchal)
- Localized Management (Peer-to-Peer)

Centralized Management (Master/Slave)

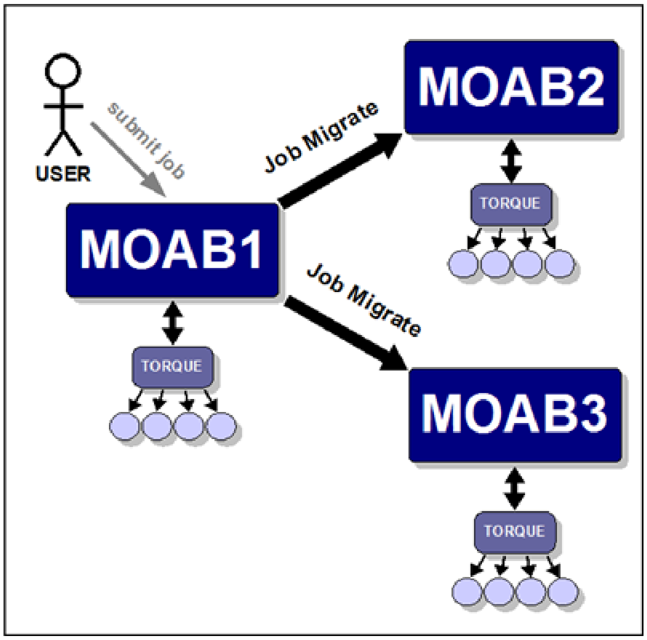
The centralized management model (master/slave) allows users to submit jobs to a centralized source server running Moab. The source Moab server obtains full resource information from all clusters and makes intelligent scheduling decisions across all clusters. Jobs (and data when configured to do so) are distributed to the remote clusters as needed. The centralized management model is recommended for intra-organization grid environments when cluster autonomy is not as necessary.

In the centralized management (master-slave) configuration, roles are clear. In other configurations, individual Moab servers may simultaneously act as sources to some clusters and destinations to others or as both a source and a destination to another cluster.

Example of the Centralized Management (Master/Slave) Model

XYZ Research has three clusters - MOAB1, MOAB2, and MOAB3--running Moab and the TORQUE resource manager. They would like to submit jobs at a single location (cluster MOAB1) and have the jobs run on whichever cluster can provide the best responsiveness.

The desired behavior is essentially a *master-slave* relationship. MOAB1 is the central, or master, cluster. On MOAB1, resource managers point to the local TORQUE resource manager and to the Moab servers on cluster MOAB2 and cluster MOAB3. The Moab servers on MOAB2 and MOAB3 are configured to trust cluster MOAB1 and to execute in slave mode.



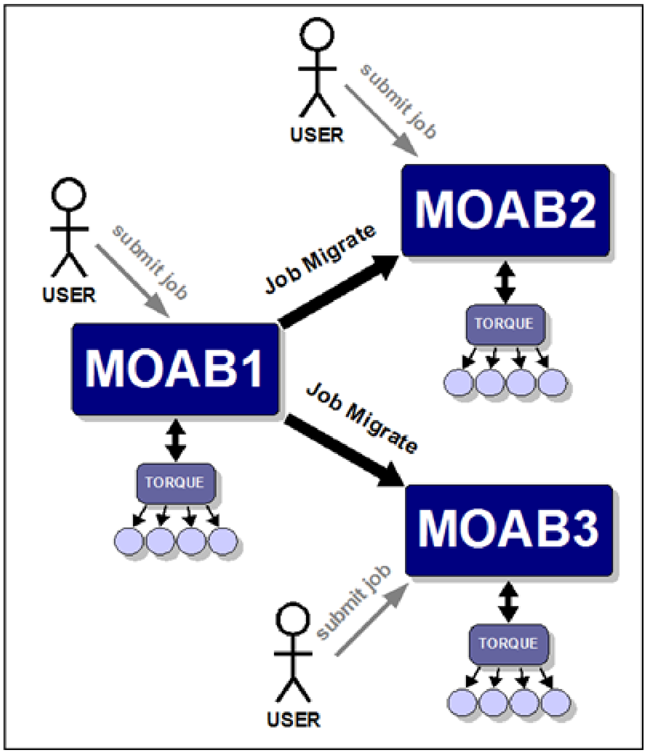
With this configuration, XYZ Research may submit jobs to the master Moab server running on cluster MOAB1 and may, as stated earlier, submit jobs from the slave nodes as well. However, only the master Moab server may schedule jobs. For example, cluster MOAB2 and cluster MOAB3 cannot schedule a job, but they can accept a job and retain it in an idle state until the master directs it to run.

> You can turn off job submission on slave nodes by setting the **DISABLESLAVEJOBSUBMIT** on page 924 parameter to *TRUE*.

The master Moab server obtains full resource information from all three clusters and makes intelligent scheduling decisions and distributes jobs (and data when configured to do so) to the remote clusters. The Moab servers running on clusters MOAB2 and MOAB3 are destinations behaving like a local resource manager. The Moab server running on MOAB1 is a source, loading and using this resource information.

Centralized/Localized Management (Hierarchal)

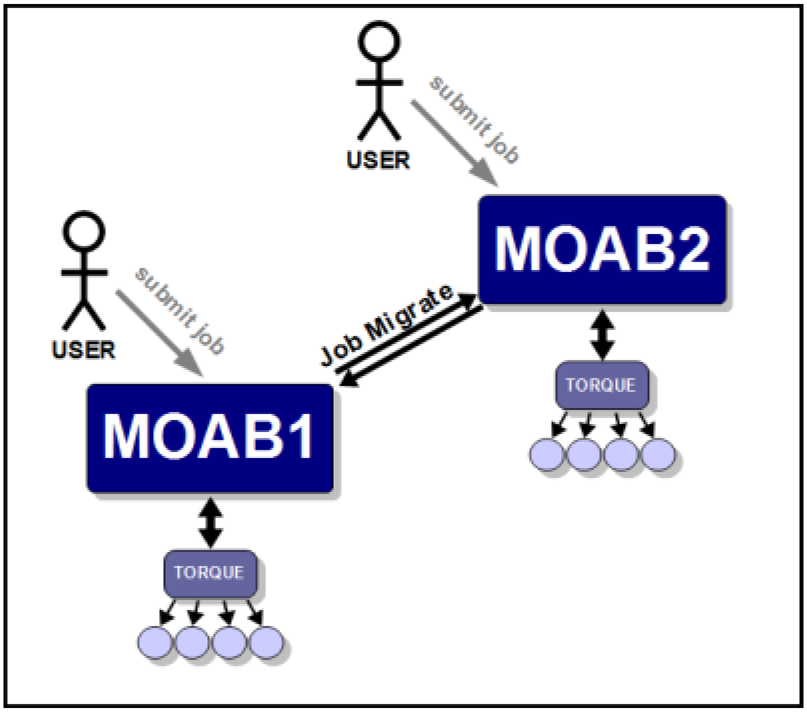
As with the centralized management model (master/slave), the hierarchal model allows users to submit jobs to a centralized source server running Moab. However, in the hierarchal model, clusters retain sovereignty, allowing local job scheduling. Thus, if communication between the source and destination clusters is interrupted, the destination cluster(s) can still run jobs locally.



In the hierarchal model, the source Moab server obtains full resource information from all clusters and makes intelligent scheduling decisions across all clusters. As needed, jobs and data are distributed to the remote clusters. Or, if preferred, a destination cluster may also serve as its own source; however, a destination cluster may not serve as a source to another destination cluster. The centralized management model is recommended for intra-organization grid environments when cluster autonomy and/or local management is necessary.

Localized Management (Peer-to-Peer)

The localized management (peer-to-peer) model allows you to submit jobs on one cluster and schedule the jobs on the other cluster (it currently works with two clusters). For example, a job may be submitted on MOAB1 and run on MOAB2. Jobs can also migrate in the opposite direction (that is, from MOAB2 to MOAB1). The source servers running Moab obtain full resource information from both clusters and make intelligent scheduling decisions across both clusters. Jobs (and data when configured to do so) are migrated to other clusters as needed.



Jobs will not migrate indefinitely. The localized management model limits them to one migration.

This model allows clusters to retain their autonomy while still allowing jobs to run on either cluster. No central location for job submission is needed, and you do not need to submit jobs from different nodes based on resource needs. You can submit a job from any location and it is either migrated to nodes on the least utilized cluster or the cluster requested in the job submission. This model is recommended for grids in an inter-organization grid environment.

Submitting Jobs to the Grid

In any peer-to-peer or grid environment where jobs must be migrated between clusters, use the Moab msub command. Once a job has been submitted to Moab using msub, Moab identifies potential destinations and migrates the job to the destination cluster.

Using Moab's msub job submission command, jobs may be submitted using PBS or LSF command file syntax and be run on any cluster using any of the resource managers. For example, a PBS job script may be submitted using msub and depending on availability, Moab may translate a subset of the job's directives and execute it on an LSF cluster.

Moab can only stage/migrate jobs between resource managers (in between clusters) that have been submitted using the msub command. If jobs are submitted directly to a low-level resource manager, such as PBS, Moab will still be able to schedule them but only on resources directly managed by the resource manager to which they were submitted.

Example 1

A small pharmaceutical company, BioGen, runs two clusters in a centralized relationship. The slave is an older IBM cluster running Loadleveler, while the master manages the slave and also directly manages a large Linux cluster running TORQUE. A new user familiar with LSF has multiple LSF job scripts he would like to continue using. To enable this, the administrators make a symbolic link between the Moab msub client and the file bsub. The user begins submitting his jobs via bsub and, according to availability, the jobs run on either the Loadleveler or TORQUE clusters.

Example 2

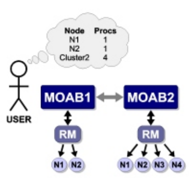
A research lab wants to use spare cycles on its four clusters, each of which is running a local resource manager. In addition to providing better site-wide load balancing, the goal is to also provide some of its users with single point access to all compute resources. Various researchers have made it clear that this new multi-cluster load balancing must not impose any changes on users who are currently using these clusters by submitting jobs locally to each cluster.

In this example, the scheduler mode of the destination clusters should be set to *NORMAL* rather than *SLAVE*. In *SLAVE* mode, Moab makes no local decisions - it simply follows the directions of remote trusted peers. In *NORMAL* mode, each Moab is fully autonomous, scheduling all local workload and coordinating with remote peers when and how to schedule migrated jobs.

From the perspective of a local cluster user, no new behaviors are seen. Remote jobs are migrated in from time to time, but to the user each job looks as if it were locally submitted. The user continues to submit, view, and manage jobs as before, using existing local jobs scripts.

Viewing Jobs and Resources

By default, each destination Moab server will report all compute nodes it finds back to the source Moab server. These reported nodes appear within the source Moab as local nodes each within a partition associated with the resource manager reporting them. If a source resource manager was named `slave1`, all nodes reported by it would be associated with the `slave1`partition. Users and administrators communicating with the source Moab via Moab Cluster Manager, Moab Access Portal, or standard Moab command line tools would be able to view and analyze all reported nodes.



> The grid view will be displayed if either the source or the destination server is configured with grid view.

For job information, the default behavior is to only report to the source Moab information regarding jobs that originated at the source. If information about other jobs is desired, this can be configured as shown in the Workload Submission and Control section.

Related topics

- Resource Control and Access

Grid Configuration Basics

Peer Configuration Overview

In the simplest case, establishing a peer relationship can be accomplished with as few as two configuration lines: one line to indicate how to contact the peer and one line to indicate how to authenticate the server. However, data migration issues, credential mapping, and usage policies must often be addressed in order to make a peer-based grid effective.

To address these issues Moab provides facilities to control how peers inter-operate, enabling full autonomy over both client and server ends of the peer relationship.

Initial Configuration

At a minimum, only two parameters must be specified to establish a peer relationship: **RMCFG** on page 1014 and **CLIENTCFG[<X>]** on page 916. **RMCFG** allows a site to specify interface information directing Moab on how to contact and inter-operate with the peer. For peer interfaces, a few guidelines must be followed with the **RMCFG** parameter:

- the **TYPE** attribute of the peer must be set to *moab*
- the **SERVER** attribute must point to the host and user interface port of the remote Moab server
- the *name* of the resource manager should match the name of the remote peer cluster as specified with the **SCHEDCFG** on page 1018 parameter in the peer `moab.cfg`.

```
# moab.cfg on MoabServer01

SCHEDCFG[MoabServer01] MODE=NORMAL SERVER=hpc-01:41111
RMCFG[MoabServer02]    TYPE=moab   SERVER=hpc-02:40559
...
```

Configuring the **CLIENTCFG** parameter is mandatory. When specifying the **CLIENTCFG** parameter for peers, the following guidelines must be followed:

- the **CLIENTCFG** parameter must be specified in the `moab-private.cfg` file on both peers
- an `RM:` prefix is required before the peer's name
- if using default secret key based security, the value of the **KEY** attribute must match the **KEY** value set on the corresponding remote peer
- the **AUTH** attribute must be set to *admin1* in the `moab-private.cfg` on the destination Moab

```
# moab-private.cfg on MoabServer01

CLIENTCFG[RM:MoabServer02] KEY=3esfv0=32re2-tdbne
....
```

```
# moab-private.cfg on MoabServer02

CLIENTCFG[RM:MoabServer01] KEY=3esfv0=32re2-tdbne AUTH=admin1
...
```

Centralized Grid Management (Master/Slave)

Master Configuration

The process of setting up the master configuration is the same as setting up a source Moab configuration. The master/slave relationship is configured in each `moab.cfg` on the slave.

```
# moab.cfg on Master

SCHEDCFG[master] SERVER=master:42559 MODE=NORMAL
...
```

```
# moab-private.cfg on Master

CLIENTCFG[RM:slave1]  KEY=3esfv0=32re2-tdbne
...
```

Slave Configuration

The slave's relationship with the master is determined by the **MODE**. Setting **MODE** to *SLAVE* notifies the master to take control of starting jobs on the slave. The master starts the jobs on the slave.. In *SLAVE* mode, jobs can be submitted locally to the slave, but are not seen or started by the master. When a job is submitted locally to the slave the job is locked into the cluster and cannot migrate to other clusters.

```
# moab.cfg on Slave

SCHEDCFG[slave1] SERVER=slave1:42559 MODE=SLAVE
...
```

```
# moab-private.cfg on Slave

CLIENTCFG[RM:master] KEY=3esfv0=32re2-tdbne  AUTH=admin1
...
```

Hierarchal Grid Management

- Configuring a Peer Server (Source)
    - Simple Hierarchal Grid

Configuring a Peer Server (Source)

Peer relationships are enabled by creating and configuring a resource manager interface using the RMCFG parameter. This interface defines how a given Moab will load resource and workload information and enforce its scheduling decisions. In non-peer cases, the **RMCFG** parameter points to a resource manager such as TORQUE, LSF, or SGE. However, if the **TYPE** attribute is set to *moab*, the **RMCFG** parameter can be used to configure and manage a peer relationship.

Simple Hierarchal Grid

The first step to create a new peer relationship is to configure an interface to a destination Moab server. In the following example, cluster C1 is configured to be able to *see* and *use* resources from two other clusters.

```
SCHEDCFG[C1] MODE=NORMAL SERVER=head.C1.xyz.com:41111
RMCFG[C2]    TYPE=moab   SERVER=head.C2.xyz.com:40559
RMCFG[C3]    TYPE=moab   SERVER=head.C3.xyz.com:40559
...
```

C1 allows a global view of the underlying clusters. From C1, jobs can be viewed and modified. C2 and C3 act as separate scheduling entities that can receive jobs from C1. C1 migrates jobs to C2 and C3 based on available resources and policies of C1. Jobs migrated to C2 and C3 are scheduled according to the polices on C2 and C3.

In this case, one **RMCFG** parameter is all that is required to configure each peer relationship if standard secret key based authentication is being used and a shared default secret key exists between the source and destination Moabs. However, if peer relationships with multiple clusters are to be established and a per-peer secret key is to be used (highly recommended), then a CLIENTCFG parameter must be specified for the authentication mechanism. Because the secret key must be kept secure, it must be specified in the moab-private.cfg file. For the current example, a per-peer secret key could be set up by creating the following moab-private.cfg file on the C1 cluster.

```
CLIENTCFG[RM:C2] KEY=fastclu3t3r
CLIENTCFG[RM:C3] KEY=14436aaa
```

The key specified can be any alphanumeric value and can be locally generated or made up. The only critical aspect is that the keys specified on each end of the peer relationship match.

Additional information can be found in the Grid Security section which provides detailed information on designing, configuring, and troubleshooting peer security.

Continuing with the example, the initial source side configuration is now complete. On the destination clusters, C2 and C3, the first step is to configure authentication. If a shared default secret key exists between all three clusters, then configuration is complete and the clusters are ready to communicate. If per-peer secret keys are used (recommended), then it will be necessary to create matching moab-private.cfg files on each of the destination clusters. With this example, the following files would be required on C2 and C3 respectively:

```
CLIENTCFG[RM:C1] KEY=fastclu3t3r AUTH=admin1
```

```
CLIENTCFG[RM:C1] KEY=14436aaa AUTH=admin1
```

Once peer security is established, a final optional step would be to configure scheduling behavior on the destination clusters. By default, each destination cluster accepts jobs from each trusted peer. However, it will also be fully autonomous, accepting and scheduling locally submitted jobs and enforcing its own local policies and optimizations. If this is the desired behavior, then configuration is complete.

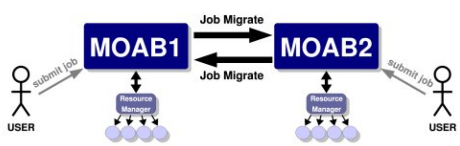
In the current example, with no destination side scheduling configuration, jobs submitted to cluster `C1` can run locally, on cluster `C2` or on cluster `C3`. However, the established configuration does not necessarily enforce a strict master-slave relationship because each destination cluster (`C2` and `C3`) has complete autonomy over how, when, and where it schedules both local and remote jobs. Each cluster can potentially receive jobs that are locally submitted and can also receive jobs from other source Moab servers. See Slave Mode for more information on setting up a master-slave grid.

Further, each destination cluster will accept any and all jobs migrated to it from a trusted peer without limitations on who can run, when and where they can run, or how many resources they can use. If this behavior is either too restrictive or not restrictive enough, then destination side configuration will be required.

Localized Grid Management

Enabling Bi-Directional Job Flow

Image 3-12: Bi-directional peer-to-peer setup



For each peer interface, an **RMCFG** on page 1014 parameter is only required for the server (or source side of the interface). If two peers are to share jobs in both directions, the relationship is considered to be bi-directional.

True Peer-to-Peer Grid

Previous examples involved grid masters that coordinated the activities of the grid and made it so direct contact between peers was not required. However, if preferred, the master is not required and individual clusters can interface directly with each other in a true peer manner. This configuration is highlighted in the following example:

```
# Cluster A

SCHEDCFG[clusterA] MODE=NORMAL SERVER=clusterA
RMCFG[clusterA]    TYPE=pbs
RMCFG[clusterB]    TYPE=moab   SERVER=clusterB:40559
CLIENTCFG[RM:clusterB] AUTH=admin1 KEY=banana16
```

```
# Cluster B

SCHEDCFG[clusterB] MODE=NORMAL SERVER=clusterB
RMCFG[clusterB]    TYPE=pbs
RMCFG[clusterA]    TYPE=moab   SERVER=clusterA:40559
CLIENTCFG[RM:clusterA] AUTH=admin1 KEY=banana16
```

If you are using Moab Accounting Manager, the *Start* action is not supported as a non-blocking accounting action in Peer-to-Peer grids. You will need to include *Start* as a blocking action. For example:

```
AMCFG[mam] BLOCKINGACTIONS=Start
```

Resource Control and Access

Controlling Resource Information

In a Moab peer-to-peer grid, resources can be viewed in one of two models:

- Direct - nodes are reported to remote clusters exactly as they appear in the local cluster
- Mapped - nodes are reported as individual nodes, but node names are mapped to a unique name when imported into the remote cluster

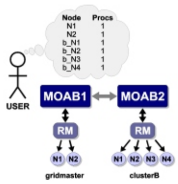
Direct Node View

Direct node import is the default resource information mode. No additional configuration is required to enable this mode.

Mapped Node View

In this mode, nodes are reported just as they appear locally by the exporting cluster. However, on the importing cluster side, Moab maps the specified node names using the resource manager object map. In an object map, node mapping is specified using the `node` keyword as in the following example:

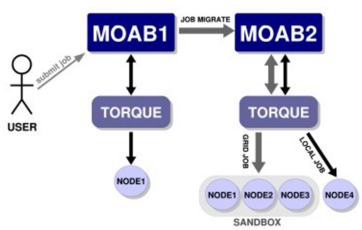
```
SCHEDCFG[gridmaster] MODE=NORMAL
RMCFG[clusterB]      TYPE=moab OMAP=file://$HOME/clusterb.omap.dat
...
node:b_*,*
```



In this example, all nodes reported by *clusterB* have the string `b_` prepended to prevent node name space conflicts with nodes from other clusters. For example, if cluster *clusterB* reported the nodes `node01`, `node02`, and `node03`, cluster *gridmaster* would report them as `b_node01`, `b_node02`, and `b_node03`.

See [object mapping] for more information on creating an object map file.

Managing Resources with Grid Sandboxes



A cluster may wish to participate in a grid but may desire to dedicate only a set amount of resources to external grid workload or may only want certain peers to have access to particular sets of resources. With Moab, this can be achieved by way of a grid sandbox which must be configured at the destination cluster. Grid sandboxes can both constrain external resource access and limit which resources are reported to other peers. This allows a cluster to only report a defined subset of its total resources to source peers and restricts peer workload to the sandbox. The sandbox can be set aside for peer use exclusively, or can allow local workload to also run inside of it. Through the use of multiple, possibly overlapping grid sandboxes, a site may fully control resource availability on a per peer basis.

A grid sandbox is created by configuring a [standing reservation] on a destination peer and then specifying the *ALLOWGRID* flag on that reservation. This flag tells the Moab destination peer to treat the standing reservation as a grid sandbox, and, by default, only the resources in the sandbox are visible to grid peers. Also, the sandbox only allows workload from other peers to run on the contained resources.

Example 3-171: Dedicated Grid Sandbox

```
SRCFG[sandbox1] PERIOD=INFINITY HOSTLIST=node01,node02,node03
SRCFG[sandbox1] CLUSTERLIST=ALL FLAGS=ALLOWGRID
...
```

The standing reservation sandbox1 creates a grid sandbox which always exists and contains the nodes node01, node02, and node03. This sandbox will only allow grid workload to run within it by default. This means that the scheduler will not consider the boxed resources for local workload.

Grid sandboxes inherit all of the same power and flexibility that standing reservations have. See Managing Reservations for additional information.

The flag *ALLOWGRID* marks the reservation as a grid sandbox and as such, it precludes grid jobs from running anywhere else. However, it does *not* enable access to the reserved resources. The **CLUSTERLIST** attribute in the above example enables access to all remote jobs.

Controlling Access on a Per Cluster Basis

Often clusters may wish to control which peers are allowed to use certain sandboxes. For example, Cluster A may have a special contract with Cluster B and will let overflow workload from Cluster B run on 60% of its resources. A third peer in the grid, Cluster C, doesn't have the same contractual agreement, and is only allowed 10% of Cluster A at any given time. Thus two separate sandboxes must be made to accommodate the different policies.

```
SRCFG[sandbox1] PERIOD=INFINITY HOSTLIST=node01,node02,node03,node04,node05
 SRCFG[sandbox1] FLAGS=ALLOWGRID CLUSTERLIST=ClusterB
 SRCFG[sandbox2] PERIOD=INFINITY HOSTLIST=node06 FLAGS=ALLOWGRID
 SRCFG[sandbox2] CLUSTERLIST=ClusterB,ClusterC,ClusterD USERLIST=ALL
 ...
```

This example configuration illustrates how cluster A could set up their sandboxes to follow a more complicated policy. In this policy, sandbox1provides exclusive access to nodes 1 through 5 to jobs coming from peer ClusterB by including ***CLUSTERLIST****=ClusterB in the definition. Reservation sandbox2provides shared access to node6 to local jobs and to jobs from clusters B, C, and D through use of the* ***CLUSTERLIST*** *and* ***USERLIST*** *attributes.*

With this setup, the following policies are enforced:

- local jobs may see all nodes and run anywhere except nodes 1 through 5
- jobs from cluster B may see and run only on nodes 1 through 6
- jobs from clusters C and D may see and run only on node 6

As shown in the example above, sandboxes can be shared across multiple peers by listing all sharing peers in the **CLUSTERLIST** attribute (comma delimited).

Access Control Lists/Granting Access to Local Jobs

It is not always desirable to have the grid sandbox reserve resources for grid consumption, exclusively. Many clusters may want to use the grid sandbox when local workload is high and demand from the grid is relatively low. Clusters may also wish to further restrict what kind of grid workload can run in a sandbox. This fine-grained control can be achieved by attaching access control lists (ACLs) to grid sandboxes.

Since sandboxes are basically special standing reservations, the syntax and rules for specifying an ACL is identical to those found in Managing Reservations.

Example

```
SRCFG[sandbox2] PERIOD=INFINITY HOSTLIST=node04,node05,node06
SRCFG[sandbox2] FLAGS=ALLOWGRID QOSLIST=high GROUPLIST=engineer
...
```

A cluster decides to dedicate resources to a sandbox, but wishes local workload to also run within it. An additional ACL is then associated with the definition. The reservation sandbox2 takes advantage of this feature by allowing local jobs running with a QOS of high, or under the group engineer, to also run on the sandboxed nodes node04, node05, and node06.

Workload Submission and Control

- Controlling Peer Workload Information
- Determining Resource Availability

Controlling Peer Workload Information

By default, a peer is only responsible for workload that is submitted via that particular peer. This means that when a source peer communicates with destination peers it only receives information about workload it sent to those destination peers. If desired, the destination peers can send information about *all* of its workload: both jobs originating locally and remotely. This is called *local workload exporting*. This may help simplify administration of different clusters by centralizing monitoring and management of jobs at one peer.

To implement local workload exporting, use the *LOCALWORKLOADEXPORT* resource manager flag. For example:

```
RMCFG[ClusterA.INBOUND] FLAGS=LOCALWORKLOADEXPORT   # source peer
...
```

This example shows the configuration on a destination peer (ClusterB) that exports its local and remote workload to the source peer (ClusterA).

LOCALWORDKLOADEXPORT does not need to be configured in master/slave grids.

Related topics

- Job Start Time Estimates

Reservations in the Grid

In some environments, globally-shared resources may need to be managed to guarantee the full environment required by a particular job. Resources such as networks, storage systems, and license managers may be used only by batch workload but this workload may be distributed among multiple independent clusters. Consequently, the jobs from one cluster may utilize resources required by jobs from another. Without a method of coordinating the needs of the various cluster schedulers, resource reservations will not be respected by other clusters and will be of only limited value.

Using the centralized model, Moab allows the importing and exporting of reservations from one peer server to another. With this capability, a source peer can be set up for the shared resource to act as a clearinghouse for other Moab cluster schedulers. This source peer Moab server reports configured and available resource state and in essence possesses a global view of resource reservations for all clusters for the associated resource.

To allow the destination peer to export reservation information to the source Moab, the **RMCFG** on page 1014 lines for all client resource managers must include the flag *RSVEXPORT*. The source Moab should be configured with a resource manager interface to the destination peer and include both the *RSVEXPORT* and *RSVIMPORT* flags. For the destination peer, *RSVEXPORT* indicates that it should *push* information about newly created reservations to the source Moab, while the *RSVIMPORT* flag indicates that the source Moab server should import and locally enforce reservations detected on the destination peer server.

Grid Usage Policies

Grid Usage Policy Overview

Moab allows extensive control over how peers interact. These controls allow the following:

- Limiting which remote users, group, and accounts can utilize local compute resources
- Limiting the total quantity of local resources made available to remote jobs at any given time
- Limiting remote resource access to a specific subset of resources
- Limiting timeframes during which local resources will be made available to remote jobs
- Limiting the types of remote jobs which will be allowed to execute

Peer Job Resource Limits

Both source and destination peers can limit the types of jobs they will allow in terms of resources requested, services provided, job duration, applications used, etc using Moab's job template feature. Using this method, one or more job profiles can be created on either the source or destination side, and Moab can be configured to allow or reject jobs based on whether or not the jobs meet the specified job profiles.

When using the **ALLOWJOBLIST** and **REJECTJOBLIST** attributes, the following rules apply:

- All jobs that meet the job templates listed by **ALLOWJOBLIST** are allowed.
- All jobs that do not meet **ALLOWJOBLIST** job templates and which do meet **REJECTJOBLIST** job templates are rejected.
- All jobs that meet no job templates in either list are allowed.

Usage Limits via Peer Credentials

With peer interfaces, destination clusters willing to accept remote jobs can map these jobs onto a select subset of users, accounts, QoSs, and queues. With the ability to lock these jobs into certain credentials comes the ability to apply any arbitrary credential constraints, priority adjustments, and resource limitations normally available within cluster management. Specifically, the following can be accomplished:

- limit number of active jobs simultaneously allowed
- limit quantity of allocated compute resources simultaneously allowed
- adjust job priority
- control access to specific scheduling features (deadlines, reservations, preemption, etc)
- adjust fairshare targets
- limit resource access

Using General Policies in a Grid Environment

While Moab does provide a number of unique grid-based policies for use in a grid environment, the vast majority of available management tools come from the transparent application of cluster policies. Cluster-level policies such as job prioritization, node allocation, fairshare, usage limits, reservations, preemption, and allocation management all just work and can be applied in a grid in exactly the same manner.

The one key concept to understand is that in a centralized based grid, these policies apply across the entire grid; in a peer-based grid, these policies apply only to local workload and resources.

Source Cluster Policies

In many cases, organizations are interested in treating jobs differently based on their point of origin. This can be accomplished by assigning and/or keying off of a unique credential associated with the remote workload. For example, a site may wish to constrain jobs from a remote cluster to only a portion of the total available cluster cycles. This could be accomplished using usage limits, fairshare targets, fairshare caps, reservations, or allocation management based policies.

The examples below show three different approaches for constraining remote resource access.

Example 3-172: Constraining Remote Resource Access via Fairshare Caps

```
# define peer relationship and map all incoming jobs to orion account
RMCFG[orion.INBOUND] SET.JOB=orion.set
JOBCFG[orion.set] ACCOUNT=orion
# configure basic fairshare for 7 one day intervals
FSPOLICY DEDICATEDPS
FSINTERVAL 24:00:00
FSDEPTH 7
FSUSERWEIGHT 100
# use fairshare cap to limit jobs from orion to 10% of cycles
ACCOUNTCFG[orion] FSCAP=10%
```

Example 3-173: Constraining Remote Resource Access via Fairshare Targets and Preemption

```
# define peer relationship and map all incoming jobs to orion account RMCFG
[orion.INBOUND] SET.JOB=orion.set
JOBCFG[orion.set] ACCOUNT=orion
# local cluster can preempt jobs from orion
USERCFG[DEFAULT] JOBFLAGS=PREEMPTOR
PREEMPTPOLICY CANCEL
# configure basic fairshare for 7 one day intervals
FSPOLICY DEDICATEDPS
FSINTERVAL 24:00:00
FSDEPTH 7
FSUSERWEIGHT 100
# decrease priority of remote jobs and force jobs exceeding 10% usage to be
preemptible
ACCOUNTCFG[orion] FSTARGET=10-
ENABLEFSVIOLATIONPREEMPTION TRUE
```

Example 3-174: Constraining Remote Resource Access via Priority and Usage Limits

```
# define peer relationship and map all incoming jobs to orion account RMCFG
[orion.INBOUND] SET.JOB=orion.set
JOBCFG[orion.set] QOS=orion
USERCFG[DEFAULT] QDEF=orion
# local cluster can preempt jobs from orion
USERCFG[DEFAULT] JOBFLAGS=PREEMPTOR
PREEMPTPOLICY CANCEL
# adjust remote jobs to have reduced priority
QOSCFG[orion] PRIORITY=-1000
# allow remote jobs to use up to 64 procs without being preemptible and up to 96 as
preemptees
QOSCFG[orion] MAXPROC=64,96
ENABLESPVIOLATIONPREEMPTION TRUE
```

Related topics

- Grid Sandbox - control grid resource access

Grid Scheduling Policies

- Peer-to-Peer Resource Affinity Overview
- Peer Allocation Policies
- Per-partition Scheduling

Peer-to-Peer Resource Affinity Overview

The concept of resource affinity stems from a number of facts:

- Certain compute architectures are able to execute certain compute jobs more effectively than others.
- From a given location, staging jobs to various clusters may require more expensive allocations, more data and network resources, and more use of system services.
- Certain compute resources are owned by external organizations and should be used sparingly.

Regardless of the reason, Moab servers allow the use of peer resource affinity to guide jobs to the clusters that make the best fit according to a number of criteria.

At a high level, this is accomplished by creating a number of job templates and associating the profiles with different peers with varying impacts on estimated execution time and peer affinity.

Peer Allocation Policies

A direct way to assign a peer allocation algorithm is with the PARALLOCATIONPOLICY parameter. Legal values are listed in the following table:

Value	Description
FirstStart	Allocates resources from the eligible peer that can start the job the soonest.
LoadBalance	Allocates resources from the eligible peer with the most available resources; measured in tasks (balances workload distribution across potential peers).
LoadBalanceP	Allocates resources from the eligible peer with the most available resources; measured in percent of configured resources (balances workload distribution across potential peers).
Random	Allocates partitions in a random order each iteration. In general, all the jobs scheduled within the same iteration receive the same randomized list of partitions. This means the randomization happens between iterations and not within the same iteration. One iteration Moab might start with partition X and the next it might start with partition Y.
RoundRobin	Allocates resources from the eligible peer that has been least recently allocated.

The mdiag -t -v command can be used to view current calculated partition priority values.

Per-partition Scheduling

Per-partition scheduling can be enabled by adding the following lines to `moab.cfg`:

```
PERPARTITIONSCHEDULING TRUE
JOBMIGRATEPOLICY JUSTINTIME
```

To use per-partition scheduling, you must configure fairshare trees where particular users have higher priorities on one partition, and other users have higher priorities on a different partition.

> Do not set the USEANYPARTITIONPRIO parameter if you use per-partition scheduling. Doing so causes Moab to schedule jobs to the first partition listed, even if nodes from another partition will be available sooner.

Grid Credential Management

- Peer User Credential Management Overview
- Credential Mapping Files

Peer Credential Management Overview

Moab provides a number of credential management features that allow sites to control which local users can utilize remote resources and which remote users can utilize local resources and under what conditions this access is granted.

Peer Credential Mapping

If two peers share a common user space (a given user has the same login on both clusters), then there is often no need to enable credential mapping. When users, groups, classes, QoS's, and accounts are not the same from one peer to another, Moab allows a site to specify an Object Map URL. This URL contains simple one to one or expression based mapping for credentials and other objects. Using the RMCFG parameter's **OMAP** attribute, a site can tell Moab where to find these mappings. The object map uses the following format:

```
<OBJECTTYPE>:<SOURCE_OBJECTID>,<DESTINATION_OBJECTID>
```

where *<SOURCE_OBJECT>* can be a particular username or an asterisk (*) which is a wildcard matching all credentials of the specified type which have not already been matched.

The object map file can be used to translate the following:

Keyword	Objects
account	accounts/projects
class	classes/queues
file	files/directories
group	groups

Keyword	Objects
node	nodes
qos	QoS
user	users

The following `moab.cfg` and `omap.dat` files demonstrate a sample credential mapping.

```
SCHEDCFG[master1] MODE=normal
RMCFG[slave1]     OMAP=file:///opt/moab/omap.dat
...
```

```
user:joe,jsmith
user:steve,sjohnson
group:test,staff
class:batch,serial
user:*,grid
```

In this example, a job that is being migrated from cluster *master1* to the peer *slave1* will have its credentials mapped according to the contents of the `omap.dat` file. In this case, a job submitted by user *joe* on *master1* will be executed under the user account *jsmith* on peer *slave1*. Any credential that is not found in the mapping file will be passed to the peer as submitted. In the case of the user credential, all users other than *joe* and *steve* will be remapped to the user *grid* due to the wildcard matching.

Because the **OMAP** attribute is specified as a URL, multiple methods can be used to obtain the mapping information. In addition to the file protocol shown in the example above, exec may be used.

Note that there is no need to use the credential mapping facility to map all credentials. In some cases, a common user space exists but it is used to map all classes/queues on the source side to a single queue on the destination side. Likewise, for utilization tracking purposes, it may be desirable to map all source account credentials to a single cluster-wide account.

Source and Destination Side Credential Mapping

Credential mapping can be implemented on the source cluster, destination cluster, or both. A source cluster may want to map all user names for all outgoing jobs to the name *generaluser* for security purposes, and a destination cluster may want to remap all incoming jobs from this particular user to the username *cluster2* and the QoS *grid*.

Preventing User Space Collisions

In some cases, a cluster may receive jobs from two independent clusters where grid wide username distinctiveness is not guaranteed. In this case, credential mapping can be used to ensure the uniqueness of each name. With credential mapping files, this can be accomplished using the *<DESTINATION_CREDENTIAL>* wildcard asterisk (*) character. If specified, this character will be replaced with the exact *<SOURCE_CREDENTIAL>* when generating the destination credential string. For example, consider the following configuration:

```
SCHEDCFG[master1] MODE=normal
RMCFG[slave1]     OMAP=file:///opt/moab/omap.dat  FLAGS=client
...
```

```
user:*,c1_*
group:*,*_grid
account:*,temp_*
```

This configuration will remap the usernames of all jobs coming in from the peer *slave1*. The username *john* will be remapped to *c1_john*, the group *staff* will be remapped to staff_*grid* and the account *demo* will be remapped to *temp_*demo.

Grid Data Management

This method of data staging has been deprecated in Moab Workload Manager 8.0.1 and will be removed from the product in a future release. See **About data staging** on page 880 for information about the new method of staging data.

- Grid Data Management Overview
- Configuring Peer Data Staging
- Peer-to-Peer SCP Key Authentication
- Diagnostics

Grid Data Management Overview

Moab provides a highly generalized data manager interface that can allow both simple and advanced data management services to be used to migrate data amongst peer clusters. Using a flexible script interface, services such as *scp*, *NFS*, and *gridftp* can be used to address data staging needs. This feature enables a Moab peer to push job data to a destination Moab peer.

Configuring Peer Data Staging

Moab offers a simple, automatic configuration, as well as advanced configuration options. At a high level, configuring data staging across a peer-to-peer relationship consists of configuring one or more storage managers, associating them with the appropriate peer resource managers, and then specifying data requirements at the local level—when the job is submitted.

To use the data staging features, you must specify the `--with-grid` option at `./configure` time. After properly configuring data staging, you can submit a job to the peer with any user who has SSH keys set up and Moab will automatically or implicitly stage back the standard out and standard error files created by the job. Files can be implicitly staged in or out before a job runs by using the mstagein or mstageout options of msub.

Simple Configuration

Moab automatically does most of the data staging configuration based on a simplified set of parameters (most common defaults) in the configuration file (`moab.cfg`).

Do the following to configure peer data staging:

1. Configure at least two Moab clusters to work in a grid. Please refer to information throughout Moab Workload Manager for Grids for help on configuring Moab clusters to work together as peers in a grid.
2. Set up SSH keys so that users on the source grid peer can SSH to destination peers without the need for a password.
3. Make necessary changes to the `moab.cfg` file of the source grid peer to activate data staging, which involves creating a new data resource manager definition within Moab. The resource manager provides data staging services to existing peers in the grid. By defining the data resource manager within the moab.cfg, Moab automatically sets up all of the necessary data staging auxiliary scripts.

   Use the following syntax for defining a data resource manager:

   ```
   RMCFG[<RMName>] TYPE=NATIVE RESOURCETYPE=STORAGE
   VARIABLES=DATASPACEUSER=<DataSpaceUser>,DATASPACEDIR=<DataSpaceDir>
   SERVER=<DataServer>
   ```

   - *<RMName>*: Name of the RM (defined as a storage RM type by RESOURCETYPE=STORAGE).
   - *<DataSpaceUser>*: User used to SSH into *<DataServer>* to determine available space in *<DataSpaceDir>*. Moab runs a command similar to the following:

     ```
     ssh <DataServer> -l <DataSpaceUser> df <DataSpaceDir>
     ```
   - *<DataSpaceDir>*: Directory where staged data is stored.
   - *<DataServer>*: Name of the server where *<DataSpaceDir>* is located.

   Define the following URLs:

   ```
   RMCFG[data] CLUSTERQUERYURL=exec://$TOOLSDIR/cluster.query.dstage.pl
   RMCFG[data] SYSTEMMODIFYURL=exec://$TOOLSDIR/system.modify.dstage.pl
   RMCFG[data] SYSTEMQUERYURL=exec://$TOOLSDIR/system.query.dstage.pl
   RMCFG[data] RMINITIALIZEURL=exec://$TOOLSDIR/setup.config.pl
   ```
4. Associate the data resource manager with a peer resource manager.

   ```
   RMCFG[remote_data] TYPE=NATIVE RESOURCETYPE=STORAGE
   VARIABLES=DATASPACEUSER=datauser,DATASPACEDIR=/tmp SERVER=clusterhead
   RMCFG[remote_cluster] TYPE=MOAB SERVER=clusterhead:42559 DATARM=remote_data
   ```
5. Restart Moab to finalize changes. You can use the mschedctl -R command to cause Moab to automatically restart and load the changes.

   When restarting, Moab recognizes the added configuration and runs a Perl script in the Moab tool directory that configures the external scripts (also found in the tools directory) that Moab uses to perform data staging. You can view the data staging configuration by looking at the `config.dstage.pl` file in *`$MOABHOMEDIR`*`/etc`.

Advanced Configuration

If you need a more customized data staging setup, contact your account representative.

Peer-to-Peer SCP Key Authentication

In order to use scp as the data staging protocol, we will need to create SSH keys which allow users to copy files between the two peers, without the need for passwords. For example, if *UserA* is present on the source peer, and his counterpart is *UserB* on the destination peer, then *UserA*will need to create an SSH key and configure *UserB* to allow password-less copying. This will enable *UserA* to copy files to and from the destination peer using Moab's data staging capabilities.

Another common scenario is that several users present on the source peer are mapped to a single user on the destination peer. In this case, each user on the source peer will need to create keys and set them up with the user at the destination peer. Below are steps that can be used to setup SSH keys among two (or more) peers:

> These instructions were written for OpenSSH version 3.6 and might not work correctly for older versions.

Generate SSH Key on Source Peer

As the user who will be submitting jobs on the source peer, run the following command:

```
ssh-keygen -t rsa
```

You will be prompted to give an optional key. Just hit return and ignore this or other settings. When finished, this command will create two files `id_rsa` and `id_rsa.pub` located inside the user's `~/.ssh/` directory.

Copy the Public SSH Key to the Destination Peer

Transfer the newly created public key (`id_rsa.pub`) to the destination peer:

```
scp ~/.ssh/id_rsa.pub ${DESTPEERHOST}:~
```

Disable Strict SSH Checking on Source Peer (Optional)

By appending the following to your `~/.ssh/config` file you can disable SSH prompts which ask to add new hosts to the "known hosts file." (These prompts can often cause problems with data staging functionality.) Note that the ${DESTPEERHOST} should be the name of the host machine running the destination peer:

```
Host ${DESTPEERHOST}
CheckHostIP no
StrictHostKeyChecking no
BatchMode yes
```

Configure Destination Peer User

Now, log in to the destination peer as the destination user and set up the newly created public key to be trusted:

```
ssh ${DESTPEERUSER}@${DESTPEERHOST}
mkdir -p .ssh; chmod 700 .ssh
cat id_rsa.pub >> .ssh/authorized_keys
chmod 600 .ssh/authorized_keys
rm id_rsa.pub
```

If multiple source users map to a single destination user, then repeat the above commands for each source user's SSH public key.

Configure SSH Daemon on Destination Peer

Some configuration of the SSH daemon may be required on the destination peer. Typically, this is done by editing the `/etc/ssh/sshd_config` file. To verify correct configuration, see that the following attributes are set (not commented):

```
---
RSAAuthentication    yes
PubkeyAuthentication yes
---
```

If configuration changes were required, the SSH daemon will need to be restarted:

```
/etc/init.d/sshd restart
```

Validate Correct SSH Configuration

If all is properly configured, if you issue the following command source peer it should succeed without requiring a password:

```
scp ${DESTPEERHOST}:/etc/motd /tmp/
```

Diagnostics

Verify data staging is properly configured by using the following diagnostic commands:

- mdiag -R -v: Displays the status of the storage manager. Verify that you set up the necessary URLs.

```
> mdiag -R -v data
diagnosing resource managers
RM[data]      State: Active  Type: NATIVE  ResourceType: STORAGE
  Server:              keche
  Timeout:             30000.00 ms
  Cluster Query URL:   exec://$TOOLSDIR/grid/cluster.query.dstage.pl
  RM Initialize URL:   exec://$TOOLSDIR/grid/setup.config.pl
  System Modify URL:   exec://$TOOLSDIR/grid/system.modify.dstage.pl
  System Query URL:    exec://$TOOLSDIR/grid/system.query.dstage.pl
  Nodes Reported:      1 (scp://keche//tmp/)
  Partition:           SHARED
  Event Management:    (event interface disabled)
  Variables:           DATASPACEUSER=root,DATASPACEDIR=/tmp
  RM Languages:        NATIVE
  RM Sub-Languages:    -
```

- checknode -v: Executing this on the storage node displays the data staging operations associated with the node and its disk usage.

The number of bytes transferred for each file is currently not used.

```
> checknode -v scp://keche//tmp/
node scp://keche//tmp/
State:      Idle  (in current state for 00:00:13)
Configured Resources: DISK: 578G
Utilized   Resources: DISK: 316G
Dedicated  Resources: ---
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Active Data Staging Operations:
  job         native.2  complete (1 bytes transferred)
(/home/brian/stage.txt)
  job         native.3  pending (1 bytes)  (/home/brian/stage.txt)
Dedicated Storage Manager Disk Usage:  0 of 592235 MB
Cluster Query URL:  exec://$TOOLSDIR/grid/cluster.query.dstage.pl
Partition:  SHARED  Rack/Slot:  ---
Flags:      rmdetected
RM[data]:   TYPE=NATIVE
EffNodeAccessPolicy: SHARED
Total Time: 00:12:15  Up: 00:12:15 (100.00%)  Active: 00:00:00 (0.00%)
Reservations:  ---
```

- mdiag -n: Displays the state of the storage node.

```
> mdiag -n
compute node summary
Name                    State   Procs      Memory         Opsys
compute1                 Idle    4:4       3006:3006      linux
compute2                 Down    0:4       3006:3006      linux
scp://keche//tmp/        Idle    0:0          0:0             -
-----                     ---    4:8       6012:6012      -----
Total Nodes: 3  (Active: 0  Idle: 2  Down: 1)
```

- checkjob -v: Displays the status of the staging request.

The remaining time and size of the file information is currently not used. The information should only be used to see file locations and whether the file has been staged or not.

```
> checkjob -v jobid
...
Stage-In Requirements:
  localhost:/home/brian/stage.txt => keche:/tmp/staged.txt  size:0B
status:[NONE]  remaining:00:00:01
    Transfer URL: file:///home/brian/stage.txt,ssh://keche/tmp/staged.txt
...
```

To ensure that SCP key authentication is properly configured, the following conditions must be met:

- Moab is running as root.
- You are able to issue the following command as the root user without being prompted for a password:

```
su - <DATASPACEUSER> -c "/usr/bin/ssh <destination host> -l <DATASPACEUSER> 'df
-k //tmp/ 2>&1 || echo FAILED'"
```

- You can SSH <*destination host*> without a password.

- The *dataSpaceLocalUser* and *dataSpaceMappedUser* variables in your `/opt/moab/etc/config.dstage.pl` script are set to the same username you assigned through *<DATASPACEUSER>*.

Grid Security

Secret Key Based Server Authentication

Secret key based security is required in order for the grid to work. It is enabled in the `moab-private.cfg` file. Configuration of `moab-private.cfg` is covered throughout the grid configuration documentation, as well as in Appendix E: Security.

Grid Diagnostics and Validation

- Peer Management Overview
- Peer Diagnostic Overview

Peer Management Overview

- Use mdiag -R to view interface health and performance/usage statistics.
- Use mrmctl to enable/disable peer interfaces.
- Use mrmctl -m to dynamically modify/configure peer interfaces.

Peer Diagnostic Overview

- Use mdiag -R to diagnose general RM interfaces.
- Use mdiag -S to diagnose general scheduler health.
- Use mdiag -R -V job <RMID> to diagnose peer-to-peer job migration.

```
> mdiag -R -V job peer1
```

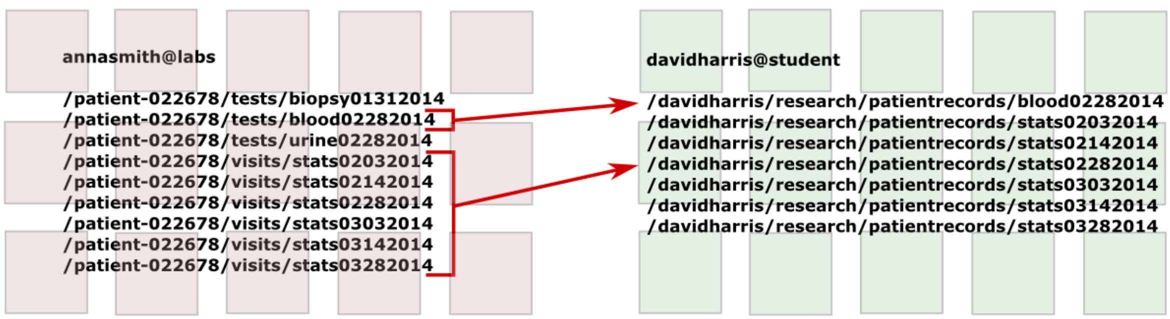
- Use mdiag -R -V data <RMID> to diagnose peer-to-peer data staging.
- Use mdiag -R -V cred <RMID> to diagnose peer-to-peer credential mapping.

Data staging

About data staging

Sometimes you might need a job to process data that resides at another site. With the proper configuration, you can submit your job with the requirement that it copies data from the external site to yours and, if needed, copy the job's resulting data out to the external site for its owner to use. Data staging is an out-of-band method of moving data without reserving compute nodes or other resources for it.

In the example below, which will appear throughout the chapter, a university researcher needs the results of tests done at a hospital to run his job. User `davidharris` on the `student` server of the university submits a job called `Moab.1` that requires several files stored by user `annasmith` on the `research` server of the hospital. `davidharris` submitted `Moab.1` with certain options in place that instruct Moab to copy the files to the `/student/davidharris/research/patientrecords/` directory on the `student` server prior to starting the job.



Moab currently supports the following data staging use cases: 1) Staging data to or from a shared file system, 2) Staging data to or from local node storage on a single compute node, and 3) Staging data to or from a shared file system on an unspecified cluster – resolved at job migration – in a grid configuration.

Before you can submit data staging jobs, you must configure certain generic metrics in your partitions, job templates, and the data staging submit filter for data staging scheduling, throttling, and policies.

Moab uses Linux file transfer utilities to stage the data and includes data staging reference scripts that support the scp and rsync Linux file transfer utilities. The scripts will work for standard installations, but you can customize the script to support data staging to and from an external staging server, the Moab server itself, or a local compute node, depending on your implementation. You can also customize your own script for other file transfer utilities, such as Aspera.

Once you configure your system to support data staging, you can begin creating data staging jobs by attaching the `--stagein`, `--stageinfile`, `--stageinsize`, `--stageout`, `--stageoutfile`, and `--stageoutsize` options to your msub commands. See **Staging data** on page 303 for more information.

How-to's

The following topics describe how to stage data in different Moab environments.

- **Configuring the SSH keys for the data staging transfer script** on page 882
- **Configuring data staging** on page 884
- **Configuring the $CLUSTERHOST variable** on page 894
- **Staging data to or from a shared file system** on page 886
- **Staging data to or from a shared file system in a grid** on page 890
- **Staging data to or from a compute node** on page 895
- **Configuring data staging with advanced options** on page 899

References

The following topics contain detailed information that you can use as reference material for data staging

- **Sample user job script** on page 901
- **Applying the msub submit filter** on page 307

Related topics

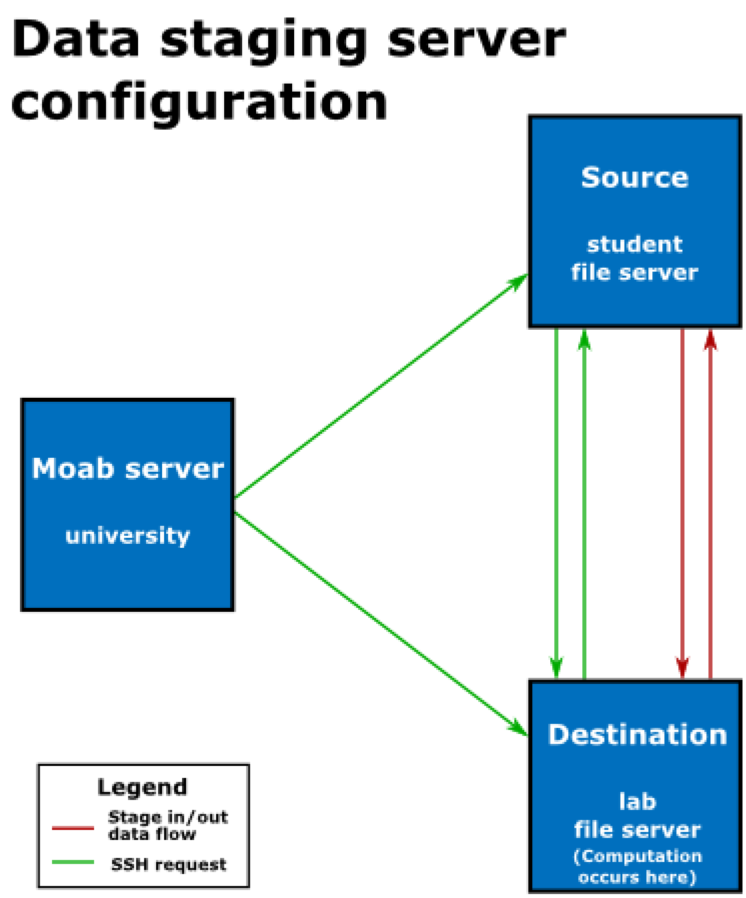
- **msub** on page 290

How-to's

Configuring the SSH keys for the data staging transfer script

Context

For data staging to work correctly, you must configure SSH keys to allow the data staging scripts to run without passphrases. In the sample data staging server configuration shown in the image below, `davidharris` on the `student` server stages data from the source server `student` to the destination server `labs`. The computation occurs on the `labs` server before Moab stages the output data from `labs` back to `student`. The image below demonstrates the SSH connections necessary and how you should configure your SSH keys.



For more details on generating keys, see the [ssh-keygen man page] and "[SSH login without password]".

To configure the SSH keys for the data staging transfer script

1. Generate a new SSH key on the Moab server (`university`) if one does not already exist. To do so, run each of the following steps.
   a. Run `ssh-keygen` to generate a public and private rsa key pair.
   ```
   davidharris@university]$ ssh-keygen
   ```
   b. Enter the name of the file where you want to store the key, or you can accept the default location.
   ```
   /home/davidharris/.ssh/id_rsa
   ```
   c. When prompted for a passphrase, leave it blank and press **Enter**. Repeat when prompted to retype passphrase.
2. Install the public key on the source and destination hosts. Note that in this example the source host is `student` and the destination host is `labs`.
   a. Copy the `university` public key to `student`. Answer **yes** to continue connecting.
   ```
   [davidharris@university]$ ssh-copy-id -i ~/.ssh/id_rsa.pub student
   ```
   b. Copy the `university` public key to `labs`. Answer **yes** to continue connecting.
   ```
   [davidharris@university]$ ssh-copy-id -i ~/.ssh/id_rsa.pub labs
   ```

> The next two steps generate a key-pair for each node. It is acceptable to generate a single key-pair and install it on each node. It does not matter where the key-pair is generated, so long as it is compatible with the SSH client/server.

3. Generate a key pair on the source host (`student`) and install the public key generated to the destination host (`labs`). When prompted for a passphrase, leave it blank and press **Enter**. Repeat when prompted to retype passphrase.
   ```
   [davidharris@student]$ ssh-keygen
   [davidharris@student]$ ssh-copy-id -i ~/.ssh/id_rsa.pub labs
   ```
4. Generate a key pair on the destination host (`labs`) and install the public key generated to the source host (`student`). When prompted for a passphrase, leave it blank and press **Enter**. Repeat when prompted to retype passphrase.
   ```
   [davidharris@labs]$ ssh-keygen
   [davidharris@labs]$ ssh-copy-id -i ~/.ssh/id_rsa.pub student
   ```
5. Ensure that each user who will run data staging jobs has read and write permissions on each source and destination server.
6. Test the configuration. To do so:
   a. Install the modules required to run the data staging scripts. `python-paramiko` is required for data staging, but `python-mock` is only required if you intend to run the unit test.

```
> yum install python-paramiko python-mock
```

b. Transfer a file from the source host to the destination host to verify that the keys work for the users configured. To do so, run `/opt/moab/tools/data-staging/ds_move_scp --test=<source>%<destination>` if you use scp or `/opt/moab/tools/data-staging/ds_move_rsync --test=<source>%<destination> script` if you use rsync. *`<source>%<destination>`* is configured the same way as the `--stagein` and `--stageout` options for msub; for help configuring your source and destination, see **Staging a file or directory** on page 303.

```
[davidharris@university]$ /opt/moab/tools/data-staging/ds_move_rsync --
test=davidharris@student:/tmp/test%davidharris@labs:/tmp
```

c. In the same way, transfer a file from the destination host to the source host to verify that the keys work for the users configured.

```
[davidharris@university]$ /opt/moab/tools/data-staging/ds_move_rsync --
test=davidharris@labs:/tmp/test%davidharris@student:/test_processed
```

Related topics

- **About data staging** on page 880
- **Configuring data staging** on page 884

Configuring data staging

Context

You must modify your Moab configuration to enable data staging. In addition to the configuration steps described below, you might also consider customizing the configuration (including the associated scripts) to meet your site's specific needs.

For advanced configuration steps and options, see **Configuring data staging with advanced options** on page 899.

To configure data staging

1. Verify that your firewall and network are correctly configured to allow the scripts to operate as designed.
2. If you have not already done so, install the modules required to run the data staging scripts. `python-paramiko` is required for data staging, but `python-mock` is only required if you intend to run the unit test.

```
> yum install python-paramiko python-mock
```

3. If you have not already, follow the instructions found in **Configuring the SSH keys for the data staging transfer script** on page 882.
4. Ensure that the data staging scripts are installed on your system. To do so, list the contents of the `/opt/moab/tools/data-staging` directory. You should see the data staging README file, reference scripts, and other related files.

```
> ls -l /opt/moab/tools/data-staging
```

You can copy and modify the reference scripts and configuration files to meet your specific needs. See the README file packaged in the `data-staging` directory for information about modifying these files.

5. Open your *moab.cfg* file for editing and do each of the following tasks:

   a. Configure the data staging msub filter, located in `/opt/moab/tools/data-staging` by default, as a client-side filter. See **Applying the msub submit filter** on page 307 for more information.

   ```
   SUBMITFILTER /opt/moab/tools/data-staging/ds_filter
   ```

   The data staging filter checks the msub argument syntax to verify that the arguments make sense and are consistent; attempts a dry run connection via SSH and the file transfer utility to ensure that keys exist for the user on the necessary systems; and attempts to determine the size of the data that will be transferred.

   You can customize the script to meet your specific needs; the file contains detailed comments illustrating its default behavior to facilitate its modification. If you replace or modify the submit filter, it is your responsibility to ensure that the same functionality described in the paragraph above is present in your filter.

   Note that this filter has the `DEFAULT_TEMPLATE` name which should match the name of the master data staging template in `moab.cfg`. For more information, see **Configuring data staging with advanced options** on page 899.

   b. Set the data staging bandwidth gmetric (`DATASTAGINGBANDWIDTH_MBITS_PER_SEC`) on each partition associated with an RM to the rate at which its network to be used for data staging transfers data in megabits per second (see **Per-Partition Settings** on page 498 for more information). Moab will use the specified rate and the data staging size specified at job submission (see **Stage in or out file size** on page 305 for more information) to determine how long staging the data will take and to schedule the job as soon after data staging completes as possible.

   Example 3-175: Non-grid

   ```
   RMCFG[torque]  Type=pbs
   PARCFG[torque] GMETRIC[DATASTAGINGBANDWIDTH_MBITS_PER_SEC]=58
   ```

   Partition `torque` has a transfer rate of 58 megabits per second. Moab uses the rate when it estimates the time it will take to stage data in and determine when to schedule the job that will use the data.

   Example 3-176: Grid

   ```
   RMCFG[m1] type=Moab
   PARCFG[m1] GMETRIC[DATASTAGINGBANDWIDTH_MBITS_PER_SEC]=100
   ```

   Partition `m1` has a transfer rate of 100 megabits per second. Moab uses the rate when it estimates the time it will take to stage data in and determine when to schedule the job that will use the data.

c. Set the bandwidth generic resource on all nodes to limit the total number of concurrent data staging jobs in your system.

```
NODECFG[GLOBAL] GRES=bandwidth:10
```

Data staging jobs can use up to 10 units of bandwidth on the system. You can specify the number of units consumed by each data staging job when you configure the data staging job templates.

6. Install the msub client filter on all client submission hosts.

Related topics

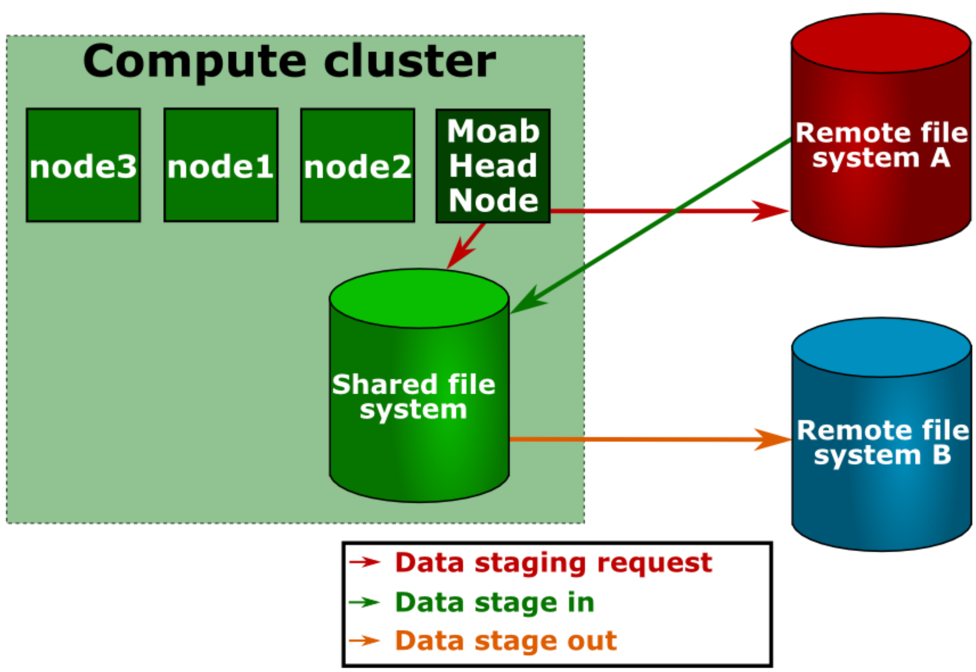
- **About data staging** on page 880

Staging data to or from a shared file system

Context

In the most common data staging use case, the cluster utilizes a shared file system between all compute nodes. This type of data staging makes data stored outside of the cluster available to a job that will run on any set of nodes in the cluster. At the time of submission, you must specify where Moab will obtain the data with a username, host name, and path to a file or directory and where on the shared file system Moab will store the data. After the job runs, you can also copy data from the shared file system back to a remote file system.

Image 3-13: Data staging to or from a shared file system



To stage data to or from a shared file system

1. If you have not already done so, configure your SSH keys and `moab.cfg` to support data staging. See **Configuring the SSH keys for the data staging transfer script** on page 882 and **Configuring data staging** on page 884 for more information.
2. Create your job templates for data staging jobs in `moab.cfg`. The templates in the example below create a compute job that stages data in before it starts and stages data out when it completes. For more information about creating job templates, see **About job templates** on page 832.
   a. Create a selectable master template, called `ds` in the example below, that creates a stage in and stage out system job. This name should match the `DEFAULT_TEMPLATE` value in `ds_config.py`. See **Configuring data staging with advanced options** on page 899 for more information.
   b. For the data staging in job template, called `dsin` in the example below, specify that it will create a data staging job by setting **DATASTAGINGJOB** to *TRUE*. Note that the name of this job template must match the name of the data stage in job template referenced in the master template.
   c. Set the bandwidth **GRES** to the amount of bandwidth a single stage in job should use. This indicates how many of the bandwidth units specified with **NODECFG**`[GLOBAL]` in **Configuring data staging** on page 884 a data staging job with this template should consume.
   d. Add **FLAGS**=*GRESONLY* to indicate that this data staging job does not require any compute resources.
   e. Create a trigger that executes the `ds_move_scp`, `ds_move_rsync`, or `ds_move_multiplex` script, depending on which file transfer utility you use. Set the *attacherror*, *objectxmlstdin*, and *user* **FLAG**s to attach any trigger stderr as a message to the job, pass the job XML to the script, and indicate that the script should run as the job's user, respectively.

   > If you use the rsync protocol, you can configure your data staging jobs to report the actual number of bytes transferred and the total data size to be transferred. To do so, use the Sets attribute to `^BYTES_IN.^DATA_SIZE_IN` for stage in jobs and `^BYTES_OUT.^DATA_SIZE_OUT` for stage out jobs. For example, a stage in trigger would look like the following:
   >
   > ```
   > JOBCFG[dsin]
   > TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
   > move_rsync --stagein",Flags=objectxmlstdin:user:attacherror,Sets=^BYTES_
   > IN.^DATA_SIZE_IN
   > ```
   >
   > A stage out trigger would look like the following:
   >
   > ```
   > JOBCFG[dsout]
   > TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
   > move_rsync --stageout",Flags=objectxmlstdin:user:attacherror,Sets=^BYTES_
   > OUT.^DATA_SIZE_OUT
   > ```
   >
   > These variables show up as events if you set your WIKIEVENTS parameter to *TRUE*.

   f. Create the stage out job, called `dsout` in the example below, by repeating steps 2b - 2e in a new template. In the example below, this template is called `dsout`. Note that the name of this job template must match the name of the data stage out job template referenced in the data staging master template.

```
JOBCFG[ds]      TEMPLATEDEPEND=AFTEROK:dsin TEMPLATEDEPEND=BEFORE:dsout
SELECT=TRUE

JOBCFG[dsin]    DATASTAGINGSYSJOB=TRUE
JOBCFG[dsin]    GRES=bandwidth:2
JOBCFG[dsin]    FLAGS=GRESONLY
JOBCFG[dsin]    TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_rsync --stagein",Flags=attacherror:objectxmlstdin:user

JOBCFG[dsout]   DATASTAGINGSYSJOB=TRUE
JOBCFG[dsout]   GRES=bandwidth:1
JOBCFG[dsout]   FLAGS=GRESONLY
JOBCFG[dsout]   TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_rsync --stageout",Flags=attacherror:objectxmlstdin:user
```

3. Create the job using msub, adding resources and specifying a script as you normally would. Then configure Moab to stage the data for it. To do so:

   a. At the end of the command, use the `--stagein`/`--stageout` option and/or `--stageinfile`/`--stageoutfile` option.

      - The `--stagein`/`--stageout` option lets you specify a single file or directory to stage in or out. You must set the option equal to *`<source>%<destination>`*, where *<source>* and *<destination>* are both `[<user>@]<host>:/<path>/[<fileName>]`. See **Staging a file or directory** on page 303 for format and details.

        > If the destination partition is down or does not have configured resources, the data staging workflow submission will fail.

        ```
        > msub --stagein=annasmith@labs:/patient-
        022678/%davidharris@university:/davidharris/research/patientrecords
        <jobScript>
        ```

        Moab copies the `/patient-022678` directory from the hospital's `labs` server to the `university` cluster where the job will run prior to job start.

      - The `--stageinfile`/`--stageoutfile` option lets you specify a file that contains the file and/or directory name(s) to stage in or out. You must set the option equal to *`<path>/<fileName>`* of the file. The file must contain at least one line with this format: *`<source>%<destination>`*, where both *<source>* and *<destination>* are `[<user>@]<host>:/<path>[<fileName>]`. See **Staging multiple files or directories** on page 304 for more information.

        > If the destination partition is down or does not have configured resources, the data staging workflow submission will fail.

        ```
        > msub --stageinfile=/davidharris/research/recordlist <jobScript>
        ```

        Moab copies all files specified in the `/davidharris/research/recordlist` file to the cluster where the job will run prior to job start.

/davidharris/research/recordlist:

```
annasmith@labs:/patient-
022678/tests/blood02282014%davidharris@university:/davidharris/research/patie
ntrecords/blood02282014
annasmith@labs:/patient-
022678/visits/stats02032014%davidharris@university:/davidharris/research/pati
entrecords/stats02032014
annasmith@labs:/patient-
022678/visits/stats02142014%davidharris@university:/davidharris/research/pati
entrecords/stats02142014
annasmith@labs:/patient-
022678/visits/stats02282014%davidharris@university:/davidharris/research/pati
entrecords/stats02282014
annasmith@labs:/patient-
022678/visits/stats03032014%davidharris@university:/davidharris/research/pati
entrecords/stats03032014
annasmith@labs:/patient-
022678/visits/stats03142014%davidharris@university:/davidharris/research/pati
entrecords/stats03142014
annasmith@labs:/patient-
022678/visits/stats03282014%davidharris@university:/davidharris/research/pati
entrecords/stats03282014
```

Moab copies the seven patient record files from the hospital's `labs` server to the `university` cluster where the job will run prior to job start.

b. The `--stageinsize`/`--stageoutsize` option lets you specify the estimated size of the files and/or directories to help Moab more quickly and accurately calculate the amount of time it will take to stage the data and therefore schedule your job correctly. If you are staging data out, then setting `--stageoutsize` is required. If you provide an integer, Moab will assume the number is in megabytes. To change the unit, add another suffix. See **Stage in or out file size** on page 305 for more information.

```
> msub --stageinfile=/davidharris/research/recordlist --stageinsize=100
<jobScript>
```

Moab copies the `/davidharris/research/recordlist` file, which is approximately 100 megabytes, from the biology node to the host where the job will run prior to job start.

4. To see the status, errors, and other details associated with your data staging job, run checkjob `-v`. See "checkjob" for details.

Related topics

- **About data staging** on page 880
- **Configuring data staging** on page 884
- **Configuring data staging with advanced options** on page 899
- **Staging data to or from a shared file system in a grid** on page 890
- **Staging data to or from a compute node** on page 895
- **Sample user job script** on page 901

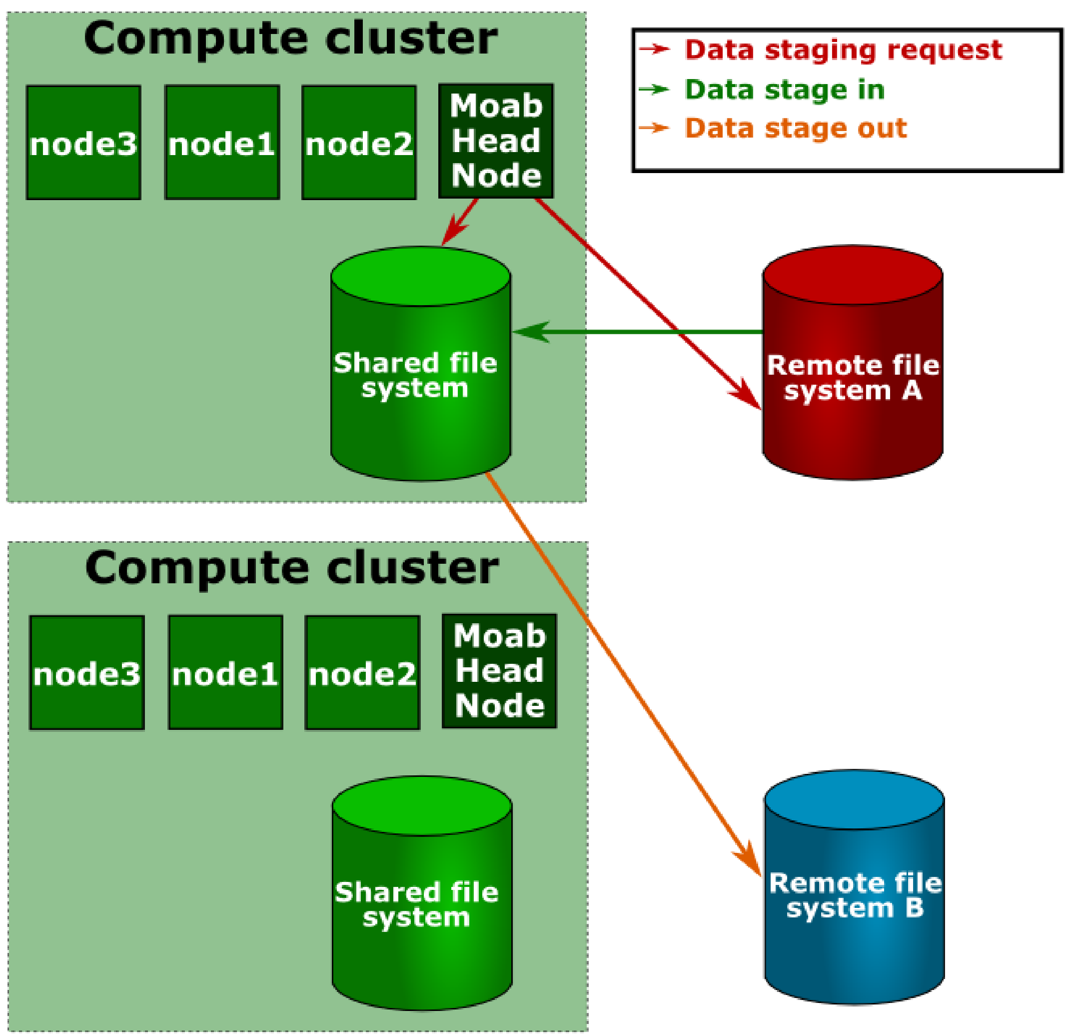
Staging data to or from a shared file system in a grid

Context

You can stage data in an environment where multiple instances of Moab run in a grid configuration. For this type of data staging, each cluster utilizes a shared file system with all compute nodes. This type of data staging will make data available to a job that will run on a set of nodes in one of the clusters in the grid. You must specify where the remote data can be obtained with a username, host name, and path to a file or directory and where on the shared storage Moab will store the data. The remote data source location is known at job submission time, but you must use the *$CLUSTERHOST* placeholder for the host name of the data transfer server on which the job will be scheduled. After the job runs, you can also copy data from the cluster shared file system to a remote file system.

Note that you cannot stage data to or from a local compute node with its own local storage in a grid environment.

Image 3-14: Data staging in a grid



To stage data to or from a shared file system in a grid

1. If you have not already done so, configure your SSH keys and `moab.cfg` to support data staging. See **Configuring the SSH keys for the data staging transfer script** on page 882 and **Configuring data staging** on page 884 for more information.
2. Create your job templates for data staging jobs in `moab.cfg`. The templates in the example below create a compute job that stages data in before it starts and stages data out when it completes. For more information about creating job templates, see **About job templates** on page 832.
   a. Create a selectable master template, called `ds` in the example below, that creates a stage in and stage out system job. This name should match the `DEFAULT_TEMPLATE` value in `ds_config.py`. For more information, see **Configuring data staging with advanced options** on page 899.
   b. For the data staging in job template, called `dsin` in the example below, specify that it will create a data staging job by setting **DATASTAGINGJOB** to *TRUE*. Note that the name of this job template must match the name of the data stage in job template referenced in the master template.
   c. Set the staging job template bandwidth **GRES** to the amount of bandwidth a single stage in job should use. This indicates how many of the bandwidth units specified with **NODECFG**`[GLOBAL]` in **Configuring data staging** on page 884 a data staging job with this template should consume.
   d. Set **JOBMIGRATEPOLICY** to *JUSTINTIME*.
   e. Add **FLAGS**=*GRESONLY* to indicate that this data staging job does not require any compute resources.
   f. Create a trigger that executes the `ds_move_scp`, `ds_move_rsync`, or `ds_move_multiplex` script, depending on which file transfer utility you use. Set the *attacherror*, *objectxmlstdin*, and *user* **FLAG**s to attach any trigger stderr as a message to the job, pass the job XML to the script, and indicate that the script should run as the job's user, respectively.

   If you use the rsync protocol, you can configure your data staging jobs to report the actual number of bytes transferred and the total data size to be transferred. To do so, use the Sets attribute to `^BYTES_IN.^DATA_SIZE_IN` for stage in jobs and `^BYTES_OUT.^DATA_SIZE_OUT` for stage out jobs. For example, a stage in trigger would look like the following:

```
JOBCFG[dsin]
TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
move_rsync --stagein",Flags=objectxmlstdin:user:attacherror,Sets=^BYTES_
IN.^DATA_SIZE_IN
```

   A stage out trigger would look like the following:

```
JOBCFG[dsout]
TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
move_rsync --stageout",Flags=objectxmlstdin:user:attacherror,Sets=^BYTES_
OUT.^DATA_SIZE_OUT
```

   These variables show up as events if you set your WIKIEVENTS parameter to *TRUE*.

g. Create the stage out job, called `dsout` in the example below, by repeating steps 2b - 2f in a new template. In the example below, this template is called `dsout`. Note that the name of this job template must match the name of the data stage out job template referenced in the master template.

```
JOBCFG[ds]      TEMPLATEDEPEND=AFTEROK:dsin TEMPLATEDEPEND=BEFORE:dsout
SELECT=TRUE

JOBCFG[dsin]    DATASTAGINGSYSJOB=TRUE
JOBCFG[dsin]    GRES=bandwidth:2
JOBCFG[dsin]    FLAGS=GRESONLY
JOBCFG[dsin]    TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_rsync --stagein",Flags=attacherror:objectxmlstdin:user

JOBCFG[dsout]   DATASTAGINGSYSJOB=TRUE
JOBCFG[dsout]   FLAGS=GRESONLY
JOBCFG[dsout]   INHERITRES=TRUE
JOBCFG[dsout]   TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_rsync --stageout",Flags=attacherror:objectxmlstdin:user
```

3. Create the job using msub, adding resources and specifying a script as you normally would. Then configure Moab to stage the data for it. To do so:

   a. At the end of the command, use the `--stagein`/`--stageout` option and/or `--stageinfile`/`--stageoutfile` option.

   - The `--stagein`/`--stageout` option lets you specify a single file or directory to stage in or out. You must set the option equal to `<source>%<destination>`, where *<source>* and *<destination>* are both `[<user>@]<host>:/<path>/[<fileName>]`. See **Staging a file or directory** on page 303 for format and details.

     Note that if you do not know the cluster where the job will run but want the data staged to the same location, you can use the *$CLUSTERHOST* variable in place of a host. If you choose to use the *$CLUSTERHOST* variable, you must first customize the `ds_config.py` file. For more information, see **Configuring the $CLUSTERHOST variable** on page 894.

     If the destination partition is down or does not have configured resources, the data staging workflow submission will fail.

     ```
     > msub ... --stagein=annasmith@labs:/patient-
     022678/%\$CLUSTERHOST:/davidharris/research/patientrecords <jobScript>
     ```

     Moab copies the `/patient-022678` directory from the hospital's `labs` server to the cluster where the job will run prior to job start.

   - The `--stageinfile`/`--stageoutfile` option lets you specify a file that contains the file (s) and directory(-ies) to stage in or out. You must set the option equal to `<path>/<fileName>` of the file. The file must contain at least one line with this format: `[<user>@]<host>:/<path>[<fileName>]`. See **Staging multiple files or directories** on page 304 for more information.

> If the destination partition is down or does not have configured resources, the data staging workflow submission will fail.

```
> msub ... --stageinfile=/davidharris/research/recordlist <jobScript>
```

Moab copies all files specified in the /davidharris/research/recordlist file to the cluster where the job will run prior to job start.

/davidharris/research/recordlist:

```
 annasmith@labs:/patient-
022678/tests/blood02282014%$CLUSTERHOST:/davidharris/research/patientrecords/b
lood02282014
 annasmith@labs:/patient-
022678/visits/stats02032014%$CLUSTERHOST:/davidharris/research/patientrecords/
stats02032014
 annasmith@labs:/patient-
022678/visits/stats02142014%$CLUSTERHOST:/davidharris/research/patientrecords/
stats02142014
 annasmith@labs:/patient-
022678/visits/stats02282014%$CLUSTERHOST:/davidharris/research/patientrecords/
stats02282014
 annasmith@labs:/patient-
022678/visits/stats03032014%$CLUSTERHOST:/davidharris/research/patientrecords/
stats03032014
 annasmith@labs:/patient-
022678/visits/stats03142014%$CLUSTERHOST:/davidharris/research/patientrecords/
stats03142014
 annasmith@labs:/patient-
022678/visits/stats03282014%$CLUSTERHOST:/davidharris/research/patientrecords/
stats03282014
```

Moab copies the seven patient record files from the hospital's labs server to the cluster where the job will run prior to job start.

b. The --stageinsize/--stageoutsize option lets you specify the estimated size of the files and/or directories to help Moab more quickly and accurately calculate the amount of time it will take to stage the data and therefore schedule your job correctly. If you used the *$CLUSTERHOST* variable to stage in, then setting --stageinsize is required. --stageoutsize is always required for staging data out. If you provide an integer, Moab will assume the number is in megabytes. To change the unit, add another suffix. See **Stage in or out file size** on page 305 for more information.

```
> msub ... --stageinfile=/davidharris/research/recordlist --stageinsize=100
<jobScript>
```

Moab copies the /davidharris/research/recordlist file, which is approximately 100 megabytes, from the biology node to the host where the job will run prior to job start.

4. To see the status, errors, and other details associated with your data staging job, run checkjob -v. See "checkjob" for details.

Related topics

- **About data staging** on page 880
- **Configuring data staging** on page 884

- **Configuring data staging with advanced options** on page 899
- **Sample user job script** on page 901

Configuring the $CLUSTERHOST variable

Context

When you submit a data staging job in a grid environment, you can use the *$CLUSTERHOST* variable instead of specifying a destination if you do not know the cluster where the job will run but want the data staged to the same location. Before the variable will work correctly, you must first configure it by customizing the `ds_config.py` script to match your unique system.

> **Use case**
>
> In a grid with three clusters, you have a partition named `master` where you want all data staged to a host named `gridheadNAS`; a partition named `csdept` where you want all data staged to a host named `fs001.cs.example.edu`; and a partition named `lab` where you want all data staged to a host named `bigfilesystem`.

To configure the $CLUSTERHOST variable

1. Open the `ds_config.py` file for modification. It is located in `/opt/moab/tools/data-staging/` by default.

```
[moab]$ vi /opt/moab/tools/data-staging/ds_config.py
```

2. Locate the `PARTITION_TO_HOST` parameter.

```
...
PARTITION_TO_HOST = {"partition_1_name":"cluster_1_staging_hostname",
                     "partition_2_name":"cluster_2_staging_hostname",
                     "partition_3_name":"cluster_3_staging_hostname"}
...
```

3. Replace the partition names and associated cluster hostnames with those that match your system. For the use case provided above, you would customize it the following way:

```
...
PARTITION_TO_HOST = {"master":"gridheadNAS",
                     "csdept":"fs001.cs.example.edu",
                     "lab":"bigfilesystem"}
...
```

Related topics

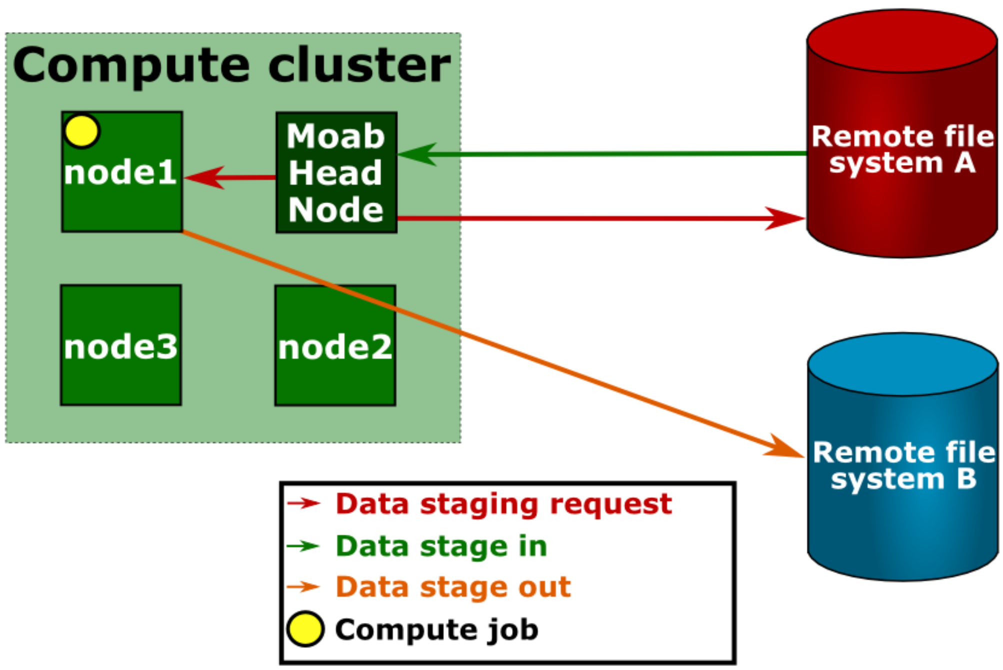
- **Staging data to or from a shared file system in a grid** on page 890

Staging data to or from a compute node

Context

You can stage data to or from a local compute node in an environment where each node on the cluster has local storage. This type of data staging will make data stored outside the cluster available to a job that will run on a single node in the cluster. You must specify the username, host name, and path to a file or directory and a location on the compute node where Moab will store the data. You will supply the remote data source location at job submission time, but you must use the *$JOBHOST* placeholder for the name of the compute node. After the job runs, you can also copy data from the local file system to a remote file system.

Image 3-15: Data staging to or from a local compute node



Before staging data to or from a local compute node, please follow the procedure in **Configuring data staging** on page 884.

To stage data to or from a local compute node

1. If you have not already done so, configure your SSH keys and `moab.cfg` to support data staging. See **Configuring the SSH keys for the data staging transfer script** on page 882 and **Configuring data staging** on page 884 for more information.
2. Create your job templates for data staging jobs in `moab.cfg`. The templates in the example below create a compute job that stages data in before it starts and stages data out when it completes. For more information about creating job templates, see **About job templates** on page 832.
    a. Create a selectable master template, called `ds` in the example below, that creates a stage in and stage out system job. This name should match the `DEFAULT_TEMPLATE` value in `ds_config.py`.

For more information, see **Configuring data staging with advanced options** on page 899.

b. For the data staging in job template, called `dsin` in the example below, specify that it will create a data staging job by setting **DATASTAGINGJOB** to *TRUE*. Note that the name of this job template must match the name of the data stage in job template referenced in the master template.

c. Set the staging job template bandwidth **GRES** to the amount of bandwidth a single stage in job should use. This indicates how many of the bandwidth units specified with **NODECFG**`[GLOBAL]` in **Configuring data staging** on page 884 a data staging job with this template should consume.

d. For local node data staging it is important that the data staging job has the entire node to itself. To prevent Moab from scheduling another job on the node at the same time as the data staging job, set the **NODEACCESSPOLICY** to *SINGLEJOB* in the staging job template.

e. Add **INHERITRES**=*TRUE* to reserve the compute node for the data staging job to prevent other compute jobs from using the node at the same time and creating input, output, and disk conflicts with the data staging job.

f. Create a trigger that executes the `ds_move_scp`, `ds_move_rsync`, or `ds_move_multiplex` script, depending on which file transfer utility you use. Set the *attacherror*, *objectxmlstdin*, and *user* **FLAG**s to attach any trigger stderr as a message to the job, pass the job XML to the script, and indicate that the script should run as the job's user, respectively.

> If you use the rsync protocol, you can configure your data staging jobs to report the actual number of bytes transferred and the total data size to be transferred. To do so, use the Sets attribute to `^BYTES_IN.^DATA_SIZE_IN` for stage in jobs and `^BYTES_OUT.^DATA_SIZE_OUT` for stage out jobs. For example, a stage in trigger would look like the following:
>
> ```
> JOBCFG[dsin]
> TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
> move_rsync --stagein",Flags=objectxmlstdin:user:attacherror,Sets=^BYTES_
> IN.^DATA_SIZE_IN
> ```
>
> A stage out trigger would look like the following:
>
> ```
> JOBCFG[dsout]
> TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
> move_rsync --stageout",Flags=objectxmlstdin:user:attacherror,Sets=^BYTES_
> OUT.^DATA_SIZE_OUT
> ```
>
> These variables show up as events if you set your WIKIEVENTS parameter to *TRUE*.

g. Create the stage out job, called `dsout` in the example below, by repeating steps 2b - 2f in a new template. In the example below, this template is called `dsout`. Note that the name of this job template must match the name of the data stage out job template referenced in the data staging master template.

```
JOBCFG[ds]      TEMPLATEDEPEND=AFTEROK:dsin TEMPLATEDEPEND=BEFORE:dsout
SELECT=TRUE

JOBCFG[dsin]    DATASTAGINGSYSJOB=TRUE
JOBCFG[dsin]    GRES=bandwidth:2
JOBCFG[dsin]    NODEACCESSPOLICY=SINGLEJOB
JOBCFG[dsin]    INHERITRES=TRUE
JOBCFG[dsin]    TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_rsync --stagein",Flags=attacherror:objectxmlstdin:user

JOBCFG[dsout]   DATASTAGINGSYSJOB=TRUE
JOBCFG[dsout]   GRES=bandwidth:1
JOBCFG[dsout]   NODEACCESSPOLICY=SINGLEJOB
JOBCFG[dsout]   INHERITRES=TRUE
JOBCFG[dsout]   TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_rsync --stageout",Flags=attacherror:objectxmlstdin:user
```

3. Create the job using msub, adding resources and specifying a script as you normally would. Then configure Moab to stage the data for it. To do so:

   a. If the compute job does not use all of the node's processors, Moab could schedule another job on the node at the same time. If you did not set **NODEACCESSPOLICY** to *SINGLEJOB* in your `moab.cfg`, set the policy for this job by adding `-l naccesspolicy=singlejob` to your msub command.

   ```
   > msub -l naccesspolicy=singlejob... <jobScript>
   ```

   b. At the end of the command, use the `--stagein`/`--stageout` option and/or `--stageinfile`/`--stageoutfile` option.

   - The `--stagein`/`--stageout` option lets you specify a single file or directory to stage in or out. You must set the option equal to `<source>%<destination>`, where *<source>* and *<destination>* are both `[<user>@]<host>:/<path>/[<fileName>]`. See **Staging a file or directory** on page 303 for format and details.

     If the destination partition is down or does not have configured resources, the data staging workflow submission will fail.

     If you do not know the host where the job will run but want the data staged to the same location, you can use the *$JOBHOST* variable in place of a host.

     ```
     > msub --stagein=annasmith@labs:/patient-
     022678/%\$JOBHOST:/davidharris/research/patientrecords <jobScript>
     ```

     Moab copies the `/patient-022678` directory from the hospital's `labs` server to the node where the job will run prior to job start.

   - The `--stageinfile`/`--stageoutfile` option lets you specify a file that contains the file and directory name(s) to stage in or out. You must set the option equal to `<path>/<fileName>%` of the file. The file must contain at least one line with this format: `<source>%<destination>`, where *<source>* and *<destination>* are both `[<user>@]<host>:/<path>[/<fileName>]`. See **Staging multiple files or directories** on page 304 for more information.

> If the destination partition is down or does not have configured resources, the data staging workflow submission will fail.

```
> msub --stageinfile=/davidharris/research/recordlist <jobScript>
```

Moab copies all files specified in the /davidharris/research/recordlist file to the host where the job will run prior to job start.

/davidharris/research/recordlist:

```
annasmith@labs:/patient-
022678/tests/blood02282014%$JOBHOST:/davidharris/research/patientrecords/blood
02282014
annasmith@labs:/patient-
022678/visits/stats02032014%$JOBHOST:/davidharris/research/patientrecords/stat
s02032014
annasmith@labs:/patient-
022678/visits/stats02142014%$JOBHOST:/davidharris/research/patientrecords/stat
s02142014
annasmith@labs:/patient-
022678/visits/stats02282014%$JOBHOST:/davidharris/research/patientrecords/stat
s02282014
annasmith@labs:/patient-
022678/visits/stats03032014%$JOBHOST:/davidharris/research/patientrecords/stat
s03032014
annasmith@labs:/patient-
022678/visits/stats03142014%$JOBHOST:/davidharris/research/patientrecords/stat
s03142014
annasmith@labs:/patient-
022678/visits/stats03282014%$JOBHOST:/davidharris/research/patientrecords/stat
s03282014
```

Moab copies the seven patient record files from the hospital's labs server to the host where the job will run prior to job start.

c. The --stageinsize/--stageoutsize option lets you specify the estimated size of the files and/or directories to help Moab more quickly and accurately calculate the amount of time it will take to stage the data and therefore schedule your job correctly. If you used the *$JOBHOST* variable to stage in, then setting --stageinsize is required. --stageoutsize is always required for staging data out. If you provide an integer, Moab will assume the number is in megabytes. To change the unit, add another suffix. See **Stage in or out file size** on page 305 for more information.

```
> msub --stageinfile=/davidharris/research/recordlist --stageinsize=100
<jobScript>
```

Moab copies the /davidharris/research/recordlist file, which is approximately 100 megabytes, from the biology node to the host where the job will run prior to job start.

4. To see the status, errors, and other details associated with your data staging job, run checkjob -v. See "checkjob" for details.

> Your checkjob output may include a warning that says "req 1 RM (internal) does not match job destination RM". You can safely ignore this message.

Related topics

- **About data staging** on page 880
- **Configuring data staging** on page 884
- **Configuring data staging with advanced options** on page 899
- **Staging data to or from a compute node** on page 895
- **Sample user job script** on page 901

Configuring data staging with advanced options

Using a different default template name

When you submit a data staging job, a data staging job template is attached to the job automatically. In the reference script configuration, the default template name is `ds`. This is the template that will be attached to the compute job by the client msub filter.

If you would like to change the name of the default template that is automatically attached, you should change the value of `DEFAULT_TEMPLATE` in the `ds_config.py` file installed on all client submit hosts. This name must match the master data staging template name specified in the Moab configuration file.

To configure the DEFAULT_TEMPLATE variable

1. Open the `ds_config.py` file for modification. It is located in `/opt/moab/tools/data-staging/` by default.

```
[moab]$ vi /opt/moab/tools/data-staging/ds_config.py
```

2. Locate the `DEFAULT_PARTITION` parameter.

```
...
DEFAULT_TEMPLATE = "ds"
...
```

3. Replace the template name with the one specified in the Moab configuration file.

```
ds_config.py
...
DEFAULT_TEMPLATE = "datastaging"
...

moab.cfg
...
JOBCFG[datastaging] TEMPLATEDEPEND=...
```

4. Make these changes on all client submit hosts.

Supporting multiple file transfer script utilities in a grid on a per-partition basis

If you want a different transfer script to run based on which partition the job is submitted to, you can configure a multiplexer script that will switch execution to various other scripts based on the partition.

To support multiple file transfer script utilities in a grid on a per-partition basis

1. Configure the trigger in your job templates in `moab.cfg` to run `ds_move_multiplex` instead of `ds_move_rsync` or `ds_move_scp`.
2. Configure the `PARTITION_TO_SCRIPT` variable in `ds_config.py` to provide a mapping from each partition to the desired script to run.
   a. Open the `ds_config.py` file for modification. It is located in `/opt/moab/tools/data-staging/` by default.

   ```
   [moab]$ vi /opt/moab/tools/data-staging/ds_config.py
   ```

   b. Locate the `PARTITION_TO_SCRIPT` parameter.

   ```
   ...
   PARTITION_TO_SCRIPT =
   {"partition_1_name":"/opt/moab/tools/data-staging/ds_move_rsynch",
    "partition_2_name":"/opt/moab/tools/data-staging/ds_move_scp",
    "partition_3_name":"/opt/moab/tools/data-staging/ds_move_rsync"}
   ...
   ```

   c. Replace the `partition_*_name`s with partitions that exist in your configuration. After each partition, specify the script that you want to execute for that partition.

Receiving notification at the completion of the data staging job

If you want explicit notification in case of failure of the stage out job, add an additional trigger to the dsout job template which will send email notification to the job's submitter. For more information, see **Using a trigger to send email** on page 731.

```
JOBCFG[dsout]  DATASTAGINGSYSJOB=TRUE
JOBCFG[dsout]  GRES=bandwidth:1
JOBCFG[dsout]  FLAGS=GRESONLY
JOBCFG[dsout]  TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
move_rsync --stageout",Flags=attacherror:objectxmlstdin:user
JOBCFG[dsout]  TRIGGER=EType=fail,AType=mail,Action="Your (stageout) data staging job
$OID failed."
```

The first trigger listed in the template configuration should be the exec trigger. Add the email trigger and any other triggers after the exec trigger. You can modify the email trigger to run at completion rather than at failure. You can also add this type of trigger to stage in jobs.

Adding a non-default template via msub

You can have multiple data staging template workflows defined in the `moab.cfg`. The submit filter is configured to add only one of them by default. If you wish to use one of the other available templates, you can do so by using the `-l template=TEMPLATENAME` option in the msub command:

Given the following `moab.cfg`:

```
#Default data staging template:

JOBCFG[ds]      TEMPLATEDEPEND=AFTEROK:dsin TEMPLATEDEPEND=BEFORE:dsout SELECT=TRUE
JOBCFG[dsin]    DATASTAGINGSYSJOB=TRUE
JOBCFG[dsin]    GRES=bandwidth:2
JOBCFG[dsin]    FLAGS=GRESONLY
JOBCFG[dsin]    TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
move_rsync --stagein",Flags=attacherror:objectxmlstdin:user

JOBCFG[dsout]   DATASTAGINGSYSJOB=TRUE
JOBCFG[dsout]   GRES=bandwidth:1
JOBCFG[dsout]   FLAGS=GRESONLY
JOBCFG[dsout]   TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
move_rsync --stageout",Flags=attacherror:objectxmlstdin:user

#experimental data staging template:

JOBCFG[dscustom]      TEMPLATEDEPEND=AFTEROK:dscustomin
TEMPLATEDEPEND=BEFORE:dscustomout SELECT=TRUE
JOBCFG[dscustomin]    DATASTAGINGSYSJOB=TRUE
JOBCFG[dscustomin]    GRES=bandwidth:2
JOBCFG[dscustomin]    FLAGS=GRESONLY
JOBCFG[dscustomin]    TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_custom --stagein",Flags=attacherror:objectxmlstdin:user

JOBCFG[dscustomout]   DATASTAGINGSYSJOB=TRUE
JOBCFG[dscustomout]   GRES=bandwidth:1
JOBCFG[dscustomout]   FLAGS=GRESONLY
JOBCFG[dscustomout]   TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_custom --stageout",Flags=attacherror:objectxmlstdin:user
```

The user could submit a job using the custom data staging template with the following command:

```
[moab]$ msub -l template=dscustom …
```

Using msub to return all the job IDs in the workflow at submission time

By default, msub will print the job ID to stdout at the time of submission. If you would like to have msub print all of the jobs that are created as part of the data staging workflow template, you can use the msub `--workflowjobids` option to show all the job IDs at submission time:

```
$ echo sleep 60 | msub -l walltime=15 --workflowjobids

MoabA.3.dsin MoabA.3 MoabA.3.dsout
```

This could be useful if you are writing scripts to do your own workflows and you need to programmatically capture the data stage out job name for use in your workflow.

Related topics

- **Configuring data staging** on page 884

References

Sample user job script

The code below is an example of a job script that a user might use to run a data staging job.

```
#!/bin/bash
#
# Sample data staging job script
#
# stage in directives
#MSUB --stageinsize=1MB
#MSUB --stagein=davidharris@source-server:/tmp/filein.tostage%davidharris@destination-
server:/tmp/filein.staged
#
# stage out directives
#MSUB --stageoutsize=10GB
#MSUB --stageout=davidharris@destination-
server:/tmp/fileout.tostage%davidharris@source-server:/tmp/fileout.staged

# run executable on the destination host using staged data
$HOME/bin/my_compute_executable < /tmp/filein.staged > /tmp/fileout.tostage
```

Related topics

- **About data staging** on page 880
- **Staging data** on page 303

Appendices

Appendix A: Moab Parameters

See the Parameters Overview in the Moab Admin Manual for further information about specifying parameters.

Index: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

ACCOUNTCFG[<ACCOUNTID>]	
Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: General Credential Flags, **CHARGERATE**, PRIORITY, ENABLEPROFILING, MEMBERULIST, **PLIST**, **QDEF**, **QLIST**, usage limit, or a fairness usage limit specification (FSCAP, FSTARGET, and FSWEIGHT).
Default	---
Description	Specifies account specific attributes. See the account overview for general information and the job flag overview for a description of legal flag values.
Example	`ACCOUNTCFG[projectX] MAXJOB=50 QDEF=highprio` *Up to 50 jobs submitted under the account ID projectX will be allowed to execute simultaneously and will be assigned the QOS highprio by default.*

ACCOUNTINGINTERFACEURL	
Format	*<URL>* where protocol can be one of **exec** or **file**
Default	---
Description	Specifies the interface to use for real-time export of Moab accounting/auditing information. See Exporting Events in Real-Time for more information.
Example	`ACCOUNTINGINTERFACEURL exec:///$TOOLSDIR/dumpacc.pl`

ACCOUNTWEIGHT	
Format	*<INTEGER>*
Default	*1*
Description	Specifies the priority weight to be applied to the specified account priority. See Credential (CRED) Factor.
Example	`ACCOUNTWEIGHT 100`

ADMIN1, ADMIN2, ADMIN3	
Description	These parameters are deprecated. Use ADMINCFG.

ADMINCFG[X]	
Format	One or more *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: **ENABLEPROXY**, **USERS**, **GROUPS**, SERVICES, or **NAME**
Default	---
Description	Allows a site to configure which services and users belong to a particular level of administration. **Note:** The first user listed in the **ADMINCFG[1]** users list is considered to be the primary admin. The option ***USERS**=ALL* is allowed. The groups list adds the groups' users as if they were listed individually as **USERS**. To prevent Moab from assigning a primary user from the first group listed, you must specify a primary user first using the **USERS** attribute, then list the desired groups.
Example	`ADMINCFG[1] USERS=root,john` `ADMINCFG[1] GROUPS=admin` `ADMINCFG[1] SERVICES=ALL` `ADMINCFG[1] NAME=batchadmin` `ADMINCFG[3] USERS=bob,carol,smoore` `ADMINCFG[3] GROUPS=science,math` `ADMINCFG[3] SERVICES=mjobctl,mcredctl,runjob` `ADMINCFG[3] NAME=helpdesk` *Members of the batchadmin admin role and members of the admin group are allowed to run all commands. Members of the helpdesk role and science and math groups are allowed to run mjobctl. They are also able to view and modify credential objects (i.e. users, groups, accounts, etc.) See the security overview for more details.* `ADMINCFG[4] USERS=ALL SERVICES=checknode` *All users can execute mdiag -n or checknode to get information on any node.*

AGGREGATENODEACTIONS	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Consolidates queued node actions into as few actions as possible to reduce communication burden with resource manager. Node actions are queued until the AGGREGATENODEACTIONSTIME setting. This may delay some node actions. When reprovisioning, the system job may expire before the provision action occurs; while the action will still occur, the job will not show it.
Example	`AGGREGATENODEACTIONS TRUE` *Queues node actions together when possible.*

AGGREGATENODEACTIONSTIME	
Format	*<SECONDS>*
Default	*60*
Description	The delay time for the AGGREGATENODEACTIONS parameter to aggregate requests before sending job batches.
Example	`AGGREGATENODEACTIONSTIME 120` *Sets the AGGREGATENODEACTIONS delay to two minutes.*

ALLOWMULTIREQNODEUSE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	By default Moab does not allow different requirements on the same job to occupy the same node. For example, if a job is submitted with `nodes=2:ppn=8+4:fast:ppn=16`, it's possible that some of the tasks requested could overlap onto the same node. This parameter instructs Moab to allow overlapping the same node, or not. This parameter also applies to the various -w clauses of an **mshow -a** on page 276 command.
Example	`ALLOWMULTIREQNODEUSE TRUE`

ALLOWROOTJOBS	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether batch jobs from the root user (UID=0) are allowed to be executed. **Note**: The resource manager must also support root jobs.
Example	`ALLOWROOTJOBS TRUE` *Jobs from the root user can execute.*

ALLOWVMMIGRATION	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Enables Moab to migrate VMs.
Example	`ALLOWVMMIGRATION TRUE`

ALWAYSEVALUATEALLJOBS	
Format	*<BOOLEAN>*
Default	FALSE
Description	When scheduling priority jobs, Moab stops scheduling when it encounters the first job that cannot run and cannot get a reservation. **ALWAYSEVALUATEALLJOBS** directs Moab to continue scheduling until all priority jobs (jobs that do not violate any limits) are evaluated.
Example	`ALWAYSEVALUATEALLJOBS TRUE`

AMCFG	
Format	One or more key-value pairs as described in the Allocation Manager Configuration Overview.
Default	---
Description	Specifies the interface and policy configuration for the scheduler-allocation manager interface. Described in detail in the Allocation Manager Configuration Overview.
Example	`AMCFG[mam] SERVER=mam://master.ufl.edu STARTFAILUREACTION=HOLD TIMEOUT=15`

APPLICATIONLIST	
Format	Space-delimited list of generic resources.
Default	---
Description	Specifies which generic resources represent actual applications on the cluster/grid. See Managing Consumable Generic Resources for more information.
Example	`NODECFG[node01] GRES=calclab:1,powerhouse:1 RCSOFTWARE=calclab:1,powerhouse:1` `NODECFG[node02] GRES=calclab:1,powerhouse:1 RCSOFTWARE=calclab:1,powerhouse:1` `APPLICATIONLIST calclab,powerhouse` *The generic resources calclab and powerhouse will now be recognized and treated as application software.*

ARRAYJOBPARLOCK	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, all sub jobs of an array are locked to a single partition. The default behavior when scheduling array sub jobs is to span the jobs across partitions when possible. The ARRAYJOBPARLOCK job flag can be used to specify partition locking at submit time. The ARRAYJOBPARSPAN job flag overrides this parameter.
Example	`ARRAYJOBPARLOCK TRUE`

ASSIGNVLANFEATURES	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	When set to *TRUE*, this forces all VMs to be contained in VLANs.
Example	`ASSIGNVLANFEATURES TRUE`

ATTRATTRWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight to be applied to jobs with the specified job attribute. See Attribute (ATTR) Factor.
Example	`ATTRATTRWEIGHT 100`

ATTRGRESWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight to be applied to jobs requesting the specified generic resource. See Attribute (ATTR) Factor.
Example	`ATTRGRESWEIGHT 200`

ATTRSTATEWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight to be applied to jobs with the specified job state. See Attribute (ATTR) Factor.
Example	`ATTRSTATEWEIGHT 200`

ATTRWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the priority component weight to be applied to the ATTR subcomponents. See Attribute (ATTR) Factor.
Example	`ATTRWEIGHT      2` `ATTRSTATEWEIGHT 200`

BACKFILLDEPTH	
Format	*<INTEGER>*
Default	*0* (no limit)
Description	Specifies the number of idle jobs to evaluate for backfill. The backfill algorithm will evaluate the top *<X>* priority jobs for scheduling. By default, all jobs are evaluated.
Example	`BACKFILLDEPTH 128` *Evaluate only the top 128 highest priority idle jobs for consideration for backfill.*

BACKFILLPOLICY	
Format	One of *FIRSTFIT* or *NONE*
Default	*FIRSTFIT*
Description	Specifies which backfill algorithm will be used. See Configuring Backfill for more information.
Example	`BACKFILLPOLICY  NONE`

BFCHUNKDURATION	
Format	*[[[DD:]HH:]MM:]SS*
Default	*0* (chunking disabled)
Description	Specifies the duration during which freed resources will be aggregated for use by larger jobs. Used in conjunction with **BFCHUNKSIZE** on page 911. See Configuring Backfill for more information.
Example	`BFCHUNKDURATION 00:05:00` `BFCHUNKSIZE     4` *Aggregate backfillable resources for up to 5 minutes, making resources available only to jobs of size 4 or larger.*

BFCHUNKSIZE

Format	*<INTEGER>*
Default	*0* (chunking disabled)
Description	Specifies the minimum job size which can utilize chunked resources. Used in conjunction with **BFCHUNKDURATION** on page 910. See Configuring Backfill for more information.
Example	`BFCHUNKDURATION 00:05:00` `BFCHUNKSIZE      4` *Aggregate backfillable resources for up to 5 minutes, making resources available only to jobs of size 4 or larger.*

BFMINVIRTUALWALLTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	---
Description	Specifies the minimum job wallclock time for virtual scaling (optimistic-like backfilling.) Any job with a wallclock time less than this setting will *not* be virtually scaled. The value specified relates to a job's original walltime and not its virtually-scaled walltime.
Example	`BFMINVIRTUALWALLTIME  00:01:30`

BFPRIORITYPOLICY

Format	One of *RANDOM*, *DURATION*, or *HWDURATION*
Default	---
Description	Specifies policy to use when prioritizing backfill jobs for preemption
Example	`BFPRIORITYPOLICY  DURATION` *Use length of job in determining which backfill job to preempt.*

BFVIRTUALWALLTIMECONFLICTPOLICY	
Format	One of the following: *PREEMPT*
Default	---
Description	Specifies how to handle scheduling conflicts when a virtually scaled job "expands" to its original wallclock time. This occurs when the job is within one scheduling iteration - **RMPOLLINTERVAL** on page 1015 - of its virtually scaled wallclock time expiring.
Example	`BFVIRTUALWALLTIMECONFLICTPOLICY  PREEMPT`

BFVIRTUALWALLTIMESCALINGFACTOR	
Format	*<DOUBLE>*
Default	*0* (virtual scaling disabled)
Description	Specifies the factor by which eligible jobs' wallclock time is virtually scaled (optimistic-like backfilling).
Example	`BFVIRTUALWALLTIMESCALINGFACTOR .4`

BYPASSCAP	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the max weighted value allowed from the bypass count subfactor when determining a job's priority (see Priority Factors for more information).
Example	`BYPASSWEIGHT 5000` `BYPASSCAP    30000`

BYPASSWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight to be applied to a job's backfill bypass count when determining a job's priority (see Priority Factors for more information).
Example	`BYPASSWEIGHT 5000`

CHECKPOINTDIR	
Format	*<STRING>*
Default	---
Description	Specifies the directory for temporary job checkpoint files (usually of the form `jobid.cp`). This is *not* the directory for Moab's checkpoint file (`.moab.ck`).
Example	`CHECKPOINTDIR   /tmp/moabcheckpoint`

CHECKPOINTEXPIRATIONTIME	
Format	*[[[DD:]HH:]MM:]SS* or *UNLIMITED*
Default	*3,000,000 seconds*
Description	Specifies how 'stale' checkpoint data can be before it is ignored and purged.
Example	`CHECKPOINTEXPIRATIONTIME 1:00:00:00` *Expire checkpoint data which has been stale for over 1 day.*

CHECKPOINTFILE	
Format	*<STRING>*
Default	*moab.ck*
Description	Name (absolute or relative) of the Moab checkpoint file.
Example	`CHECKPOINTFILE /var/adm/moab/moab.ck` *Maintain the Moab checkpoint file in the file specified.*

CHECKPOINTINTERVAL	
Format	*[[[DD:]HH:]MM:]SS*
Default	*00:05:00*
Description	Time between automatic Moab checkpoints. If **RMPOLLINTERVAL** on page 1015 does not specify both a minimum and maximum poll time, Moab will ignore **CHECKPOINTINTERVAL** and checkpoint every iteration.
Example	`CHECKPOINTINTERVAL 00:15:00` *Moab should checkpoint state information every 15 minutes.*

CHECKPOINTWITHDATABASE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, Moab stores checkpoint information to a database rather than to the `.moab.ck` flat text file.
Example	`CHECKPOINTWITHDATABASE    TRUE`

CHECKSUSPENDEDJOBPRIORITY	
Format	*<BOOLEAN>*
Default	*TRUE*
Description	Prevents Moab from starting a job on any node containing a suspended job of higher priority.
Example	`CHECKSUSPENDEDJOBPRIORITY     FALSE`

CHILDSTDERRCHECK	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, child processes Moab executes are considered failed if their standard error stream contains the text "ERROR".
Example	`CHILDSTDERRCHECK     TRUE`

CLASSCFG[<CLASSID>]	
Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: General Credential Flags, DEFAULT.ATTR, DEFAULT.DISK, DEFAULT.FEATURES, DEFAULT.GRES, DEFAULT.MEM, DEFAULT.NODE, DEFAULT.NODESET, DEFAULT.PROC, ENABLEPROFILING, EXCL.FEATURES, EXCLUDEUSERLIST, HOSTLIST, JOBEPILOG, JOBPROLOG, MAXPROCPERNODE, MAX.NODE, MAX.PROC, MAX.WCLIMIT, MIN.NODE, MIN.PROC, MIN.TPN, MIN.WCLIMIT, PARTITION, PRIORITY, PRIORITYF, QDEF, QLIST, REQ.FEATURES, REQUIREDACCOUNTLIST, REQUIREDUSERLIST, RESFAILPOLICY, SYSPRIO, WCOVERRUN, usage limit, or fairshare usage limit specification.
Default	---
Description	Specifies class specific attributes (see Credential Overview for details).
Example	`CLASSCFG[batch] MAXJOB=50 QDEF=highprio` *Up to 50 jobs submitted to the class batch will be allowed to execute simultaneously and will be assigned the QOS highprio by default.*

CLASSWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the weight to be applied to the class priority of each job (see Credential (CRED) Factor and credential priority).
Example	`CLASSWEIGHT 10`

CLIENTCFG[<X>]

Format	One or more of *<ATTR>*-*<VALUE>* pairs where *<X>* indicates the specified peer and *<ATTR>* is one of the following: **AUTH**, **AUTHCMD**, **AUTHTYPE**, **HOST**, **KEY**, or **DEFAULTSUBMITPARTITION**.
Default	---
Description	Specifies the shared secret key and authentication method which Moab will use to communicate with the named peer daemon. See Security Overview for more information. **Note**: The **AUTHTYPE** and **KEY** attributes of this parameter may only be specified in the `moab-private.cfg` config file.
Example	`CLIENTCFG[silverB] KEY=apple7 AUTH=admin1` *Moab will use the session key apple7 for peer authentication and for encrypting and decrypting messages sent from silverB. Also, client connections from this interface will be authorized at an admin 1 level.*

CLIENTMAXCONNECTIONS

Format	*<INTEGER>*
Default	*128*
Description	Changes the maximum number of connections that can simultaneously connect to Moab. The value can be increased during runtime, but it cannot be decreased. The value cannot be reduced below the default value of *128*.
Example	`CLIENTMAXCONNECTIONS 256` *Doubles the maximum number of connections.*

CLIENTMAXPRIMARYRETRY	
Format	*<INTEGER>* or *INFINITY*
Default	*1*
Description	Specifies how many times the client command will attempt to retry its connection to the primary server if Moab is not available.
Example	`CLIENTMAXPRIMARYRETRY 5` `CLIENTMAXPRIMARYRETRYTIMEOUT 1000` *The client command will attempt to retry its connection to the primary server 5 times with 1 second intervals before giving up.* ***Note****: If INFINITY is specified, Moab will attempt 2,140,000,000 times.*

CLIENTMAXPRIMARYRETRYTIMEOUT	
Format	*<INTEGER>* (milliseconds)
Default	*2000*
Description	Specifies how much time to wait until the client command will attempt to retry its connection to the primary server if Moab is not available.
Example	`CLIENTMAXPRIMARYRETRY 3` `CLIENTMAXPRIMARYRETRYTIMEOUT 500` *The client command will attempt to retry its connection to the primary server 3 times with .5 second intervals before giving up.*

CLIENTTIMEOUT	
Format	*[[[DD:]HH:]MM:]SS*
Default	*00:00:30*
Description	Time which Moab client commands will wait for a response from the Moab server. See Client Configuration for more information. **Note**: May also be specified as an environment variable.
Example	`CLIENTTIMEOUT 00:15:00` *Moab clients will wait up to 15 minutes for a response from the server before timing out.*

CREDDISCOVERY	
Format	*TRUE*
Default	*FALSE*
Description	Specifies that Moab should create otherwise unknown credentials when it discovers them in the statistics files.
Example	`CREDDISCOVERY TRUE`

CREDWEIGHT	
Format	*<INTEGER>*
Default	*1*
Description	Specifies the credential component weight associated with the credential priority. See Credential (CRED) Factor for more information.
Example	`CREDWEIGHT 2`

DATASTAGEHOLDTYPE	
Format	Any Job Hold type
Default	*DEFER*
Description	Specifies what to do if a job's data staging operations fail.
Example	`DATASTAGEHOLDTYPE  BATCH`

DEADLINEPOLICY	
Format	One of *CANCEL*, *HOLD*, *IGNORE*, or *RETRY*
Default	*NONE*
Description	Specifies what to do when a requested deadline cannot be reached (see Job Deadlines).
Example	`DEADLINEPOLICY  IGNORE`

DEFAULTCLASSLIST	
Format	Space-delimited list of one or more <*STRING*>s.
Default	---
Description	Specifies the default classes supported on each node for RM systems which do not provide this information.
Example	`DEFAULTCLASSLIST serial parallel`

DEFAULTSUBMITPARTITION	
Format	See parameter CLIENTCFG[].
Default	---
Description	If a user submits a job using msub which does not specify host, feature, or partition constraints, then the msub client will insert the specified default submit partition into the newly submitted job as a hard requirement.
Example	`CLIENTCFG[DEFAULT] DEFAULTSUBMITPARTITION=partition1`

DEFERCOUNT	
Format	*<INTEGER>*
Default	*24*
Description	Specifies the number of times a job can be deferred before it will be placed in batch hold.
Example	`DEFERCOUNT 12`

DEFERSTARTCOUNT	
Format	*<INTEGER>*
Default	*1*
Description	Specifies the number of times a job will be allowed to fail in its start attempts before being deferred. **JOBRETRYTIME** overrides **DEFERSTARTCOUNT**; **DEFERSTARTCOUNT** only begins when the **JOBRETRYTIME** window elapses. **Note**: A job's startcount will increase each time a start request is made to the resource manager regardless of whether or not this request succeeded. This means start count increases if job starts fail or if jobs are started and then rejected by the resource manager. (For related information, see Reservation Policies, DEFERTIME, RESERVATIONRETRYTIME, NODEFAILURERESERVETIME, JOBRETRYTIME, and GUARANTEEDPREEMPTION.)
Example	`DEFERSTARTCOUNT 3`

DEFERTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*1:00:00*
Description	Specifies the amount of time a job will be held in the deferred state before being released back to the Idle job queue. **Note**: A job's defer time will be restarted if Moab is restarted. (For related information, see Reservation Policies, DEFERSTARTCOUNT, RESERVATIONRETRYTIME, NODEFAILURERESERVETIME, JOBRETRYTIME, and GUARANTEEDPREEMPTION.)
Example	`DEFERTIME 0:05:00`

DELETESTAGEOUTFILES	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether the scheduler should delete explicitly specified stageout files after they are successfully staged. By default, such files are not deleted but are left on the nodes where the job ran.
Example	`DELETESTAGEOUTFILES TRUE` `Example of an explicit stageout request` `msub x=MSTAGEOUT:ssh://source_node/tmp/file,file:///results_folder` `job.cmd` *With this parameter set to TRUE, `/tmp/file` on source_node is deleted after it is copied to the specified destination (`file:///results_folder`). If the parameter is not set, or if it is set to FALSE, `/tmp/file` remains on source_node after the job terminates.*

DEPENDFAILUREPOLICY	
Format	*HOLD* or *CANCEL*
Default	*HOLD*
Description	Specifies what happens to a job if its dependencies cannot be fulfilled; that is, what happens when a job depends on another job to complete successfully but the other job fails.
Example	`DEPENDFAILUREPOLICY CANCEL` *If job A is submitted with depend=afterok:B and job B fails, job A is canceled.*

DIRECTORYSERVER	
Format	*<HOST>[:<PORT>]*
Default	---
Description	Specifies the interface for the directory server.
Example	`DIRECTORYSERVER calli3.icluster.org:4702`

DISABLEEXCHLIST	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If the resource manager rejects a job and the value is set to *TRUE*, then the node is not added to the job's exclude host list.
Example	`DISABLEEXCHLIST TRUE`

DISABLEINTERACTIVEJOBS	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Disallows interactive jobs submitted with msub -I. **Note**: It is possible for users to submit interactive jobs directly to a resource manager, which can bypass the **DISABLEINTERACTIVEJOBS** parameter. However, some resource managers (such as TORQUE) will check with Moab before allowing an interactive job.
Example	`DISABLEINTERACTIVEJOBS TRUE`

DISABLEREGEXCACHING	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Turns off regular expression caching. Turning off regular expression caching preserves memory with hostlist reservations and speeds up start time.
Example	`DISABLEREGEXCACHING TRUE`

DISABLEREQUIREDGRESNONE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	When set to *TRUE*, this causes Moab to reject msub requests that have a gres of "none". ENFORCEGRESSACCESSS must also be set to TRUE for this feature to work.
Example	(see below)

```
##########  moab.cfg  ##########
ENFORCEGRESSACCESS TRUE
DISABLEREQUIREDGRESNONE TRUE
################################

> msub -A ee -l nodes=1,ttc=5,walltime=600,partition=g02 -l gres=none
ERROR: cannot submit job - cannot locate required resource 'none'
```

DISABLESAMECREDPREEMPTION

Format	Comma-delimited list of one or more credentials: *ACCT*, *CLASS*, *GROUP*, *QOS*, or *USER*
Default	---
Description	This parameter prevents specified credentials from preempting its own jobs.
Example	`DISABLESAMECREDPREEMPTION  QOS,USER`

DISABLESCHEDULING

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether or not the scheduler will schedule jobs. If set to *TRUE*, Moab will continue to update node and job state but will not start, preempt, or otherwise modify jobs. The command mschedctl -r will clear this parameter and resume normal scheduling.
Example	`DISABLESCHEDULING FALSE`

DISABLESLAVEJOBSUBMIT

Format	*<BOOLEAN>*
Default	*TRUE*
Description	This parameter can be added to the `moab.cfg` file on a slave Moab server (in a grid configuration) to prevent users from submitting jobs to the master Moab server from the slave Moab server. Some grid configurations allow the user to submit jobs on the slave that are migrated to the master and submitted from the master. Other grid configurations do not allow the jobs to be migrated to the master from the slave, in which case, jobs submitted from the slave remain idle on the slave and never run. This parameter will reject the job submissions on the slave to prevent the submission of jobs that will never run.
Example	`DISABLESLAVEJOBSUBMIT TRUE` `example (node04 is a slave and node06 is the master)` `[test@node04 moab-slurm]$ echo sleep 100 | msub` `ERROR:    cannot submit job from slave`

DISABLETHRESHOLDTRIGGERS

Format	*<BOOLEAN>*
Default	*FALSE*
Description	This makes Moab not fire threshold-based triggers, but will log the intended action to the event logs. Similar to DISABLEVMDECISIONS.
Example	`DISABLETHRESHOLDTRIGGERS TRUE`

DISABLEVMDECISIONS

Format	*<BOOLEAN>*
Default	*FALSE*
Description	This makes Moab not take any automatic decisions with respect to VM's, namely powering on/off nodes and migrating VMs. Intended actions will instead be logged in the event logs. Similar to DISABLETHRESHOLDTRIGGERS.
Example	`DISABLEVMDECISIONS TRUE`

DISKWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight to be applied to the amount of dedicated disk space required per task by a job (in MB).
Example	`RESWEIGHT  10` `DISKWEIGHT 100` *If a job requires 12 tasks and 512 MB per task of dedicated local disk space, Moab will increase the job's priority by 10 * 100 * 12 * 512*

DISPLAYFLAGS	
Format	One or more of the following values (space delimited): *ACCOUNTCENTRIC*, *HIDEBLOCKED*, *HIDEDRAINED*, *NODECENTRIC*, or *USEBLOCKING*
Default	---
Description	Specifies flags that control how Moab client commands display varied information. *ACCOUNTCENTRIC* will display account information in showq, rather than group information. *HIDEBLOCKED* will prevent showq from listing information about blocked jobs which are not owned by the user if the user is not an admin. *HIDEDRAINED* prevents mdiag -n from displaying nodes and mvmctl -q from displaying VMs in the DRAINED state. An override option of mdiag -n -w nodestate=drained lists only those nodes with a DRAINED state. Similarly, an override option of mvmctl -q -w state=drained lists only those VMs with a DRAINED state. *NODECENTRIC* will display node allocation information instead of processor allocation information in showq. *USEBLOCKING* disables threading for commands that support it; those commands include showq, mdiag -n, and mdiag -j.
Example	`DISPLAYFLAGS NODECENTRIC`

DISPLAYPROXYUSERASUSER	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, Moab shows the proxy users instead of the real user on some queries of system jobs that have proxy users. Commands affected include mjobctl -q diag and checkjob.
Example	`DISPLAYPROXYUSERASUSER TRUE`

DONTCANCELINTERACTIVEHJOBS	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, Moab does not cancel interactive jobs that are held.
Example	`DONTCANCELINTERACTIVEHJOBS TRUE`

DONTENFORCEPEERJOBLIMITS	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, only the scheduler that is running the job can cancel the job or enforce other limits.
Example	`DONTENFORCEPEERJOBLIMITS TRUE`

EMULATIONMODE	
Format	*SLURM*
Default	---
Description	Specifies whether or not the scheduler will perform the automatic setup of a particular resource manager environment.
Example	`EMULATIONMODE SLURM` *Moab will perform the automated setup steps as if it were interfacing with a slurm resource manager (automatic QOS creation).*

ENABLEFAILUREFORPURGEDJOB	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	By default, when a job is purged or removed by the TORQUE resource manager for a walltime violation, the job takes on a state of Completed and a completion code of 0. If *TRUE*, the job state is set to Removed and has a completion code of 98. **ENABLEFAILUREFORPURGEDJOB** is for the TORQUE resource manager only. For **ENABLEFAILUREFORPURGEDJOB** to return Removed job states, you must reset the TORQUE server attribute `keep_completed` to `0` in `qmgr`. See "**Queue Attributes** on page 2277" in the TORQUE Administrator Guide for more information.
Example	`ENABLEFAILUREFORPURGEDJOB TRUE` *Jobs that are purged or removed by TORQUE are given a state of Removed and a completion code of 98.*

ENABLEFSVIOLATIONPREEMPTION	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, Moab will allow jobs within the same class/queue to preempt when the preemptee is violating a fairshare target and the preemptor is not.
Example	`ENABLEFSVIOLATIONPREEMPTION TRUE`

ENABLEHIGHTHROUGHPUT	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Configures Moab so that it will accept msub submissions, start jobs, process triggers, etc., in a manner which minimizes their processing time. The downside is that Moab will return minimal information about these jobs at submit time. If **ENABLEHIGHTHROUGHPUT** is *TRUE*, you must set **NODEALLOCATIONPOLICY** on page 978 to *FIRSTAVAILABLE*.
Example	`ENABLEHIGHTHROUGHPUT TRUE` *Moab can now accept hundreds of jobs per second using msub instead of 20-30.*

ENABLEJOBARRAYS	
Format	*<BOOLEAN>*
Default	*TRUE*
Description	If set to *TRUE*, job arrays will be enabled .
Example	`ENABLEJOBARRAYS TRUE`

ENABLENEGJOBPRIORITY	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, the scheduler allows job priority value to range from -INFINITY to MMAX_PRIO; otherwise, job priority values are given a lower bound of '1'. For more information, see REJECTNEGPRIOJOBS.
Example	`ENABLENEGJOBPRIORITY TRUE` *Job priority may range from -INFINITY to MMAX_PRIO.*

ENABLENODEADDRLOOKUP	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, the scheduler will use the default host name service lookup mechanism (i.e., `/etc/hosts`, DNS, NIS, etc.) to determine the IP address of the nodes reported by the resource manager. This information is used to correlate information reported by multi-homed hosts.
Example	`ENABLENODEADDRLOOKUP TRUE`

ENABLEPOSUSERPRIORITY	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, the scheduler will allow users to specify positive job priority values which will be honored. In other words, users can specify a priority that falls in the range of -1024 to +1023, inclusive. If set to *FALSE* (the default), user priority values are given an upper bound of '0' when users request a positive priority. See USERPRIOWEIGHT.
Example	`ENABLEPOSUSERPRIORITY TRUE` *Users may now specify positive job priorities and have them take effect (e.g.* `msub -p 100 job.cmd`*).*

ENABLESPVIOLATIONPREEMPTION	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, Moab will allow jobs within the same class/queue to preempt when the preemptee is violating a *soft* usage policy and the preemptor is not.
Example	`ENABLESPVIOLATIONPREEMPTION TRUE`

ENABLEVMDESTROY	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, enables the automatic destruction of a VM when the VM wall time is expired or when the VM is stale and configured to be destroyed (for more information, see VMSTALEACTION).
Example	`ENABLEVMDESTROY TRUE`

ENFORCEACCOUNTACCESS	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether or not Moab will enforce account access constraints without an allocation manager.
Example	`ENFORCEACCOUNTACCESS TRUE`

ENFORCEGRESACCESS	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If a user submits a job with a non-existent gres (e.g. in the case of a typo) and **ENFORCEGREACCESS** is not set in `moab.cfg`, or is set to *FALSE*, then the requested gres will be created (but will not exist on any nodes) and the job will be deferred (similar to **ENFORCEACCOUNTACCESS**).
Example	`ENFORCEGRESACCESS TRUE`

EVENTSERVER	
Format	*<HOST>[:<PORT>]*
Default	---
Description	Specifies the interface for the event server.
Example	`EVENTSERVER calli3.icluster.org:4702`

FEATURENODETYPEHEADER	
Format	*<STRING>*
Default	---
Description	Specifies the header used to specify node type via node features (i.e. LL features or PBS node attributes).
Example	`FEATURENODETYPEHEADER xnt` *Moab will interpret all node features with the leading string* xnt *as a nodetype specification - as used by the allocation manager and other allocation managers, and assign the associated value to the node. i.e., xntFast.*

FEATUREPARTITIONHEADER	
Format	*<STRING>*
Default	---
Description	Specifies the header used to specify node partition via node features (i.e. LL features or PBS node attributes).
Example	`FEATUREPARTITIONHEADER xpt` *Moab will interpret all node features with the leading string* xpt *as a partition specification and assign the associated value to the node. i.e., xptGold.*

FEATUREPROCSPEEDHEADER

Format	*<STRING>*
Default	---
Description	Specifies the header used to extract node processor speed via node features (i.e., LL features or PBS node attributes). **Note**: Adding a trailing '$' character will specify that only features with a trailing number be interpreted. For example, the header 'sp$' will match 'sp450' but not 'sport'.
Example	`FEATUREPROCSPEEDHEADER xps` *Moab will interpret all node features with the leading string xps as a processor speed specification and assign the associated value to the node. i.e., xps950.*

FEATURERACKHEADER

Format	*<STRING>*
Default	---
Description	Specifies the header used to extract node rack index via node features (i.e., LL features or PBS node attributes). **Note**: Adding a trailing '$' character will specify that only features with a trailing number be interpreted. For example, the header 'rack$' will match 'rack4' but not 'racket'.
Example	`FEATURERACKHEADER rack` *Moab will interpret all node features with the leading string rack as a rack index specification and assign the associated value to the node. i.e., rack16.*

FEATURESLOTHEADER	
Format	*<STRING>*
Default	---
Description	Specifies the header used to extract node slot index via node features (i.e., LL features or PBS node attributes). **Note**: Adding a trailing '$' character will specify that only features with a trailing number be interpreted. For example, the header 'slot$' will match 'slot12' but not 'slotted'.
Example	`FEATURESLOTHEADER slot` *Moab will interpret all node features with the leading string* slot *as a slot index specification and assign the associated value to the node. i.e., slot16.*

FEEDBACKPROGRAM	
Format	*<STRING>*
Default	---
Description	Specifies the name of the program to be run at the completion of each job. If not fully qualified, Moab will attempt to locate this program in the 'tools' subdirectory.
Example	`FEEDBACKPROGRAM /var/moab/fb.pl` *Moab will run the specified program at the completion of each job.*

FILEREQUESTISJOBCENTRIC	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether a job's file request is a total request for the job or a per task request.
Example	`FILEREQUESTISJOBCENTRIC TRUE` *Moab will treat file requests as a total request per job.*

FILTERCMDFILE	
Format	*<BOOLEAN>*
Default	*TRUE*
Description	Running the msub command performs the following operations on the submission script: • Replace all comments with spaces (excludes Resource Manager directives) • Strip empty lines • Replace `\r` with `\n` • Lock job to a PBS resource manager if `$PBS` is found in the script Include the **FILTERCMDFILE** parameter in the `moab.cfg` file that resides on the clients. **FILTERCMDFILE** must be *FALSE* for **REJECTDOSSCRIPTS** on page 1005 to work correctly.
Example	`FILTERCMDFILE FALSE` *Running the msub command does not perform the actions detailed earlier.*

FORCENODEREPROVISION	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	When set to *TRUE*, this config option causes Moab to reprovision a node, even if it is to the same operating system (in essence rewriting the OS).
Example	`FORCENODEREPROVISION TRUE`

FORCERSVSUBTYPE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies that admin reservations must have a subtype associated with them.
Example	`FORCERSVSUBTYPE TRUE` *Moab will require all admin reservations to include a subtype.*

FREETIMELOOKAHEADDURATION	
Format	*[[[DD:]HH:]MM:]SS*
Default	*2 Months*
Description	Specifies how far ahead Moab will look when calculating free time on a node.
Example	`FREETIMELOOKAHEADDURATION 7:00:00:00` *Moab will look 1 week ahead when it calculates free time on a node.*

FSACCOUNTWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to the account subcomponent of the fairshare component of priority.
Example	`FSACCOUNTWEIGHT 10`

FSCAP	
Format	*<DOUBLE>*
Default	*0* (NO CAP)
Description	Specifies the maximum allowed absolute value for a job's total pre-weighted fairshare component.
Example	`FSCAP 10.0` *Moab will bind a job's pre-weighted fairshare component by the range +/- 10.0.*

FSCLASSWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to the class subcomponent of the fairshare component of priority.
Example	`FSCLASSWEIGHT 10`

FSDECAY	
Format	*<DOUBLE>*
Default	*1.0*
Description	Specifies decay rate applied to past fairshare interval when computing effective fairshare usage. Values may be in the range of 0.01 to 1.0. A smaller value causes more rapid decay causing *aged* usage to contribute less to the overall effective fairshare usage. A value of 1.0 indicates that no decay will occur and all fairshare intervals will be weighted equally when determining effective fairshare usage. See Fairshare Overview.
Example	`FSPOLICY   DEDICATEDPS` `FSDECAY    0.8` `FSDEPTH    8` *Moab will apply a decay rate of 0.8 to all fairshare windows.*

FSDEPTH	
Format	*<INTEGER>*
Default	*8*
Description	**Note**: The number of available fairshare windows is bounded by the MAX_FSDEPTH value (32 in Moab). See Fairshare Overview.
Example	`FSDEPTH 12`

FSENABLECAPPRIORITY	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Fairshare priority will increase to target and stop.
Example	`FSENABLECAPPRIORITY TRUE`

FSGROUPWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	
Example	`FSGROUPWEIGHT 4`

FSINTERVAL	
Format	*[[[DD:]HH:]MM:]SS*
Default	*12:00:00*
Description	Specifies the length of each fairshare window.
Example	`FSINTERVAL 12:00:00` *Track fairshare usage in 12 hour blocks.*

FSJPUWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the fairshare weight assigned to jobs per user.
Example	`FSJPUWEIGHT 10`

FSMOSTSPECIFICLIMIT	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	When checking policy usage limits in a fairshare tree, if the most specific policy limit is passed then do not check the same policy again at higher levels in the tree. For example, if a user has a MaxProc policy limit then do not check the MaxProc policy limit on the account for this same user.
Example	`FSMOSTSPECIFICLIMIT  TRUE`

FSPOLICY

Format	*<POLICY>[*]* where *<POLICY>* is one of the following: *DEDICATEDPS*, *DEDICATEDPS%*, *DEDICATEDPES*, or *UTILIZEDPS*.
Default	---
Description	Specifies the unit of tracking fairshare usage. The following options are: • *DEDICATEDPS* (dedicated processor seconds delivered) tracks dedicated processor seconds. • *DEDICATEDPS%* (dedicated processor seconds available) to specify that percentage based fairshare should be used. See Fairshare Overview and Fairshare Consumption Metrics for more information. • *DEDICATEDPES* (dedicated processor-equivalent seconds delivered) tracks dedicated processor-equivalent seconds. • *UTILIZEDPS* (utilized processor seconds delivered) tracks the number of utilized processor seconds.
Example	`FSPOLICY DEDICATEDPES` *Moab will track fairshare usage by dedicated processor-equivalent seconds.*

FSPPUWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the fairshare weight assigned to processors per user.
Example	`FSPPUWEIGHT 10`

FSPSPUWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the fairshare weight assigned to processor-seconds per user.
Example	`FSPSPUWEIGHT 10`

FSQOSWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight assigned to the QOS fairshare subcomponent.
Example	`FSQOSWEIGHT 16`

FSTARGETISABSOLUTE

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether Moab should base fairshare targets off of delivered cycles or up/available cycles.
Example	`FSTARGETISABSOLUTE TRUE`

FSTREE

Format	List of zero or more space delimited *<ATTR>*=*<VALUE>* pairs where *<ATTR>* is one of the following: SHARES or MEMBERLIST
Default	---
Description	Specifies the share tree distribution for job fairshare prioritization (see Hierarchical Share Tree Overview).
Example	`FSTREE[geo]  SHARES=16  MEMBERLIST=geo103,geo313,geo422`

FSTREEACLPOLICY

Format	*OFF*, *PARENT*, or *FULL*
Default	*FULL*
Description	Specifies how Moab should interpret credential membership when building the FSTREE (see Hierarchical Share Tree Overview).
Example	`FSTREEACLPOLICY PARENT` *Credentials will be given access to their parent node when applicable.*

FSTREEISREQUIRED

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether a job must have an applicable node in the partition's FSTREE in order to execute within that partition (see Hierarchical Share Tree Overview).
Example	`FSTREEISREQUIRED TRUE` *Jobs must have an applicable node in the FSTREE in order to execute.*

FSTREEUSERISREQUIRED	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether the user must be given explicit access to a branch in the FSTREE (see Hierarchical Share Tree Overview).
Example	`FSTREEUSERISREQUIRED TRUE` *Users must be given explicit access to FSTREE nodes in order to gain access to the FSTREE.*

FSUSERWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight assigned to the user fairshare subfactor.
Example	`FSUSERWEIGHT 8`

FSWEIGHT	
Format	*<INTEGER>*
Default	*1*
Description	Specifies the priority weight assigned to the summation of the fairshare subfactors (see Priority Factor and Fairshare overviews).
Example	`FSWEIGHT 500`

GEVENTCFG[<GEVENT>]

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is **ACTION**, **ECOUNT**, **REARM**, or **SEVERITY**. See Responding to Generic Events for details on values you can assign to each attribute.
Default	---
Description	Specifies how the scheduler should behave when various cluster events are detected. See the Generic Events Overview for more information.
Example	`GEVENTCFG[hitemp] ACTION=avoid,record,notify  REARM=00:10:00` `GEVENT[nodeerror] SEVERITY=3` *If a hitemp event is detected, Moab adjusts the node allocation policy to minimize the allocation of the node. Moab also sends emails to cluster administrators and reports the event in the Moab event log.*

GRESCFG[<GRES>]

Format	List of zero or more space delimited *<ATTR >=<VALUE>* pairs where *<ATTR>* is one of the following: **TYPE**, **PRIVATE**, **INVERTTASKCOUNT**, FEATUREGRES, or STARTDELAY
Default	---
Description	Specifies associations of generic resources into resource groups. When **PRIVATE** is set to *TRUE*, Moab puts the requested generic resource on a separate job request. By default a private request is a request with 1 task with *X* number of generic resources per task. When **INVERTTASKCOUNT***and***PRIVATE** are set to *TRUE*, Moab makes the generic resource's private request a request with *X* number of tasks with 1 generic resource per task. See 12.6 Managing Consumable Generic Resources for more information.
Example	`GRESCFG[scsi1] TYPE=fastio` `GRESCFG[scsi2] TYPE=fastio` `GRESCFG[scsi3] TYPE=fastio` *The generic resources scsi1, scsi2, and scsi3 are all associated with the generic resource type* fastio.

GRESTOJOBATTR

Format	Comma delimited list of generic resources
Default	---
Description	The list of generic resources will also be interpreted as JOB features. See Managing Reservations.
Example	`GRESTOJOBATTR  matlab,ccs` *Jobs which request the generic resources matlab or ccs will have a corresponding job attribute assigned to them.*

GROUPCFG[<GROUPID>]

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: General Credential Flags, **PRIORITY**, ENABLEPROFILING, **QLIST**, **QDEF**, **PLIST**, **FLAGS**, usage limits, or a fairshare usage limit specification.
Default	---
Description	Specifies group specific attributes. See the flag overview for a description of legal flag values.
Example	`GROUPCFG[staff] MAXJOB=50 QDEF=highprio` *Up to 50 jobs submitted by members of the group staff will be allowed to execute simultaneously and will be assigned the QOS highprio by default.*

GROUPWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the priority weight assigned to the specified group priority (See Credential (CRED) Factor).
Example	`GROUPWEIGHT 20`

GUARANTEEDPREEMPTION	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Causes Moab to lock PREEMPTOR jobs until JOBRETRYTIME expires (essentially, waiting for the PREEMPTEE jobs to finish). It may take some time for the PREEMPTEE jobs to clear out. During that time, the PREEMPTOR job might want to look elsewhere to run, which would be disruptive as it might preempt another set of jobs. If you wish it prevent this, it is recommended that you set **GUARANTEEDPREEMPTION** to **TRUE**. For related information, see About preemption, Reservation Policies, DEFERSTARTCOUNT, DEFERTIME, RESERVATIONRETRYTIME, NODEFAILURERESERVETIME, and JOBRETRYTIME.
Example	`GUARANTEEDPREEMPTION TRUE`

HALOCKCHECKTIME	
Format	[[[DD:]HH:]MM:]SS
Default	9
Description	Specifies how frequently the secondary server checks the timestamp on the lock file. See High Availability Overview for more info.
Example	`HALOCKCHECKTIME 00:00:15` *The Moab fallback server will check the health of the Moab primary server every 15 seconds.*

HALOCKUPDATETIME	
Format	[[[DD:]HH:]MM:]SS
Default	3
Description	Specifies how frequently the primary server checks the timestamp on the lock file. See High Availability Overview for more info.
Example	`HALOCKUPDATETIME 00:00:03` *The Moab primary server will check the timestamp of the lock file every 3 seconds.*

HIDEVIRTUALNODES	
Format	*<BOOLEAN>*
Default	---
Description	Enables VM management; also used to reveal hypervisors.
Example	`HIDEVIRTUALNODES TRANSPARENT`

IDCFG[X]	
Format	One or more of the following attribute/value pairs: *BLOCKEDCREDLIST*, CREATECRED, CREATECREDURL, *REFRESHPERIOD*, *RESETCREDLIST* or *SERVER*.
Default	---
Description	This parameter enables the identity manager interface allowing credential, policy, and usage information to be shared with an external information service. Only one identity manager can be configured at a time.
Example	`IDCFG[info] SERVER=exec://dbquery.pl REFRESHPERIOD=hour` *Moab will refresh credential info every hour using the specified script.*

IGNOREMDATASTAGING	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	When set to *TRUE*, Moab will ignore any resource manager specific data staging on a job and assume the resource manager is processing the request. Currently, this only applies to PBS.
Example	`IGNORERMDATASTAGING TRUE`

IGNORECLASSES	
Format	*[!]<CLASS>[,<CLASS>]...*
Default	---
Description	By default, if using the TORQUE resource manager, jobs from all listed classes are ignored and not scheduled, tracked, or otherwise processed by Moab. If the **not**(i.e., '!') character is specified, only jobs from listed classes are processed. See the Moab Side-by-Side Analysis for more information.
Example	`IGNORECLASSES dque,batch` *Moab will ignore jobs from classes dque and batch.*

IGNOREJOBS	
Format	*[!]<JOBID>[,<JOBID>]...*
Default	---
Description	By default, listed jobs are ignored and not scheduled, tracked, or otherwise processed by Moab. If the **not**(i.e., '!') character is specified, only listed jobs are processed. See the Moab Side-by-Side Analysis for more information.
Example	`IGNOREJOBS !14221,14223` *Moab will ignore jobs all classes except 14221 and 14223.*

IGNORENODES

Format	*[!]<NODE>[,<NODE>]...*
Default	---
Description	By default, all listed nodes are ignored and not scheduled, tracked, or otherwise processed by Moab. If the **not**(i.e., '!') character is specified, only listed nodes are processed. See the Moab Side-by-Side Analysis for more information.
Example	`IGNORENODES !host3,host4` *Moab will only process nodes host3 and host4.*

IGNOREPREEMPTEEPRIORITY

Format	*<BOOLEAN>*
Default	*FALSE*
Description	By default, preemptor jobs can only preempt preemptee jobs if the preemptor has a higher job priority than the preemptee. When this parameter is set to true, the priority constraint is removed allowing any preemptor to preempt any preemptees once it reaches the top of the eligible job queue.
Example	`IGNOREPREEMPTEEPRIORITY TRUE` *A preemptor job can preempt any preemptee jobs when it is at the top of the eligible job queue.*

IGNOREUSERS

Format	*[!]<USERNAME>[,<USERNAME>]...*
Default	---
Description	By default, if using the TORQUE resource manager, jobs from all listed users are ignored and not scheduled, tracked, or otherwise processed by Moab. If the **not**(i.e., '!') character is specified, only jobs from listed users are processed. (See the Moab Side-by-Side Analysis for more information.)
Example	`IGNOREUSERS testuser1,annapolis` *Moab will ignore jobs from users testuser1 and annapolis.*

#INCLUDE

Format	*<STRING>*
Default	---
Description	Specifies another file which contains more configuration parameters. If *<STRING>* is not an absolute path, Moab will search its home directory for the file.
Example	`#INCLUDE moab.acct` *Moab will process the parameters in `moab.acct` as well as `moab.cfg`*

INSTANTSTAGE

Description	This parameter is deprecated. Use JOBMIGRATEPOLICY.

INVALIDFSTREEMSG	
Format	"*<STRING>*"
Default	*"no valid fstree node found"*
Description	Specifies the error message that should be attached to jobs that cannot run because of a fairshare tree configuration violation.
Example	`INVALIDFSTREEMSG       "account is invalid for requested partition"`

JOBACTIONONNODEFAILURE	
Format	**CANCEL** on page 413, **FAIL** on page 413, **HOLD** on page 413, **IGNORE** on page 413, **NOTIFY** on page 413, or **REQUEUE** on page 413
Default	---
Description	Specifies the action to take if Moab detects that a node allocated to an active job has failed (state is down). By default, Moab only reports this information via diagnostic commands. If this parameter is set, Moab will cancel or requeue the active job. See Reallocating Resources When Failures Occur for more information. **Note**: The *HOLD* value is only applicable when using checkpointing.
Example	`JOBACTIONONNODEFAILURE REQUEUE` *Moab will requeue active jobs which have allocated nodes which have failed during the execution of the job.*

JOBAGGREGATIONTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*0*
Description	Specifies the minimum amount of time the scheduler should wait after receiving a job event until it should process that event. This parameter allows sites with *bursty* job submissions to process job events in groups decreasing total job scheduling cycles and allowing the scheduler to make more intelligent choices by aggregating job submissions and choosing between the jobs. See Considerations for Large Clusters.
Example	`JOBAGGREGATIONTIME 00:00:04` `RMPOLLINTERVAL     30,30` *Moab will wait 4 seconds between scheduling cycles when job events have been received and will wait 30 seconds between scheduling cycles otherwise.*

JOBCFG

Format	*<ATTR>=<VAL>* where *<ATTR>* is one of FLAGS, GRES, NODERANGE, PRIORITYF, PROCRANGE, QOS, RARCH, RFEATURES, ROPSYS, SELECT, or TARGETLOAD
Default	---
Description	Specifies attributes for jobs which satisfy the specified profile. The **SELECT** attribute allows users to specify the job template by using msub -l template=. The **JOBCFG** parameter supports the following attributes: **NONE**, **ACCOUNT**, **ACTION**, **AUTOSIZE**, **CLASS**, **CPULIMIT**, **DESCRIPTION**, **DGRES**, **FAILUREPOLICY**, **GROUP**, **IFLAGS**, **JOBSCRIPT MEM** (for **MEM**=*<value>*), **MEMORY** (for **MEMORY**=*$LEARN*), **NODEACCESSPOLICY**, **NODEMOD**, **PARTITION**, **PREF**, **QOS**, **RESTARTABLE**, **RM**, **RMSERVICEJOB**, **SELECT**, **SOFTWARE**, **SRM**, **TEMPLIMIT**, **TFLAGS**, **USER**, **VMUSAGE**, **WALLTIME**, **WORK** It also supports the following Wiki attributes: **ARGS**, **DMEM**, **DDISK**, **DWAP**, **ERROR**, **EXEC**, **EXITCODE**, **GATTR**, **GEVENT**, **IWD**, **JNAME**, **NAME**, **PARTITIONMASK**, **PRIORITYF**, **RDISK**, **RSWAP**, **RAGRES**, **RCGRES**, **TASKPERNODE**, **TRIGGER**, **VARIABLE**, **NULL** **Note**: The *index* to the **JOBCFG** parameter can either be an admin-chosen job template name or the exact name of job reported by one or more workload queries. See Wiki Attributes and Job Template Extensions.
Example	`JOBCFG[sql] RFEATURES=sqlnode QOS=service` *When the sql job is detected, it will have the specified default qos and node feature attributes set.*

JOBCPURGETIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*00:05:00*
Description	Specifies the amount of time Moab will preserve detailed information about a completed job (see showq -c and checkjob).
Example	`JOBCPURGETIME 02:00:00` *Moab will maintain detailed job information for 2 hours after a job has completed.*

JOBTRUNCATENLCP	
Format	*<BOOLEAN>*
Default	TRUE
Description	Specifies whether Moab will store only the first node of the node list for a completed job in the checkpoint file.
Example	`JOBTRUNCATENLCP TRUE` **JOBTRUNCATENLCP** *reduces the amount of memory Moab uses to store completed job information.*

JOBEXTENDSTARTWALLTIME	
Format	*<BOOLEAN>*
Default	---
Description	Extends the job walltime when Moab starts the job up to the lesser of the maximum or the next reservation (rounded down to the nearest minute). JOBEXTENDSTARTWALLTIME TRUE and JOBEXTENDDURATION cannot be configured together. If they are in the same moab.cfg or are both active, then the JOBEXTENDDURATION will not be honored.
Example	`JOBEXTENDSTARTWALLTIME TRUE` Submit job with a minimum wallclock limit and a walltime; for example: `echo sleep 500 | msub -A ee -l nodes=5,minwclimit=5:00,walltime=30:00,partition=g02` *At job start, Moab recognizes the nodes assigned to the specified job and extends the walltime for the job (one time at job start) up to the lesser of the maximum walltime requested or the least amount of time available for any of the nodes until the next reservation on that node.*

JOBFAILRETRYCOUNT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the number of times a job is requeued and restarted by Moab if the job fails (if the job itself returns a non-zero exit code). Some types of jobs may succeed if automatically retried several times in short succession. This parameter was created with these types of jobs in mind. Note that the job in question must also be restartable (the job needs to have the "RESTARTABLE" flag set on it) and the RM managing the job must support requeuing and starting completed jobs. If a job fails too many times, and reaches the number of retries given by JobFailRetryCount, then a UserHold is placed on the job and a message is attached to it signifying that the job has a "restart count violation."
Example	`JOBFAILRETRYCOUNT  7` *Any job with a RESTARTABLE flag is requeued, if it fails, up to 7 times before a UserHold is placed on it.*

JOBIDWEIGHT

Format	*<INTEGER>*
Default	---
Description	Specifies the weight to be applied to the job's id. See Attribute (ATTR) Factor.
Example	`JOBIDWEIGHT -1` *Later jobs' priority will be negatively affected.*

JOBMATCHCFG

Format	*<ATTR>=<VAL>* where *<ATTR>* is one of JMIN, JMAX, JDEF, JSET, or JSTAT
Default	---
Description	Specifies the job templates which must be matched and which will be applied in the case of a match.
Example	`JOBMATCHCFG[sql] JMIN=interactive JSTAT=istat`

JOBMAXHOLDTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	---
Description	Specifies the amount of time a job can be held before it is canceled automatically.
Example	`JOBMAXHOLDTIME 02:00:00` *Moab will keep jobs in any HOLD state for 2 hours before canceling them.*

JOBMAXNODECOUNT

Format	*<INTEGER>*
Default	*1024*
Description	Specifies the maximum number of nodes which can be allocated to a job. After changing this parameter, Moab must be restarted. **Note**: This value cannot exceed either **MMAX_NODE** or **MMAX_TASK_PER_JOB**. If larger values are required, these values must also be increased. Moab must be restarted before changes to this command will take effect. The command mdiag -S will indicate if any job node count overflows have occurred. See Consideration for Large Clusters.
Example	`JOBMAXNODECOUNT 4000`

JOBMAXOVERRUN	
Format	*[[[DD:]HH:]MM:]SS,][[[DD:]HH:]MM:]SS*
Default	(no soft limit), *10 minutes* (hard limit)
Description	Soft and hard limit of the amount of time Moab will allow a job to exceed its wallclock limit before it first sends a mail to the primary admin (soft limit) and then terminates the job (hard limit). See WCVIOLATIONACTION or Usage-based Limits. If you run Moab with the TORQUE resource manager, you must set the **$ignwalltime** parameter to *true* in the `/var/spool/torque/mom_priv/config` file, otherwise the pbs_mom will kill any job that exceeds its walltime. See the TORQUE documentation for more information.
Example	`JOBMAXOVERRUN 15:00,1:00:00` *Jobs may exceed their wallclock limit by up to 1 hour, but Moab will send an email to the primary administrator when a job exceeds its walltime by 15 minutes.*

JOBMAXPREEMPTPERITERATION	
Format	*<INTEGER>*
Default	*0* (No Limit)
Description	Maximum number of jobs allowed to be preempted per iteration.
Example	`JOBMAXPREEMPTPERITERATION 10`

JOBMAXSTARTPERITERATION	
Format	*<INTEGER>*
Default	*0* (No Limit)
Description	Maximum number of jobs allowed to start per iteration.
Example	`JOBMAXSTARTPERITERATION 10`

JOBMAXSTARTTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*-1* (NO LIMIT)
Description	length of time a job is allowed to remain in a 'starting' state. If a 'started' job does not transition to a running state within this amount of time, Moab will cancel the job, believing a system failure has occurred.
Example	`JOBMAXSTARTTIME 2:00:00` *Jobs may attempt to start for up to 2 hours before being canceled by the scheduler*

JOBMAXTASKCOUNT	
Format	*<INTEGER>*
Default	*4096*
Description	Specifies the total number of tasks allowed per job.
Example	`JOBMAXTASKCOUNT 226000`

JOBMIGRATEPOLICY	
Format	One of the following: *IMMEDIATE*, *JUSTINTIME*, or *AUTO*
Default	*AUTO*
Description	Upon using the msub command to submit a job, you can allocate the job to immediately (*IMMEDIATE*) migrate to the resource manager, or you can instruct Moab to only migrate the job to the resource manager when it is ready to run (*JUSTINTIME*). Specifying *AUTO* allows MOAB to determine on a per-job basis whether to use *IMMEDIATE* or *JUSTINTIME*.
Example	`JOBMIGRATEPOLICY JUSTINTIME`

JOBNAMEWEIGHT	
Format	*<INTEGER>*
Default	---
Description	Specifies the weight to be applied to the job's name if the Name contains an integer. See Attribute (ATTR) Factor.
Example	`JOBNAMEWEIGHT 1`

JOBNODEMATCHPOLICY	
Format	*EXACTNODE* or *EXACTPROC*
Default	---
Description	Specifies additional constraints on how compute nodes are to be selected. *EXACTNODE* indicates that Moab should select as many nodes as requested even if it could pack multiple tasks onto the same node. *EXACTPROC* indicates that Moab should select only nodes with exactly the number of processors configured as are requested per node even if nodes with excess processors are available.
Example	`JOBNODEMATCHPOLICY EXACTNODE` *In a PBS/Native job with resource specification* `nodes=<x>:ppn=<y>`*, Moab will allocate exactly <y> task on each of <x> distinct nodes.*

JOBPREEMPTMAXACTIVETIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*0*
Description	The amount of time in which a job may be eligible for preemption. See Job Preemption.
Example	`JOBPREEMPTMAXACTIVETIME 00:05:00` *A job is preemptable for the first 5 minutes of its run time.*

JOBPREEMPTMINACTIVETIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*0*
Description	The minimum amount of time a job must be active before being considered eligible for preemption. See Job Preemption.
Example	`JOBPREEMPTMINACTIVETIME 00:05:00` *A job must execute for 5 minutes before Moab will consider it eligible for preemption.*

JOBPRIOACCRUALPOLICY	
Format	*ACCRUE* or *RESET*
Default	*ACCRUE*
Description	Specifies how Moab should track the dynamic aspects of a job's priority. *ACCRUE* indicates that the job will accrue queuetime based priority from the time it is submitted unless it violates any of the policies not specified in JOBPRIOEXCEPTIONS. *RESET* indicates that it will accrue priority from the time it is submitted unless it violates any of the JOBPRIOEXCEPTIONS. However, with *RESET*, if the job does violate JOBPRIOEXCEPTIONS then its queuetime based priority will be reset to 0. JOBPRIOACCRUALPOLICY is a global parameter, but can be configured to work only in QOSCFG: `QOSCFG[arrays]  JOBPRIOACCRUALPOLICY=ACCRUE` The following old JOBPRIOACCRUALPOLICY values have been **deprecated** and should be adjusted to the following values: • *QUEUEPOLICY*= *ACCRUE* and JOBPRIOEXCEPTIONS*SOFTPOLICY*,*HARDPOLICY* • *QUEUEPOLICYRESET*= *RESET* and JOBPRIOEXCEPTIONS*SOFTPOLICY*,*HARDPOLICY* • *ALWAYS*= *ACCRUE* and JOBPRIOEXCEPTIONS*ALL* • *FULLPOLICY*= *ACCRUE* and JOBPRIOEXCEPTIONS*NONE* • *FULLPOLICYRESET*= *RESET* and JOBPRIOEXCEPTIONS*NONE*
Example	`JOBPRIOACCRUALPOLICY   RESET` *Moab will adjust the job's dynamic priority subcomponents, i.e., QUEUETIME, XFACTOR, and TARGETQUEUETIME, etc. each iteration that the job does not violate any JOBPRIOEXCEPTIONS, if it is found in violation, its queuetime will be reset to 0.*

JOBPRIOEXCEPTIONS

Format	Comma delimited list of any of the following: *DEFER*, *DEPENDS*, *SOFTPOLICY*, *HARDPOLICY*, *IDLEPOLICY*, *USERHOLD*, *BATCHHOLD*, and *SYSTEMHOLD* (*ALL* or *NONE* can also be specified on their own)
Default	*NONE*
Description	Specifies exceptions for calculating a job's dynamic priority (QUEUETIME, XFACTOR, TARGETQUEUETIME). Normally, when a job violates a policy, is placed on hold, or has an unsatisfied dependency, it will not accrue priority. Exceptions can be configured to allow a job to accrue priority in spite of any of these violations. With DEPENDS a job will increase in priority even if there exists an unsatisfied dependency. With *SOFTPOLICY*, *HARDPOLICY*, or *IDLEPOLICY* a job can accrue priority despite violating a specific limit. With *DEFER*, *USERHOLD*, *BATCHHOLD*, or *SYSTEMHOLD* a job can accrue priority despite being on hold. **JOBPRIOEXCEPTIONS** is a global parameter, but can be configured to work only in QOSCFG: `QOSCFG[arrays]  JOBPRIOEXCEPTIONS=IDLEPOLICY`
Example	`JOBPRIOEXCEPTIONS BATCHHOLD,SYSTEMHOLD,DEPENDS` *Jobs will accrue priority in spite of batchholds, systemholds, or unsatisfied dependencies.*

JOBPRIOF

Format	*<ATTRIBUTE>[<VALUE>]=<PRIORITY>* where *<ATTRIBUTE>* is one of **ATTR**, **GRES** or **STATE**
Default	---
Description	Specifies attribute priority weights for jobs with specific attributes, generic resource requests, or states. State values must be one of the standard Moab job states. See Attribute-Based Job Prioritization.
Example	`JOBPRIOF         STATE[Running]=100  STATE[Suspended]=1000  ATTR[PREEMPTEE]=200` `GRES[biocalc]=5` `ATTRATTRWEIGHT   1` `ATTRSTATEWEIGHT  1` *Moab will adjust the job's dynamic priority subcomponents.*

JOBPURGETIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*0* (purge immediately if the resource manager does not report the job)
Description	The amount of time Moab will keep a job record which is no longer reported by the resource manager. Useful when using a resource manager which *drops* information about a job due to internal failures. See JOBCPURGETIME.
Example	`JOBPURGETIME 00:05:00` *Moab will maintain a job record for 5 minutes after the last update regarding that object received from the resource manager.*

JOBREJECTPOLICY	
Format	One or more of *CANCEL*, *HOLD*, *IGNORE* (beta), *MAIL*, or *RETRY*
Default	*HOLD*
Description	Specifies the action to take when the scheduler determines that a job can never run. *CANCEL* issues a call to the resource manager to cancel the job. *HOLD* places a *batch* hold on the job preventing the job from being further evaluated until released by an administrator. (**Note**: Administrators can dynamically alter job attributes and possibly *fix* the job with mjobctl -m.) With *IGNORE*(currently in beta), the scheduler will allow the job to exist within the resource manager queue but will neither process it nor report it. MAIL will send email to both the admin and the user when rejected jobs are detected. If *RETRY* is set, then Moab will allow the job to remain idle and will only attempt to start the job when the policy violation is resolved. Any combination of attributes may be specified. See QOSREJECTPOLICY.
Example	`JOBREJECTPOLICY  MAIL,CANCEL`

JOBREMOVEENVVARLIST	
Format	Comma-delimited list of strings
Default	---
Description	Moab will remove the specified environment variables from the job's environment before migrating the job to its destination resource manager. This is useful when jobs submit themselves from one cluster to another with the full environment. This parameter is currently only supported with TORQUE resource managers.
Example	`JOBREMOVEENVVARLIST PBS_SERVER,TZ` *Moab will remove the environment variables PBS_SERVER and TZ before submitting jobs.*

JOBRETRYTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	00:00:60
Description	Period of time Moab will continue to attempt to start a job which has failed to start due to transient failures or which has successfully started and was then rejected by the resource manager due to transient failures. (For related information, see Reservation Policies, DEFERSTARTCOUNT, DEFERTIME, RESERVATIONRETRYTIME, NODEFAILURERESERVETIME, and GUARANTEEDPREEMPTION.)
Example	`JOBRETRYTIME   00:05:00` *Moab will try for up to 5 minutes to restart jobs if the job start has failed due to transient errors.*

LIMITEDJOBCP	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether there should be limited job checkpointing (see Consideration for Large Clusters). With **LIMITEDJOBCP** enabled, Moab will only checkpoint a job if it is modified with **mjobctl** on page 216 or if it has been submitted with **msub** on page 290 but has not migrated. In all other cases, Moab does not checkpoint the job and all Moab-specific information (such as messages attached to the job) is lost. No TORQUE-specific information will be lost.
Example	`LIMITEDJOBCP TRUE` *Moab will only maintain scheduler checkpoint information for jobs with explicitly modified job attributes. Some minor job performance and usage statistics may be lost.*

LIMITEDNODECP	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether there should be limited node checkpointing (see Consideration for Large Clusters).
Example	`LIMITEDNODECP TRUE` *Moab will only maintain scheduler checkpoint information for nodes with explicitly modified job attributes. (some minor node performance and usage statistics may be lost)*

LOADALLJOBCP	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether Moab should load, during startup, all non-completed jobs in the checkpoint files regardless of whether or not their corresponding resource managers are active. For example, this allows source peers to continue showing remote jobs in the queue based on checkpointed info, even though the destination peer is offline.
Example	`LOADALLJOBCP TRUE` *Moab will load, at startup, all non-completed jobs from all checkpoint files.*

LOCKFILE	
Format	*<STRING>*
Default	---
Description	Specifies the path for the lock (pid) file used by Moab.
Example	`LOCKFILE /var/spool/moab/lock`

LOGDIR	
Format	*<STRING>*
Default	log
Description	Specifies the directory in which log files will be maintained. If specified as a relative path, **LOGDIR** will be relative to *$(MOABHOMEDIR)* See Logging Overview for more information.
Example	`LOGDIR /var/spool/moab` *Moab will record its log files directly into the* `/var/spool/moab` *directory*

LOGFACILITY

Format	Colon delimited list of one or more of the following: *CORE*, *SCHED*, *SOCK*, *UI*, *LL*, *CONFIG*, *STAT*, *SIM*, *STRUCT*, *FS*, *CKPT*, *BANK*, *RM*, *PBS*, *WIKI*, *ALL*
Default	*ALL*
Description	Specifies which types of events to log (see Logging Overview).
Example	`LOGFACILITY RM:PBS` *Moab will log only events involving general resource manager or PBS interface activities.*

LOGFILE

Format	*<STRING>*
Default	*moab.log*
Description	Name of the Moab log file. This file is maintained in the directory pointed to by *<LOGDIR>* unless *<LOGFILE>* is an absolute path (see Logging Overview)
Example	`LOGFILE moab.test.log` *Log information will be written to the file moab.test.log located in the directory pointed to by the LOGDIR parameter.*

LOGFILEMAXSIZE

Format	*<INTEGER>*
Default	*10000000*
Description	Maximum allowed size (in bytes) of the log file before it will be rolled (see Logging Overview).
Example	`LOGFILEMAXSIZE 50000000` *Log files will be rolled when they reach 50 MB in size*

LOGFILEROLLDEPTH	
Format	*<INTEGER>*
Default	*3*
Description	Number of old log files to maintain (i.e., when full, `moab.log` will be renamed `moab.log.1`, `moab.log.1` will be renamed `moab.log.2`, ...). See Logging Overview.
Example	`LOGFILEROLLDEPTH 5` *Moab will maintain and roll the last 5 log files.*

LOGLEVEL	
Format	*<INTEGER>* (0-9)
Default	*0*
Description	Specifies the verbosity of Moab logging where 9 is the most verbose (**Note**: each logging level is approximately an order of magnitude more verbose than the previous level). See Logging Overview.
Example	`LOGLEVEL 4` *Moab will write all Moab log messages with a threshold of 4 or lower to the `moab.log` file.*

LOGLEVELOVERRIDE

Format	*<BOOLEAN>*
Default	*FALSE*
Description	When this parameter is on, if someone runs a command with `--loglevel=<x>`, that loglevel, if higher than the current loglevel, is used on the scheduler side for the duration of the command. All logs produced during that time are put into a separate log file (this creates a "gap" in the normal logs). This can be very useful for debugging, but it is recommend that this be used only when diagnosing a specific problem so that users can't affect performance by submitting multiple `--loglevel` commands. This parameter does not work with threaded commands (such as showq, mdiag -n, and mdiag -j).
Example	`LOGLEVELOVERRIDE    TRUE`

LOGPERMISSIONS

Format	*<INTEGER>*
Default	*644*
Description	Specifies the octal number that represents read, write, and execute permissions.
Example	`LOGPERMISSIONS 600` *Allows the file owner to read and write permissions, but denies rights to the group and others.*

LOGROLLACTION	
Format	*<STRING>*
Default	---
Description	Specifies a script to run when the logs roll. The script is run as a trigger and can be viewed using mdiag -T. For example, a script can be specified that always moves the first rolled log file, `moab.log.1`, to an archive directory for longer term storage.
Example	`LOGROLLACTION /usr/local/tools/logroll.pl`

MAILPROGRAM	
Format	[*<Full_Path_To_Mail_Command>* | *DEFAULT* | *NONE*][@*<DEFAULTMAILDOMAIN>*]
Default	*NONE*
Description	If set to *NONE*, no mail is sent. If set to *DEFAULT*, Moab sends mail via the system's default mail program (usually `/usr/bin/sendmail`). If set to the local path of a mail program, Moab uses the specified mail program to send mail. By default, Moab mail notification is disabled. To enable, you must set **MAILPROGRAM** to *DEFAULT* or specify some other locally available mail program. If the *default mail domain* is set, emails will be routed to this domain unless a per-user domain is specified using the **EMAILADDRESS** attribute of the USERCFG parameter. If neither of these values is set, Moab uses "@localhost" as the mail domain. See Notify Admins. For jobs, the email address used on the msub -M option overrides all other user email addresses. Additionally, administrators are notified in the case of job violations.
Example	`MAILPROGRAM DEFAULT` *Moab sends mail via the system's default mail program,* `/usr/bin/sendmail`. `MAILPROGRAM /usr/local/bin/sendmail@mydomain.com` *Moab sends mail via the mail program located at /usr/local/bin/sendmail with default mail domain @mydomain.com*

MAXGRES	
Format	*<INTEGER>*
Default	*512*
Description	Specifies how many generic resources Moab should manage.
Example	`MAXGRES 1024`

MAXGMETRIC	
Format	*<INTEGER>*
Default	*10*
Description	Specifies how many generic metrics Moab should manage.
Example	`MAXGMETRIC 20`

MAXJOB	
Format	*<INTEGER>*
Default	*4096*
Description	Specifies the maximum quantity of jobs for which Moab should allocate memory used for tracking jobs. If Moab is tracking the maximum quantity of jobs specified by this parameter, it rejects subsequent jobs submitted by any user since it has no memory left with which to track newly submitted jobs. If a user submitted a job with the msub command, this rejection behavior requires the user to resubmit the job at a later time after other jobs have completed, which frees memory in which Moab can place later-submitted jobs. If a user submitted a job with the TORQUE qsub command, TORQUE will automatically resubmit the job to Moab until Moab accepts it. The mdiag -S command indicates if any job overflows have occurred. If this parameter's value is changed, it does not go into effect until Moab restarts. Moab reads the parameter only on initial startup and uses its value to allocate the memory it uses to track jobs.
Example	`MAXJOB 45000`

MAXNODE	
Format	*<INTEGER>*
Default	*5120*
Description	Specifies the maximum number of compute nodes supported.
Example	`MAXNODE 10000`

MAXRSVPERNODE	
Format	*<INTEGER>*
Default	*48*
Description	Specifies the maximum number of reservations on a node. For large SMP systems (>512 processors/node), Adaptive Computing advises adjusting the value to approximately twice the average sum of admin, standing, and job reservations present. A second number, led by a comma, can also be specified to set a maximum number of reservations for nodes that are part of the SHARED partition. The maximum possible value of **MAXRSVPERNODE** is *8192* for a global node and *4096* for any other node. Moab must be restarted for any changes to this parameter to take effect. The command mdiag -S indicates whether any node reservation overflows have occurred. See Considerations for Large Clusters. Do not lower the MAXRSVPERNODE value while there are active jobs in the queue. This can lead to queue instability and certain jobs could become stuck or disconnected from the system.
Example	`MAXRSVPERNODE 64` *64 is the maximum number of reservations on a single node.* `MAXRSVPERNODE 100,7000` *100 is the maximum number of reservations on a single node, and 7000 is the maximum number of reservations for global nodes.*

MEMREFRESHINTERVAL

Format	*[[[DD:]HH:]MM:]:SS* | job:*<COUNT>*
Default	---
Description	Specifies the time interval or total job query count at which Moab will perform garbage collection to free memory associated with resource manager API's which possess memory leaks (i.e., Loadleveler, etc.).
Example	`# free memory associated with leaky RM API` `MEMREFRESHINTERVAL 24:00:00` *Moab will perform garbage collection once every 24 hours.*

MEMWEIGHT

Format	*<INTEGER>*
Default	0
Description	Specifies the coefficient to be multiplied by a job's MEM (dedicated memory in MB) factor. See Resource Priority Overview.
Example	`RESWEIGHT 10` `MEMWEIGHT 1000` *Each job's priority will be increased by 10 * 1000 * <request memory>.*

MESSAGEQUEUEADDRESS	
Format	The IP address of the machine on which Moab is generating events.
Default	* (all)
Description	When a user subscribes to the events Moab provides and delivers via zeroMQ, s/he must do so by specifying `tcp://`*`<ipAddress>`*`:`*`<port>`*. MESSAGEQUEUEADDRESS specifies the *<ipAddress>*, which must match the IP address of the machine on which Moab is installed. To specify the port, see **MESSAGEQUEUEPORT** on page 975.
Example	`MESSAGEQUEUEADDRESS 10.1.0.10` *To subscribe to Moab events, users must use* `tcp://10.1.0.10:<port>`.

MESSAGEQUEUEPORT	
Format	The port of the machine on which Moab is generating events.
Default	*5563*
Description	When a user subscribes to the events Moab provides and delivers via zeroMQ, s/he must do so by specifying `tcp://`*`<ipAddress>`*`:`*`<port>`*. MESSAGEQUEUEPORT specifies the *<port>*, which must match the port of the machine on which Moab is installed. To specify the IP address, see **MESSAGEQUEUEADDRESS** on page 975.
Example	`MESSAGEQUEUEPORT 1010` *To subscribe to Moab events, users must use* `tcp://<ipAddress>:1010`.

MESSAGEQUEUESECRETKEY	
Format	*<STRING>*
Default	---
Description	Causes Moab to encrypt the events delivered via zeroMQ using the Advanced Encryption Standard (AES) algorithm. Must be a Base64-encoded, 128-bit (16-byte) key. Messages will be encrypted using AES in CBC mode where inputs are padded with PKCS5 padding. The initialization vector is calculated by using an MD5 hash of the key specified in **MESSAGEQUEUESECRETKEY**. **MESSAGEQUEUESECRETKEY** can only be specified in the `moab-private.cfg` file.
Example	`MESSAGEQUEUESECRETKEY 1r6RvfqJa6voezy5wAx0hw==`

MINADMINSTIME	
Format	*<INTEGER>*
Default	*60* seconds
Description	Specifies the minimum time a job will be suspended if suspended by an administrator or by a scheduler policy.
Example	`MINADMINSTIME 00:10:00` *Each job suspended by administrators or policies will stay in the suspended state for at least 10 minutes.*

MISSINGDEPENDENCYACTION	
Format	*CANCEL*, *HOLD*, or *RUN*
Default	*HOLD*
Description	Controls what Moab does with a dependent job when its dependency job cannot be found when Moab evaluates the dependent job for scheduling. This only affects jobs whose dependent job cannot be found.
Example	`MISSINGDEPENDENCYACTION CANCEL` *Any job that has a dependent job that cannot be found is canceled.*

MSUBQUERYINTERVAL	
Format	*<INTEGER>*
Default	*5* seconds
Description	Specifies the length of the interval (in seconds) between job queries when using msub -K. Jobs submitted with the -K option query the scheduler every **MSUBQUERYINTERVAL** seconds until the job is completed. **MSUBQUERYINTERVAL** can exist as an environment variable. Any value in `moab.cfg` overrides the environment variable. **Note**: If **MSUBQUERYINTERVAL** is set to 0, the -K option will be disabled. Jobs will still submit correctly, but the client will not continue to check on the job.
Example	`MSUBQUERYINTERVAL  60` *If a user uses the msub -K command, the client remains open and queries the server every 60 seconds until the job completes.*

NODEACCESSPOLICY

Format	One of the following: SHARED, SHAREDONLY, SINGLEJOB, SINGLETASK, SINGLEUSER, or UNIQUEUSER
Default	**SHARED**
Description	Specifies how node resources will be shared by various tasks (See the Node Access Overview for more information).
Example	`NODEACCESSPOLICY SINGLEUSER` Moab will allow resources on a node to be used by more than one job provided that the jobs are all owned by the same user.

NODEALLOCATIONPOLICY

Format	One of the following: FIRSTAVAILABLE, LASTAVAILABLE, MINRESOURCE, CPULOAD, LOCAL, CONTIGUOUS, MAXBALANCE, PRIORITY, or PLUGIN.
Default	*LASTAVAILABLE*
Description	Specifies how Moab should allocate available resources to jobs. See Node Allocation Overview for more information. If **ENABLEHIGHTHROUGHPUT** on page 929 is *TRUE*, you must set **NODEALLOCATIONPOLICY** to *FIRSTAVAILABLE*.
Example	`NODEALLOCATIONPOLICY MINRESOURCE` *Moab will apply the node allocation policy MINRESOURCE to all jobs by default.*

NODEALLOCRESFAILUREPOLICY	
Format	One of the following: *CANCEL*, *HOLD*, *IGNORE*, *MIGRATE*, *NOTIFY*, or *REQUEUE*
Default	*NONE*
Description	Specifies how Moab should handle active jobs which experience node failures during execution. See the RESFAILPOLICY resource manager extension or the Node Availability Overview.
Example	`NODEALLOCRESFAILUREPOLICY REQUEUE` *Moab will requeue jobs which have allocated nodes fail during execution.*

NODEAVAILABILITYPOLICY	
Format	*<POLICY>[:<RESOURCETYPE>] ...* where *<POLICY>* is one of **COMBINED**, **DEDICATED**, or **UTILIZED** and *<RESOURCETYPE>* is one of *PROC*, *MEM*, *SWAP*, or *DISK*
Default	**COMBINED**
Description	Specifies how available node resources are reported. Moab uses the following calculations to determine the amount of available resources: **Dedicated**(use what Moab has scheduled to be used): Available = Configured - Dedicated **Utilized**(use what the resource manager is reporting is being used): Available = Configured - Utilized **Combined**(use the larger of dedicated and utilized): Available = Configured - (MAX(Dedicated,Utilized)) Moab marks a node as busy when it has no available processors, so **NODEAVAILABILTYPOLICY**, by affecting how many processors are reported as available, also affects node state. See Node Availability Policies for more information.
Example	`NODEAVAILABILITYPOLICY DEDICATED:PROCS COMBINED:MEM` *Moab will ignore resource utilization information in locating available processors for jobs but will use both dedicated and utilized memory information in determining memory availability.*

NODEBUSYSTATEDELAYTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*0:01:00* (one minute)
Description	Length of time Moab will assume busy nodes will remain unavailable for scheduling if a system reservation is not explicitly created for the node.
Example	`NODEBUSYSTATEDELAYTIME 0:30:00` *Moab will assume busy nodes are not available for scheduling for at least 30 minutes from the current time. Thus, these nodes will never be allocated to starting jobs. Also, these nodes will only be available for reservations starting more than 30 minutes in the future.*

NODECATCREDLIST

Format	*<LABEL>=<NODECAT>[,<NODECAT>]...[<LABEL>=<NODECAT>[,<NODECAT>]...]...* where *<LABEL>* is any string and *<NODECAT>* is one of the defined node categories.
Default	---
Description	If specified, Moab will generate node category groupings and each iteration will assign usage of matching resources to pseudo-credentials with a name matching the specified label. See the Node Categorization section of the Admin manual for more information.
Example	`NODECATCREDLIST down=BatchFailure,HardwareFailure,NetworkFailure idle=Idle` *Moab will create a **down** user, group, account, class, and QoS and will associate BatchFailure, HardwareFailure, and NetworkFailure resources with these credentials. Additionally, Moab will assign all Idle resources to matching **idle** credentials.*

NODECFG[X]

Format	List of space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: ACCESS, CHARGERATE, FEATURES, FLAGS, GRES, MAXJOB, MAXJOBPERUSER, MAXLOAD, MAXPE, NODEINDEX, NODETYPE, OSLIST, PARTITION, **POWERPOLICY** on page 560, PRIORITY, PRIORITYF, PROCSPEED, RACK, RADISK, SLOT, SPEED, or TRIGGER
Default	---
Description	Specifies node-specific attributes for the node indicated in the array field. See the General Node Administration Overview for more information.
Example	`NODECFG[nodeA] MAXJOB=2 SPEED=1.2` *Moab will only allow 2 simultaneous jobs to run on node nodeA and will assign a relative machine speed of 1.2 to this node.*

NODEDOWNSTATEDELAYTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	-1 (never)
Description	Length of time Moab will assume down, drained (offline), or corrupt nodes will remain unavailable for scheduling if a system reservation is not explicitly created for the node. The default specification of "-1" causes Moab to never create job reservations on down nodes. See Node Availability for more information.
Example	`NODEDOWNSTATEDELAYTIME 0:30:00` *Moab will assume down, drained, and corrupt nodes are not available for scheduling for at least 30 minutes from the current time. Thus, these nodes will never be allocated to starting jobs. Also, these nodes will only be available for reservations starting more than 30 minutes in the future.*

NODEDOWNTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	---
Description	The maximum time a previously reported node remains unreported by a resource manager before the node is considered to be in the down state. This can happen if communication with a resource manager or a peer server is lost for more than the specified length of time, or if there is communication with the resource manager but it fails to report the node status.
Example	`NODEDOWNTIME 10:00` *If Moab loses communication with the resource manager for more than 10 minutes, it sets the state of all nodes belonging to that resource manager to DOWN.*

NODEDRAINSTATEDELAYTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	3:00:00 (three hours)
Description	Length of time Moab will assume drained nodes will remain unavailable for scheduling if a system reservation is not explicitly created for the node. Specifying "-1" will cause Moab to never create job reservations on drained nodes. See Node Availability for more information.
Example	`NODEDRAINSTATEDELAYTIME 0:30:00` *Moab will assume down, drained, and corrupt nodes are not available for scheduling for at least 30 minutes from the current time. Thus, these nodes will never be allocated to starting jobs. Also, these nodes will only be available for reservations starting more than 30 minutes in the future.*

NODEFAILURERESERVETIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*0:05:00*
Description	Duration of reservation Moab will place on any node in which it detects a failure from the resource manager (0 indicates no reservation will be placed on the node). See Node Availability for more information. See also RMCFG[] NODEFAILURERSVPROFILE. (For related information, see Reservation Policies, DEFERSTARTCOUNT, DEFERTIME, RESERVATIONRETRYTIME, JOBRETRYTIME, and GUARANTEEDPREEMPTION.)
Example	`NODEFAILURERESERVETIME 10:00` *Moab will reserve failed nodes for 10 minutes.*

NODEIDFORMAT

Format	*<STRING>*
Default	**$N**
Description	Specifies how a node id can be processed to extract possible node, rack, slot, and cluster index information. The value of the parameter may include the markers *$C* (cluster index), *$N* (node index), *$R* (rack index), or *$S* (slot index) separated by *(asterisk - representing any number of non-numeric characters) or other characters to indicate this encoding. See Node Selection for more information on use of node, rack, and slot indices.
Example	`NODEIDFORMAT *$R*$S` *Moab will extract rack and slot information from the cluster node ids (i.e. `tg-13s08`).*

NODEIDLEPOWERACTION

Format	[STANDBY | SUSPEND | SLEEP | HIBERNATE | SHUTDOWN | OFF]
Default	*OFF*
Description	Specifies what to do with a node that exceeds the NODEIDLEPOWERTHRESHOLD limit.
Example	`PARCFG[ALL] NODEIDLEPOWERACTION STANDBY` *Nodes that exceed the NODEIDLEPOWERTHRESHOLD limit are placed in standby.*

NODEIDLEPOWERTHRESHOLD

Format	*<INTEGER>*
Default	*60 seconds*
Description	Specifies how long to allow a node to be idle before performing a power action. Increasing the idle duration prevents power on/off thrashing.
Example	`NODEIDLEPOWERTHRESHOLD 300` *Moab will wait 5 minutes before performing a power action on a node that has become idle.*

NODEMAXLOAD

Format	*<DOUBLE>*
Default	*0.0*
Description	Specifies that maximum cpu load on an idle or running node. If the node's load reaches or exceeds this value, Moab will mark the node busy.
Example	`NODEMAXLOAD 0.75` *Moab will adjust the state of all idle and running nodes with a load >= .75 to the state busy.*

NODEMEMOVERCOMMITFACTOR	
Format	*<DOUBLE>*
Default	---
Description	The parameter overcommits available and configured memory and swap on a node by the specified factor (for example: mem/swap * factor). Used to show that the node has more mem and swap than it really does. Only works for PBS RM types.
Example	`NODEMEMOVERCOMMITFACTOR .5` *Moab will overcommit the memory and swap of the node by a factor of 0.5.*

NODEPOLLFREQUENCY	
Format	*<INTEGER>*
Default	*0* (Poll Always)
Description	Specifies the number of scheduling iterations between scheduler initiated node manager queries. If set to ' -2, Moab will never query the node manager daemons. If set to ' -1', Moab will only query on the first iteration. **Note**: this parameter is most often used with OpenPBS and PBSPro. It is not required when using TORQUE, LoadLeveler, LSF, or SGE as the resource managers.
Example	`NODEPOLLFREQUENCY 5` *Moab will update node manager based information every 5 scheduling iterations.*

NODESETATTRIBUTE

Format	*FEATURE* or *VARATTR*
Default	---
Description	Specifies the type of node attribute by which node set boundaries will be established. See Node Set Overview.
Example	`NODESETPOLICY     ONEOF` `NODESETATTRIBUTE  FEATURE` *Moab will create node sets containing nodes with common features.*

NODESETDELAY

Format	*[[[DD:]HH:]MM:]SS*
Default	*0:00:00*
Description	Causes Moab to attempt to span a job evenly across nodesets unless doing so delays the job beyond the requested **NODESETDELAY**. Must use with **NODESETPLUS** on page 988 set to *SPANEVENLY*; if you do not want to use *SPANEVENLY*, use **NODESETISOPTIONAL** on page 987 instead of **NODESETDELAY**.
Example	`NODESETPLUS SPANEVENLY` `NODESETDELAY     5:00` *Moab tries to span the job evenly across nodesets unless doing so delays the job by 5 minutes.*

NODESETISOPTIONAL

Format	*<BOOLEAN>*
Default	*TRUE*
Description	Specifies whether or not Moab will start a job if a requested node set cannot be satisfied. See Node Set Overview.
Example	`NODESETISOPTIONAL TRUE` *Moab will not block a job from running if its node set cannot be satisfied.*

NODESETLIST

Format	*<ATTR>[{ :,|}<ATTR>]...*
Default	---
Description	Specifies the list of node attribute values which will be considered for establishing node sets. See Node Set Overview.
Example	`NODESETPOLICY      ONEOF` `NODESETATTRIBUTE  FEATURE` `NODESETLIST       switchA,switchB` *Moab will allocate nodes to jobs either using only nodes with the switchA feature or using only nodes with the switchB feature.*

NODESETPLUS	
Format	*DELAY* or *SPANEVENLY*
Default	---
Description	Specifies how Moab distributes jobs among nodesets. See Node Set Overview. Neither *SPANEVENLY* nor *DELAY* values of the NODESETPLUS parameter will work with multi-req jobs or preemption.
Example	`NODESETPLUS SPANEVENLY` *Moab attempts to fit all jobs on a single nodeset or to span them evenly across a number of nodesets, unless doing so would delay a job beyond the requested NODESETDELAY.* `NODESETPLUS DELAY` *Moab attempts to schedule the job within a nodeset for the configured NODESETDELAY. If Moab cannot find space for the job to start within NODESETDELAY (Moab considers future workload to determine if space will open up in time and might create a future reservation), then Moab schedules the job and ignores the nodeset requirement.*

NODESETPOLICY	
Format	*ANYOF*, *FIRSTOF*, or *ONEOF*
Default	---
Description	Specifies how nodes will be allocated to the job from the various node set generated. See Node Set Overview.
Example	`NODESETPOLICY     ONEOF` `NODESETATTRIBUTE  NETWORK` *Moab will create node sets containing nodes with common network interfaces.*

NODESETPRIORITYTYPE	
Format	one of *AFFINITY*, *BESTFIT*, *WORSTFIT*, or *MINLOSS*
Default	*MINLOSS*
Description	Specifies how resource sets will be selected when more than one feasible resource can be found. See Node Set Overview.
Example	`NODESETPRIORITYTYPE BESTFIT` `NODESETATTRIBUTE    PROCSPEED` *Moab will select the resource set that most closely matches the set of resources requested.*

NODESYNCTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*00:10:00*
Description	Specifies the length of time after which Moab will sync up a node's expected state with an unexpected reported state. **IMPORTANT Note**: Moab will not start new jobs on a node with an expected state which does not match the state reported by the resource manager.
Example	`NODESYNCTIME 1:00:00`

NODETOJOBATTRMAP	
Format	Comma delimited list of node features
Default	---
Description	Job requesting the listed node features will be assigned a corresponding job attribute. These job attributes can be used to enable reservation access, adjust job priority or enable other capabilities.
Example	`NODETOJOBATTRMAP  fast,big` *Jobs requesting node feature* fast *or* big *will be assigned a corresponding job attribute.*

NODEUNTRACKEDRESDELAYTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*0:00:00*
Description	Length of time Moab will assume untracked generic resources will remain unavailable for scheduling if a system reservation is not explicitly created for the node. If NODEUNTRACKEDRESDELAYTIME is enabled and there is an untracked resource preventing a job from running, then the job remains in the idle queue instead of being deferred.
Example	`NODEUNTRACKEDRESDELAYTIME 0:30:00` *Moab will assume untracked generic resources are not available for scheduling for at least 30 minutes from the current time. Thus, these nodes will never be allocated to starting jobs. Also, these nodes will only be available for reservations starting more than 30 minutes in the future.*

NODEVMFEATURECHECKTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*0:10:00*
Description	The length of time between each Moab check on node and VM features. If a running VM requires a feature but the resource manager is no longer reporting that feature on the VM's host node, Moab migrates the VM to a node that has the feature. If no other node has that feature, no migration occurs.
Example	`NODEVMFEATURECHECKTIME 10:00` *Moab checks node and VM features every 10 minutes.*

NODEVMREQATTRCHECKTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*0:10:00*

NODEVMREQATTRCHECKTIME	
Description	The length of time between each Moab check on a VM's requested node attributes. If a running VM requires node attributes but the resource manager is no longer reporting requested attributes on the VM's host node, Moab migrates the VM to a node that has the requested attributes. If no other node has the requested attributes, no migration occurs.
Example	`NODEVMREQATTRCHECKTIME 10:00` *Moab checks requested node attributes of a node running a VM every 10 minutes.*

NODEWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight which will be applied to a job's requested node count before this value is added to the job's cumulative priority. **Note**: this weight currently only applies when a nodecount is specified by the user job. If the job only specifies tasks or processors, no node factor will be applied to the job's total priority. This will be rectified in future versions.
Example	`NODEWEIGHT 1000`

NOLOCALUSERENV	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If *TRUE*, specifies that a user's UserID, GroupID, and HomeDirectory are available on the Moab server host.
Example	`NOLOCALUSERENV TRUE`

NOJOBHOLDNORESOURCES	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If *TRUE*, Moab does not place a hold on jobs that don't have feasible resources. For example, suppose there are 20 processors available for ClassA and 50 processors for the entire system. If a job requests 21 or more processors from ClassA, or 51 or more processors from the entire system, Moab idles the job (instead of putting a hold on it) until the resources become available.
Example	`NOJOBHOLDNORESOURCES TRUE`

NOTIFICATIONPROGRAM	
Format	*<STRING>*
Default	---
Description	Specifies the name of the program to handle all notification call-outs.
Example	`NOTIFICATIONPROGRAM   tools/notifyme.pl`

NOWAITPREEMPTION	
Format	*<BOOLEAN>*
Default	---
Description	Generally when a job is trying to preempt another, it just waits for the original jobs that it chose to preempt to end. If this parameter is on, the preemptor will continue trying to preempt jobs until it can get in.
Example	`NOWAITPREEMPTION   TRUE`

OSCREDLOOKUP	
Format	NEVER
Default	---
Description	Disables all Moab OS credential lookups, including UID, GID, user to group mappings, and any other OS specific information. Setting OSCREDLOOKUP by itself does not allow job submission; additional configuration is required. When submitting jobs from user accounts that do not exist on the head node (where Moab HPC Suite and TORQUE are running), you must also set the PROXYJOBSUBMISSION flag in addition to specifying configuration settings in the resource manager configuration file. See the example that follows for information on required resource manager settings.
Example	`OSCREDLOOKUP NEVER` `RMCFG[] FLAGS=PROXYJOBSUBMISSION` To allow job submission, in the TORQUE configuration file (torque.cfg): `VALIDATEPATH FALSE` Run the following qmgr directive: `set server disable_server_id_check = True` Restart both Moab HPC Suite and pbs_server.

PARALLOCATIONPOLICY	
Format	One of *BestFit*, *BestFitP*, *FirstStart*, *LoadBalance*, *LoadBalanceP*, *Random*, or *RoundRobin*
Default	*FirstStart*
Description	Specifies the approach to use to allocate resources when more than one eligible partition can be found. See Grid Scheduling Policies for more information.
Example	`PARALLOCATIONPOLICY  LOADBALANCE` *New jobs will be started on the most lightly allocated partition.*

PARCFG	
Format	NODEPOWEROFFDURATION, NODEPOWERONDURATION, NODEALLOCATIONPOLICY or one or more key-value pairs as described in the Partition Overview
Default	---
Description	Specifies the attributes, policies, and constraints for the given partition.
Example	`PARCFG[oldcluster] MAX.WCLIMIT=12:00:00` *Moab will not allow jobs to run on the oldcluster partition which has a wallclock limit in excess of 12 hours.*

PBSACCOUNTINGDIR	
Format	*<PATH>*
Default	---
Description	When specified, Moab will write out job events in standard PBS/ TORQUE tracejob format to the specified directory using the standard PBS/TORQUE log file naming convention.
Example	`PBSACCOUNTINGDIR /var/spool/torque/sched_logs/` *Job events will be written to the specified directory (can be consumed by PBS's tracejob command).*

PERPARTITIONSCHEDULING	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	By default Moab's scheduling routine schedules each job on each partition using the following algorithm: *prioritize* *foreach (job)* *find the partition on which that job should run* *schedule job* In this model, a job's priority is the same on each partition as it uses a single global priority. Because a job's priority is the same on every partition, Moab prioritizes the queue once and then schedules the prioritized queue across all partitions. When `PERPARTITIONSCHEDULING TRUE` is set, the following algorithm is used: *foreach (partition)* *prioritize* *foreach (job)* *schedule job* In this case, each partition may have a unique priority configuration and Moab will re-prioritize the jobs for each partition on the system. Each job is prioritized and scheduled on each partition. See **PARCFG** on page 994 for more information. Also, note that Moab will order the partitions as they are discovered in the moab.cfg file. Partitions should be explicitly ordered via PARCFG in the moab.cfg file.
Example	`PERPARTITIONSCHEDULING TRUE` `PARCFG[p1] CONFIGFILE=/opt/moab/etc/p1.cfg` `PARCFG[p2] CONFIGFILE=/opt/moab/etc/p2.cfg` *Rather than prioritizing the job queue once, Moab prioritizes the job queue for each partition, p1 and p2 respectively, and schedules each partition in turn using the policies located in their respective configuration files. (See* **Per-Partition Settings** on page 498 *for more information).*

PEWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the coefficient to be multiplied by a job's PE (processor equivalent) priority factor.
Example	`RESWEIGHT 10` `PEWEIGHT  100` *Each job's priority will be increased by 10 * 100 * its PE factor.*

PREEMPTPOLICY	
Format	one of the following: CANCEL, REQUEUE, SUSPEND, or CHECKPOINT
Default	*REQUEUE*
Description	Specifies how preemptable jobs will be preempted. **Note:** If this policy is set to REQUEUE, preemptible jobs should be marked as *RESTARTABLE*. If this policy is set to SUSPEND, preemptible jobs should be marked as *SUSPENDABLE*. **Note:** Moab uses preemption escalation to preempt resources if the specified preemption facility is not applicable. This means if the policy is set to *SUSPEND* and the job is not *SUSPENDABLE*, Moab may attempt to requeue or even cancel the job.
Example	`PREEMPTPOLICY CHECKPOINT` *Jobs that are to be preempted will be checkpointed and restarted at a later time.*

PREEMPTPRIOJOBSELECTWEIGHT	
Format	*<DOUBLE>*
Default	*256.0*
Description	Determines which jobs to preempt based on size or priority. The higher the value, the more emphasis is placed on the priority of the job, causing the lower priority jobs to be preempted first. The lower the value, the more emphasis is placed on the size of the job, causing the smaller jobs to be preempted first. If set to *0*, job priority will be ignored, job size will take precedence and the smallest jobs will be preempted. The special setting of *-1* places the emphasis solely on resource utilization. This means that jobs will be preempted in a manner that keeps the resource utilization at the highest level, regardless of job priority or size.
Example	`PREEMPTPRIOJOBSELECTWEIGHT 220.5`

PREEMPTRTIMEWEIGHT	
Format	*<DOUBLE>*
Default	*0*
Description	If set to anything other than *0*, a job's remaining time is added into the calculation of which jobs will be preempted. If a positive weight is specified, jobs with a longer remaining time are favored. If a negative weight is specified, jobs with a shorter remaining time are favored.
Example	`PREEMPTRTWEIGHT 1.5`

PREEMPTSEARCHDEPTH

Format	*<INTEGER>*
Default	*unlimited*
Description	Specifies how many preemptible jobs will be evaluated as potential targets for serial job preemptors. See Preemption Overview for more information.
Example	`PREEMPTSEARCHDEPTH 8` *Serial job preemptors will only consider the first 8 feasible preemptee jobs when determining the best action to take.*

PRIORITYTARGETDURATION

Format	*[[[DD:]HH:]MM:]SS*
Default	---
Description	Specifies the *ideal* job duration which will maximize the value of the WALLTIMEWEIGHT priority factor. If specified, this factor will be calculated as the distance from the ideal. Consequently, in most cases, the associated subcomponent weight should be set to a negative value.
Example	`WALLTIMEWEIGHT          -2500` `PRIORITYTARGETDURATION  1:00:00`

PRIORITYTARGETPROCCOUNT

Format	*<INTEGER>*{+|-|%}
Default	---
Description	Specifies the ideal job requested proc count which will maximize the value of the PROCWEIGHT priority factor. If specified, this factor will be calculated as the distance from the ideal (proc count - ideal = coefficient of PROCWEIGHT). Consequently, in most cases, the associated subcomponent weight should be set to a negative value.
Example	`PROCWEIGHT               -1000` `PRIORITYTARGETPROCCOUNT  64`

PROCWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the coefficient to be multiplied by a job's requested processor count priority factor.
Example	`PROCWEIGHT 2500`

PROFILECOUNT	
Format	*<INTEGER>*
Default	*600*
Description	Specifies the number of statistical profiles to maintain. PROFILECOUNT must be set high enough that at least one day of statistics is maintained. The statistics time window can be determined by measuring PROFILEDURATION * PROFILECOUNT. If PROFILEDURATION is one hour then PROFILECOUNT must be at least 24 so 24 hours worth of statistics are maintained. If PROFILEDURATION is 30:00 then PROFILECOUNT must be set to at least 48. If PROFILECOUNT is not high enough for at least one day of statistics, Moab adjusts it automatically.
Example	`PROFILECOUNT 300`

PROFILEDURATION	
Format	*[[[DD:]HH:]MM:]SS*
Default	*00:30:00*
Description	Specifies the duration of each statistical profile. The duration cannot be more than 24 hours, and any specified duration must be a factor of 24. For example, factors of 1/4, 1/2, 1, 2, 3, 4, 6, 8, 12, and 24 are acceptable durations.
Example	`PROFILEDURATION 24:00:00`

PURGETIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*0*
Description	The amount of time Moab will keep a job or node record for an object no longer reported by the resource manager. Useful when using a resource manager which 'drops' information about a node or job due to internal failures. Note: This parameter is superseded by **JOBPURGETIME** on page 964.
Example	`PURGETIME 00:05:00` *Moab will maintain a job or node record for 5 minutes after the last update regarding that object received from the resource manager.*

PUSHCACHETOWEBSERVICE

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether or not you want to send cache objects (nodes, jobs, services, etc.) to Moab Web Services.
Example	`PUSHCACHETOWEBSERVICE TRUE`

QOSCFG[<QOSID>]

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: General Credential Flags, **PRIORITY**, ENABLEPROFILING, **FSTARGET**, JOBPRIOACCRUALPOLICY, JOBPRIOEXCEPTIONS, MEMBERULIST, **QTWEIGHT**, **QTTARGET**, **XFWEIGHT**, **XFTARGET**, PREEMPTMINTIME, PREEMPTMAXTIME, **PREEMPTQTTHRESHOLD**, **PREEMPTXFTHRESHOLD**, PREEMPTEES, **RSVQTTHRESHOLD**, **RSVXFTHRESHOLD**, **ACLBLTHRESHOLD**, **ACLQTTHRESHOLD**, **ACLXFTHRESHOLD**, **PLIST**, QFLAGS, or a usage limit.
Default	---
Description	Specifies QOS specific attributes. See the flag overview for a description of legal flag values. See the QOS Overview section for further details.
Example	`QOSCFG[commercial] PRIORITY=1000 MAXJOB=4 MAXPROC=80` *Moab will increase the priority of jobs using QOS commercial, and will allow up to 4 simultaneous QOS commercial jobs with up to 80 total allocated processors.*

QOSDEFAULTORDER

Format	Comma-delimited list of QOS names.
Default	---
Description	Sets a global QOS default order for all QOS's which overrides any specific default QOS. If the order is defined as `b,a,c` and a user has access to `c,a` and submits a job without requesting a specific QOS, the job is assigned `a` as the default QOS.
Example	`QOSDEFAULTORDER  b,a,c` *If the job does not have a QOS specified, it is assigned a QOS from the QOSDEFAULTORDER list (if the user has access to one of them).*

QOSISOPTIONAL

Format	*<BOOLEAN>*
Default	*FALSE*
Description	An entity's default QOS will be the first QOS specified in the QLIST parameter. When this parameter is set to *TRUE* the default QOS for the associated credential (user, account, class, etc.) will not be automatically set to the first QOS specified in the QLIST.
Example	`QOSISOPTIONAL TRUE` `USERCFG[bob]  QLIST=high,low` *Moab will set the QOSList for user bob to high and low but will not set the QDEF. Should bob decide to submit to a particular QOS he will have to do so manually.*

QOSREJECTPOLICY

Format	One or more of *CANCEL*, *HOLD*, *IGNORE*, or *MAIL*
Default	*HOLD*(*IGNORE* for SLURM users)
Description	Specifies the action to take when Moab determines that a job cannot access a requested QoS. *CANCEL* issues a call to the resource manager to cancel the job. *HOLD* places a *batch* hold on the job preventing the job from being further evaluated until released by an administrator. (**Note**: Administrators can dynamically alter job attributes and possibly *fix* the job with mjobctl -m.) With *IGNORE*, Moab will ignore the QoS request and schedule the job using the default QoS for that job. MAIL will send email to both the admin and the user when QoS request violations are detected. Most combinations of attributes may be specified; however, if both *MAIL* and *IGNORE* are specified, Moab will not implement *MAIL*. Similarly, while *CANCEL* and *HOLD* are mutually exclusive, *CANCEL* will supersede *HOLD* if both are specified. (see JOBREJECTPOLICY).
Example	`QOSREJECTPOLICY  MAIL,CANCEL`

QOSWEIGHT	
Format	*<INTEGER>*
Default	*1*
Description	Specifies the weight to be applied to the qos priority of each job (see Credential (CRED) Factor).
Example	`QOSWEIGHT 10`

QUEUETIMECAP	
Format	*<DOUBLE>*
Default	*0* (NO CAP)
Description	Specifies the maximum allowed absolute pre-weighted queuetime priority factor.
Example	`QUEUETIMECAP    10000` `QUEUETIMEWEIGHT 10` *A job that has been queued for 40 minutes will have its queuetime priority factor calculated as 'Priority = QUEUETIMEWEIGHT * MIN(10000,40)'.*

QUEUETIMEWEIGHT	
Format	*<INTEGER>*
Default	*1*
Description	Specifies multiplier applied to a job's queue time (in minutes) to determine the job's queuetime priority factor.
Example	`QUEUETIMEWEIGHT 20` *A job that has been queued for 4:20:00 will have a queuetime priority factor of 20 * 260.*

REALTIMEDBOBJECTS	
Format	Comma-delimited list of one or more of the following: *JOB*, *NODE*, *RSV* (reservation), *TRIG* (trigger), *VC* (virtual container). You can also specify *ALL* or *NONE*.
Default	*ALL*
Description	Specifies which objects Moab will store in the unixodbc database.
Example	`REALTIMEDBOBJECTS JOB,RSV,TRIG` *Moab stores jobs, reservations, and triggers in the uxodbc database. It will no longer record real time information about nodes and VCs.*

RECORDEVENTLIST

Format	One or more comma (',') or plus ('+') separated events of GEVENT, *ALLSCHEDCOMMAND*, *AMCREATE*, *AMDELETE*, *AMEND*, *AMPAUSE*, *AMQUOTE*, *AMRESUME*, *AMSTART*, *AMUPDATE*, *JOBCANCEL*, *JOBCHECKPOINT*, *JOBEND*, *JOBFAILURE*, *JOBMIGRATE*, *JOBMODIFY*, *JOBPREEMPT*, *JOBREJECT*, *JOBRESUME*, *JOBSTART*, *JOBSUBMIT*, *NODEDOWN*, *NODEFAILURE*, *NODEUP*, *QOSVIOLATION*, *RMDOWN*, *RMPOLLEND*, *RMPOLLSTART*, *RMUP*, *RSVCANCEL*, *RSVCREATE*, *RSVEND*, *RSVMODIFY*, *RSVSTART*, *SCHEDCOMMAND*, *SCHEDCYCLEEND*, *SCHEDCYCLESTART*, *SCHEDPAUSE*, *SCHEDSTART*, *SCHEDSTOP*, *VMCREATE*, *VMDESTROY*, *VMMIGRATE*, *VMPOWEROFF*, *VMPOWERON*, or *ALL*
Default	*JOBSTART*, *JOBCANCEL*, *JOBEND*, *JOBFAILURE*, *SCHEDPAUSE*, *SCHEDSTART*, *SCHEDSTOP*, *TRIGEND*, *TRIGFAILURE*, *TRIGSTART*
Description	Specifies which events should be recorded in the appropriate event file found in Moab's `stats/` directory. These events are recorded for both local and remotely staged jobs. (See Event Log Overview) **Note**: If a plus character is included in the list, the specified events will be added to the default list; otherwise, the specified list will replace the default list.
Example	`RECORDEVENTLIST JOBSTART,JOBCANCEL,JOBEND` *When a local and/or remote job starts, is canceled, or ends, the respective event will be recorded.*

REJECTDOSSCRIPTS

Format	*<BOOLEAN>*
Default	*TRUE*
Description	Moab rejects DOS-formatted scripts submitted with the msub command. This is useful if you use SLURM as your resource manager, since it does not handle DOS scripts well. For REJECTDOSSCRIPTS to work correctly, **FILTERCMDFILE** on page 936 must be *FALSE*. Otherwise, Moab modifies the script instead of rejecting it, leading to job errors.
Example	`REJECTDOSSCRIPTS FALSE` *Moab does not reject DOS-formatted scripts submitted with msub.*

REJECTINFEASIBLEJOBS	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If zero feasible nodes are found for a job among the currently available nodes on the cluster, the scheduler rejects the job. See JOBREJECTPOLICY for more information.
Example	`REJECTINFEASIBLEJOBS TRUE` `JOBREJECTPOLICY      MAIL,CANCEL` *Any job with zero feasible nodes for execution will be rejected.*

REJECTNEGPRIOJOBS	
Format	*<BOOLEAN>*
Default	*TRUE*
Description	If enabled, the scheduler will refuse to start any job with a negative priority. See Job Priority Overview and ENABLENEGJOBPRIORITY for more information.
Example	`ENABLENEGJOBPRIORITY TRUE` `REJECTNEGPRIOJOBS    TRUE` *Any job with a priority less than zero will be rejected.*

REMAPCLASS

Format	<ClassID>
Default	---
Description	Specifies which class/queue will be remapped based on the processors, nodes, and node features requested and the resource limits of each class. See Remap Class Overview for more information. In order to use **REMAPCLASS**, you must specify a **DEFAULTCLASS**.
Example	`RMCFG[internal]  DEFAULTCLASS=batch` `REMAPCLASS       batch` `CLASSCFG[small]  MAX.PROC=2` `CLASSCFG[medium] MAX.PROC=16` `CLASSCFG[large]  MAX.PROC=1024` *Class batch will be remapped based on the number of processors requested.*

REMAPCLASSLIST

Format	Comma delimited list of class names
Default	---
Description	Specifies the order in which classes will be searched when attempting to remap a class. Only classes included in the list will be searched and Moab will select the first class with matches. **Note**: If no **REMAPCLASSLIST** is specified, Moab will search all classes and will search them in the order they are discovered. See Remap Class Overview for more information.
Example	`RMCFG[internal] DEFAULTCLASS=batch` `REMAPCLASS      batch` `REMAPCLASSLIST  short,medium,long` *Class batch will be re-mapped to one of the listed classes.*

REMOTEFAILTRANSIENT

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Only applicable to Moab configurations with multiple resource managers able to run jobs (such as in a grid environment). When Moab attempts to migrate a job to one of these resource managers, a remote failure may occur. For example, a destination peer in a grid that has an error accepting a job results in a remote error, and the job is rejected. **REMOTEFAILTRANSIENT** controls how Moab reacts to remote errors. By default, Moab considers such an error permanent and does not try to migrate the same job to that resource manager again. If **REMOTEFAILTRANSIENT** is set to *TRUE*, then Moab considers such an error as transient and will not exclude the erring resource manager in future migration attempts.
Example	`REMOTEFAILTRANSIENT    TRUE`

REMOVETRIGOUTPUTFILES

Format	*<BOOLEAN>*
Default	*FALSE*
Description	When Moab launches external trigger actions, the standard output and error of those trigger actions are redirected to files located in Moab's spool directory. By default, these files are cleaned every 24 hours. (Files older than 24 hours are removed.) If, however, you wish to have Moab immediately remove the spool files after they are no longer needed, set RemoveTrigOutputFiles to *TRUE*.
Example	`REMOVETRIGOUTPUTFILES  TRUE`

RESCAP	
Format	*<DOUBLE>*
Default	*0* (NO CAP)
Description	Specifies the maximum allowed absolute pre-weighted job resource priority factor.
Example	`RESCAP 1000` *The total resource priority factor component of a job will be bound by +/- 1000*

RESERVATIONDEPTH[X]	
Format	*<INTEGER>*
Default	*1*
Description	Specifies the number of priority reservations which are allowed in the associated reservation bucket. **Note**: The array index, *X*, is the bucket label and can be any string up to 64 characters. This label should be synchronized with the RESERVATIONQOSLIST parameter. See Reservation Policies.
Example	`RESERVATIONDEPTH[bigmem]    4` `RESERVATIONQOSLIST[bigmem] special,fast,joshua` *Jobs with QOS's of special, fast, or joshua can have a cumulative total of up to 4 priority reservations.*

RESERVATIONPOLICY	
Format	One of the following: *CURRENTHIGHEST*, *HIGHEST*, *NEVER*
Default	*CURRENTHIGHEST*
Description	Specifies how Moab reservations will be handled. (See also RESERVATIONDEPTH) See Reservation Policies.
Example	`RESERVATIONPOLICY          CURRENTHIGHEST` `RESERVATIONDEPTH[DEFAULT]  2` *Moab will maintain reservations for only the 2 currently highest priority jobs.*

RESERVATIONQOSLIST[X]	
Format	One or more QOS values or *[ALL]*
Default	*[ALL]*
Description	Specifies which QOS credentials have access to the associated reservation bucket. **Note**: The array index, *X*, is the bucket label and can be any string up to 64 characters. This label should be synchronized with the RESERVATIONDEPTH parameter. See Reservation Policies.
Example	`RESERVATIONDEPTH[big]   4` `RESERVATIONQOSLIST[big] hi,low,med` *Jobs with QOS's of hi, low, or med can have a cumulative total of up to 4 priority reservations.*

RESERVATIONRETRYTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*60 seconds*
Description	Period of time Moab will continue to attempt to allocate resources to start a job after the time resources should be made available. This parameter takes into account resource manager node state race conditions, nodes with residual high load, network glitches, etc. (For related information, see Reservation Policies, DEFERSTARTCOUNT, DEFERTIME, NODEFAILURERESERVETIME, JOBRETRYTIME, and GUARANTEEDPREEMPTION.)
Example	`RESERVATIONRETRYTIME   00:05:00` *Moab will try for up to 5 minutes to maintain immediate reservations if the reservations are blocked due to node state, network, or batch system based race conditions.*

RESOURCELIMITMULTIPLIER[<PARID>]	
Format	*<RESOURCE>:<MULTIPLIER>[,...]* Where *<RESOURCE>* is one of the following: **NODE**, **PROC**, **JOBPROC**, **MEM**, **JOBMEM**, **SWAP**, **DISK**, or **WALLTIME**
Default	*1.0*
Description	If set to less than one, then the hard limit will be the specified limit and the soft limit will be the specified limit multiplied by the multiplier. If set to a value greater than one, then the specified limit will be the soft limit and the hard limit will be the specified limit multiplied by the multiplier. See Usage-based Limits.
Example	`RESOURCELIMITMULTIPLER  PROC:1.1,MEM:2.0` *Sets hard limit for **PROC** at 1.1 times the **PROC** soft limit, and the hard limit of **MEM** to 2.0 times the MEM soft limit.*

RESOURCELIMITPOLICY	
Format	*<RESOURCE>:[<SPOLICY>,]<HPOLICY> :[<SACTION>,]<HACTION> [:[<SVIOLATIONTIME>,]<HVIOLATIONTIME>]...* Where **RESOURCE** is one of *CPUTIME*, *DISK*, *JOBMEM*, *JOBPROC*, *MEM*, *MINJOBPROC*, *NETWORK*, *PROC*, *SWAP*, or *WALLTIME* where ***POLICY** is one of *ALWAYS*, *EXTENDEDVIOLATION*, or *BLOCKEDWORKLOADONLY* and where ***ACTION** is one of *CANCEL*, *CHECKPOINT*, *NOTIFY*, *REQUEUE*, *SIGNAL*, or *SUSPEND*.
Default	No limit enforcement.
Description	Specifies how the scheduler should handle jobs which utilize more resources than they request. See Usage-based Limits.
Example	`RESOURCELIMITPOLICY  MEM:ALWAYS,BLOCKEDWORKLOADONLY:REQUEUE,CANCEL` *Moab will cancel all jobs which exceed their requested memory limits.*

RESTARTINTERVAL	
Format	*[[[DD:]HH:]MM:]SS*
Default	---
Description	Causes Moab daemon to recycle/restart when the given interval of time has transpired.
Example	`RESTARTINTERVAL  20:00:00` *Moab daemon will automatically restart every 20 hours.*

RESOURCEQUERYDEPTH	
Format	*<INTEGER>*
Default	*3*
Description	Maximum number of options which will be returned in response to an mshow -a resource query.
Example	`RESOURCEQUERYDEPTH  1` *The mshow -a command will return at most 1 valid collection of resources.*

RESWEIGHT	
Format	*<INTEGER>*
Default	*1*
Description	All resource priority components are multiplied by this value before being added to the total job priority. See Job Prioritization.
Example	`RESWEIGHT  5` `MEMWEIGHT  10` `PROCWEIGHT 100` `SWAPWEIGHT 0` `RESCAP     2000` *The job priority resource factor will be calculated as MIN(2000,5 * (10 * JobMemory + 100 * JobProc)).*

RMCFG	
Format	One or more key-value pairs as described in the Resource Manager Configuration Overview
Default	---
Description	Specifies the interface and policy configuration for the scheduler-resource manager interface. Described in detail in the Resource Manager Configuration Overview.
Example	`RMCFG[TORQUE3] TYPE=PBS` *The PBS server will be used for resource management.*

RMMSGIGNORE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether or not Moab should adjust node state based on generic resource manager failure messages. See **Compute Node Health Check** on page 2328 for more info. For green or ONDEMAND computing, RMMSGIGNORE must be set to *TRUE* to prevent Moab HPC Suite from powering off a down node.
Example	`RMMSGIGNORE TRUE` *Moab will load and report resource manager failure messages but will not adjust node state as a result of them.*

RMPOLLINTERVAL	
Format	*[<MINPOLLTIME>,]<MAXPOLLTIME>* where poll time is specified as *[[[DD:]HH:]MM:]SS*
Default	*0,30*
Description	Specifies the interval between RM polls. The poll interval will be no less than MINPOLLTIME and no more than MAXPOLLTIME. If you specify a single value, Moab interprets the value as the MAXPOLLTIME with a MINPOLLTIME of 0. If you use TORQUE as your resource manager, prevent communication errors by giving **tcp_timeout** on page 2434 at least twice the value of the Moab **RMPOLLINTERVAL**.
Example	`RMPOLLINTERVAL 30,45` *Moab will refresh its resource manager information between a minimum of 30 seconds and a maximum of 45 seconds.* ***Note****: This parameter specifies the default global poll interval for all resource managers.*

RMRETRYTIMECAP	
Format	*[[[DD:]HH:]MM:]SS*
Default	*1:00:00*
Description	Moab attempts to contact RMs that are in state 'corrupt' (not down). If the attempt is unsuccessful, Moab tries again later. If the second attempt is unsuccessful, Moab increases the gap (the gap grows exponentially) between communication attempts. **RMRETRYTIMECAP** puts a cap on the length between connection attempts.
Example	`RMRETRYTIMECAP 24:00:00` *Moab stops increasing the gap between connection attempts once the retry gap reaches 24 hours.*

RSVLIMITPOLICY	
Format	*HARD* or *SOFT*
Default	---
Description	Specifies what limits should be enforced when creating reservations.
Example	`RSVLIMITPOLICY  HARD` *Moab will limit reservation creation based on the HARD limits configured.*

RSVNODEALLOCATIONPOLICY	
Format	One of the following: FIRSTAVAILABLE, LASTAVAILABLE, MINRESOURCE, CPULOAD, LOCAL, CONTIGUOUS, MAXBALANCE, or PRIORITY
Default	*LASTAVAILABLE*
Description	Specifies how Moab should allocate available resources to reservations.
Example	`RSVNODEALLOCATIONPOLICY MINRESOURCE` *Moab will apply the node allocation policy MINRESOURCE to all reservations by default.*

RSVNODEALLOCATIONPRIORITYF	
Format	User specified algorithm
Default	---
Description	When RSVNODEALLOCATIONPOLICY is set to *PRIORITY*, this parameter allows you to specify your own priority algorithm. The priority functions available are the same as the node priority functions.
Example	`RSVNODEALLOCATIONPOLICY PRIORITY` `RSVNODEALLOCATIONPRIORITYF 'SPEED + .01 * AMEM - 10 * JOBCOUNT'`

RSVPROFILE[X]	
Format	One or more of the following: *Allowed*: TRIGGERACL (ACCOUNTLIST, CLASSLIST, GROUPLIST, MAXTIME, QOSLIST, USERLIST) HostExp (HOSTLIST) Features (NODEFEATURES) FLAGS TASKCOUNT RSVACCESSLIST **Note**: Lists of more than one ACL value cannot be whitespace delimited. Such lists must be delimited with a comma, pipe, or colon. *Not allowed*: ACCESS CHARGEACCOUNT DAYS DEPTH ENDTIME OWNER PARTITION PERIOD **PRIORITY** RESOURCES STARTTIME TPN
Default	---
Description	Specifies attributes of a reservation profile using syntax similar to that for specifying a standing reservation. See Using Reservation Profiles for details.
Example	`RSVPROFILE[fast] USERLIST=john,steve` `RSVPROFILE[fast] QOSLIST=high,low` `RSVPROFILE[fast]` `TRIGGER=ETYPE=start,OFFSET=5:00,ATYPE=exec,ACTION="/opt/moab/rp.pl"` *Moab will create a reservation profile including trigger and ACL information.*

RSVSEARCHALGO

Format	*LONG* or *WIDE*
Default	*NONE*
Description	When Moab is determining when and where a job can run, it either searches for the most resources (WIDE) or the longest range of resources (LONG). In almost all cases, searching for the longest range is ideal and returns the soonest starttime. In some rare cases, however, a particular job may need to search for the most resources. In those cases sites can configure this parameter to prevent the starvation of large jobs that fail to hold onto their reservation starttimes. See the WIDERSVSEARCHALGO job flag. If this parameter is not set, it will be displayed in mschedctl -l as *NONE* but the algorithm that is used will be *LONG*.
Example	`RSVSEARCHALGO WIDE`

SCHEDCFG

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: FBSERVER, FLAGS, MAXJOBID, **MAXRECORDEDCJOBID**, MINJOBID, HTTPSERVERPORT, MODE, RECOVERYACTION, SERVER, or TRIGGER
Default	---
Description	Specifies scheduler policy and interface configuration. The **SERVER** attribute can also be set using the environment variable **$MOABSERVER**. Using this variable allows you to quickly change the Moab server that client commands will connect to. `> export MOABSERVER=cluster2:12221`
Example	`SCHEDCFG[zylem3] SERVER=geronimo.scc.com:3422 MODE=NORMAL` *Moab will execute in NORMAL mode on the host geronimo.scc.com.*

SERVERHOST

Description	This parameter is deprecated. See SCHEDCFG for replacement parameter.

SERVERMODE	
Description	This parameter is deprecated. See SCHEDCFG for replacement parameter.

SERVERNAME	
Format	*<STRING>*
Default	*<SERVERHOST>*
Description	Specifies the name the scheduler will use to refer to itself in communication with peer daemons. See SCHEDCFG for replacement parameter.
Example	`SERVERNAME moabA`

SERVERPORT	
Format	*<INTEGER>* (range: 1-64000)
Default	*40559*
Description	Port on which moab will open its user interface socket. See SCHEDCFG for replacement parameter.
Example	`SERVERPORT 30003` *Moab will listen for client socket connections on port 30003.*

SERVERSUBMITFILTER	
Format	*<PATH>*
Default	---
Description	Specifies the location of a global job submit filter script. When you configure a global job submit filter, Moab executes it on the head node and uses it to filter every job submission it receives. See **Server-based submit filter** on page 309 for more information about job submit filters.

SERVERSUBMITFILTER

Example	`SERVERSUBMITFILTER /opt/moab/scripts/globalfilter.pl` *Moab uses /opt/moab/scripts/globalfilter.pl to filter every job submitted to Moab.*

SERVICEWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the service component weight associated with the service factors. See Service (SERV) Factor for more information.
Example	`SERVICEWEIGHT 2`

SHOWMIGRATEDJOBSASIDLE

Format	*<BOOLEAN>*
Default	*FALSE*
Description	By default, migrated jobs in the grid will show as blocked. This is to prevent jobs from counting against the idle policies of multiple clusters rather than just the cluster to which the job was migrated.
Example	`SHOWMIGRATEDJOBSASIDLE TRUE` *When set to TRUE, migrated jobs will show as idle and will count against the idle policies of the cluster showing the job as migrated.*

SIMAUTOSHUTDOWN	
Format	*<BOOLEAN>*
Default	*TRUE*
Description	If *TRUE*, the scheduler will end simulations when the active queue and idle queue become empty.
Example	`SIMAUTOSHUTDOWN TRUE` *The simulation will end as soon as there are no jobs running and no idle jobs which could run.*

SIMINITIALQUEUEDEPTH	
Format	*<INTEGER>*
Default	*16*
Description	Specifies how many jobs the simulator will initially place in the idle job queue (see Simulation Overview).
Example	`SCHEDCFG[sim1]          MODE=SIMULATION` `SIMINITIALQUEUEDEPTH    64` `SIMJOBSUBMISSIONPOLICY CONSTANTJOBDEPTH` *Moab will initially place 64 idle jobs in the queue and, because of the specified queue policy, will attempt to maintain this many jobs in the idle queue throughout the duration of the simulation.*

SIMJOBSUBMISSIONPOLICY

Format	One of the following: *NORMAL*, *CONSTANTJOBDEPTH*, *CONSTANTPSDEPTH*, or REPLAY
Default	*CONSTANTJOBDEPTH*
Description	Specifies how the simulator will submit new jobs into the idle queue. *NORMAL* mode causes jobs to be submitted at the time recorded in the workload trace file, *CONSTANTJOBDEPTH* and *CONSTANTPSDEPTH* attempt to maintain an idle queue of SIMINITIALQUEUEDEPTH jobs and proc-seconds respectively. *REPLAY* will force jobs to execute at the exactly the time specified in the simulation job trace file. This mode is most often used to generate detailed profile statistics for analysis in Moab Cluster Manager (see Simulation Overview).
Example	`SIMJOBSUBMISSIONPOLICY NORMAL` *Moab will submit jobs with the relative time distribution specified in the workload trace file.*

SIMPURGEBLOCKEDJOBS

Format	*<BOOLEAN>*
Default	*TRUE*
Description	Specifies whether Moab should remove jobs which can never execute (see Simulation Overview).
Example	`SIMPURGEBLOCKEDJOBS  FALSE`

SIMRMRANDOMDELAY	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the random delay added to the RM command base delay accumulated when making any resource manager call in simulation mode.
Example	`SIMRMRANDOMDELAY 5` *Moab will add a random delay of between 0 and 5 seconds to the simulated time delay of all RM calls.*

SIMSTARTTIME	
Format	*[HH[:MM[:SS]]][_MO[/DD[/YY]]]*
Default	---
Description	Specifies the time when the simulation starts.
Example	`SIMSTARTTIME 00:00:00_01/01/00` *Moab will set its clock to January 1, 2000 at 12:00:00 in the morning before starting the simulation*

SIMSTOPTIME	
Format	*[HH[:MM[:SS]]][_MO[/DD[/YY]]]*
Default	---
Description	Specifies the time when the simulation should pause.
Example	`SIMSTOPTIME 00:00:00_01/01/04` *Moab will stop scheduling when its internal simulation time reaches January 1, 2004.*

SIMWORKLOADTRACEFILE	
Format	*<STRING>*
Default	*Traces/workload.trace*
Description	Specifies the file from which moab will obtain job information when running in simulation mode. Moab will attempt to locate the file relative to *<MOABHOMEDIR>* unless specified as an absolute path. See Simulation Overview and Workload Accounting Records.
Example	`SIMWORKLOADTRACEFILE traces/jobs.2` *Moab will obtain job traces when running in simulation mode from the* `<MOABHOMEDIR>/traces/jobs.2` *file.*

SPOOLDIR	
Format	*<STRING>*
Default	---
Description	Specifies the directory for temporary spool files created by Moab while submitting a job to the RM.
Example	`SPOOLDIR /tmp/moab/spool`

SPOOLDIRKEEPTIME	
Format	*<INTEGER>* (seconds) or [[[DD:]HH:]MM:]SS
Default	---
Description	Specifies the interval to delete spool files and other temporary files that have been left in the spool directory.
Example	`SPOOLDIRKEEPTIME 4:00:00`

SPVIOLATIONWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight to be applied to a job which violates soft usage limit policies (see **Service (SERVICE) Component** on page 388).
Example	`SPVIOLATIONWEIGHT 5000`

SRCFG[X]

Format	One or more of the following *<ATTR>=<VALUE>* pairs **ACCESS**, ACCOUNTLIST, **CHARGE** on page 468, CHARGEACCOUNT, CHARGEUSER, **CLASSLIST**, CLUSTERLIST, COMMENT, DAYS, **DEPTH**, DISABLE, ENDTIME, FLAGS, **GROUPLIST**, HOSTLIST, **JOBATTRLIST**, **MAXTIME**, **NODEFEATURES**, OWNER, **PARTITION**, PERIOD, **PRIORITY**, **QOSLIST**, REQUIREDTPN, RESOURCES, ROLLBACKOFFSET, RSVACCESSLIST, RSVGROUP, STARTTIME, TASKCOUNT, **TIMELIMIT**, **TPN**, **TRIGGER**, or USERLIST **Note:HOSTLIST** and **ACL** list values must be comma delimited. For example: **HOSTLIST**=*nodeA,nodeB*
Default	---
Description	Specifies attributes of a standing reservation. See Managing Reservations for details.
Example	`SRCFG[fast] STARTTIME=9:00:00 ENDTIME=15:00:00` `SRCFG[fast] HOSTLIST=node0[1-4]$` `SRCFG[fast] QOSLIST=high,low` *Moab will create a standing reservation running from 9:00 AM to 3:00 PM on nodes 1 through 4 accessible by jobs with QOS high or low.*

STARTCOUNTCAP	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the max weighted value allowed from the startcount subfactor when determining a job's priority (see Priority Factors for more information).
Example	`STARTCOUNTWEIGHT 5000` `STARTCOUNTCAP    30000`

STARTCOUNTWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight to be applied to a job's startcount when determining a job's priority (see Priority Factors for more information).
Example	`STARTCOUNTWEIGHT 5000`

STATDIR	
Format	*<STRING>*
Default	*stats*
Description	Specifies the directory in which Moab statistics will be maintained.
Example	`STATDIR /var/adm/moab/stats`

STATPROCMAX	
Format	*<INTEGER>*
Default	*1*
Description	Specifies the maximum number of processors requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via mschedctl -m or changeparam while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.
Example	`STATPROCMAX     256` `STATPROCSTEPCOUNT 4` `STATPROCSTEPSIZE  4` *Each matrix output will display data in rows for jobs requesting between 4 and 256 processors.* A **NONE** in services will still allow users to run showq and checkjob on their own jobs.

STATPROCMIN

Format	*<INTEGER>*
Default	*1*
Description	Specifies the minimum number of processors requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.
Example	`STATPROCMIN 4` `STATPROCSTEPCOUNT 4` `STATPROCSTEPSIZE 4` *Each matrix output will display data in rows for jobs requesting between 4 and 256 processors.* A **NONE** in services will still allow users to run showq and checkjob on their own jobs.

STATPROCSTEPCOUNT

Format	*<INTEGER>*
Default	*5*
Description	Specifies the number of rows of processors requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.
Example	`STATPROCMIN 4` `STATPROCSTEPCOUNT 4` `STATPROCSTEPSIZE 4` *Each matrix output will display data in rows for jobs requesting between 4 and 256 processors.*

STATPROCSTEPSIZE

Format	*<INTEGER>*
Default	*4*
Description	Specifies the processor count multiplier for rows of processors requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.
Example	`STATPROCMIN 4` `STATPROCSTEPCOUNT 4` `STATPROCSTEPSIZE 4` *Each matrix output will display data in rows for jobs requesting between 4 and 256 processors.*

STATTIMEMAX

Format	*[[DD:]HH:]MM:]SS*
Default	*00:15:00*
Description	Specifies the maximum amount of time requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.
Example	`STATTIMEMAX 02:08:00` `STATTIMESTEPCOUNT 4` `STATTIMESTEPSIZE 4` *Each matrix output will display data in columns for jobs requesting between 2 and 128 minutes.*

STATTIMEMIN

Format	*[[DD:]HH:]MM:]SS*
Default	*00:15:00*
Description	Specifies the minimum amount of time requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.
Example	`STATTIMEMIN       00:02:00` `STATTIMESTEPCOUNT 4` `STATTIMESTEPSIZE  4` *Each matrix output will display data in columns for jobs requesting between 2 and 128 minutes.*

STATTIMESTEPCOUNT

Format	*<INTEGER>*
Default	*6*
Description	Specifies the number of columns of time requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.
Example	`STATTIMEMIN       00:02:00` `STATTIMESTEPCOUNT 4` `STATTIMESTEPSIZE  4` *Each matrix output will display data in columns for jobs requesting between 2 and 128 minutes.*

STATTIMESTEPSIZE	
Format	*<INTEGER>*
Default	*4*
Description	Specifies the time multiplier for columns of time requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.
Example	`STATTIMEMIN 00:02:00` `STATTIMESTEPCOUNT 4` `STATTIMESTEPSIZE 4` *Each matrix output will display data in columns for jobs requesting between 2 and 128 minutes.*

STOPITERATION	
Format	*<INTEGER>*
Default	*-1* (don't stop)
Description	Specifies which scheduling iteration Moab will stop and wait for a command to resume scheduling.
Example	`STOPITERATION 10` *Moab should stop after iteration 10 of scheduling and wait for administrator commands.*

STOREJOBSUBMISSION	
Format	*<BOOLEAN>*
Default	---
Description	When set to *TRUE*, specifies that Moab will save a job's submit arguments and script to `$MOABHOMEDIR/stats/jobarchive/jobNumber`. If you use TORQUE as your resource manager, you can configure it to store completed job information, and it will store the same information returned by the qstat -f command. For more information, see **Job Logging** on page 2315 in the TORQUE documentation.
Example	`STOREJOBSUBMISSION TRUE`

STRICTPROTOCOLCHECK	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies how Moab reacts to differences in XML protocols when communicating with other Moab peers. If set to *TRUE*, Moab will reject any communication that does not strictly conform to the expected protocol. If set to *FALSE* (the default), Moab will not reject XML that has extra or unknown attributes.
Example	`STRICTPROTOCOLCHECK TRUE` *Moab will reject any XML communication that does not strictly conform to the expected protocol definition.*

SUBMITENVFILELOCATION	
Format	*FILE* or *PIPE*
Default	---
Description	If set to *FILE*, these behaviors are expected: • The environment file is owned by a user with 600 permissions. • Moab writes the environment variables ('\0' delimited) to a random file in Moab's spool directory. • Moab adds the `--export-file=<path_to_file>` on the sbatch command line. • Moab deletes the file after the job completes. If set to *PIPE*, these behaviors are expected: • Moab creates a pipe and passes the read end of the pipe's file descriptor to sbatch. • Moab's parent process writes the environment ('\0' delimited) into the write end of the pipe. Adaptive Computing recommends that you configure this parameter for a more secure environment.
Example	`SUBMITENVFILELOCATION PIPE`

SUBMITFILTER	
Format	*<STRING>*
Default	---
Description	Specifies the directory of a given submit filter script.
Example	`SUBMITFILTER /home/submitfilter/filter.pl`

SUBMITHOSTS	
Format	space delimited list of host names
Default	---
Description	If specified, **SUBMITHOSTS** specifies an explicit list of hosts where jobs can be submitted.
Example	`SUBMITHOSTS hostA hostB`

SUSPENDRESOURCES[<PARID>]

Format	*<RESOURCE>[,...]* Where *<RESOURCE>* is one of the following: *NODE*, *PROC*, *MEM*, *SWAP*, *DISK*
Default	---
Description	List of resources to dedicate while a job is suspended (available in Moab version 4.5.1 and higher).
Example	`SUSPENDRESOURCES[base]  MEM,SWAP,DISK` *While a job is suspended in partition base, the memory, swap and disk for that job will remain dedicated to the job.*

SYSCFG

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: **PRIORITY**, **FSTARGET**, **QLIST**, **QDEF**, **PLIST**, **FLAGS**, or a fairness policy specification.
Default	---
Description	Specifies system-wide default attributes. See the Attribute/Flag Overview for more information.
Example	`SYSCFG PLIST=Partition1 QDEF=highprio` *By default, all jobs will have access to partition Partition1 and will use the QOS highprio.*

SWAPWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight assigned to the virtual memory request of a job.
Example	`SWAPWEIGHT 10`

SYSTEMMAXPROCPERJOB	
Format	*<INTEGER>*
Default	*-1* (NO LIMIT)
Description	Specifies the maximum number of processors that can be requested by any single job.
Example	`SYSTEMMAXPROCPERJOB 256` *Moab will reject jobs requesting more than 256 processors.*

SYSTEMMAXPROCSECONDPERJOB	
Format	*<INTEGER>*
Default	*-1* (NO LIMIT)
Description	Specifies the maximum number of proc-seconds that can be requested by any single job.
Example	`SYSTEMMAXJOBPROCSECOND 86400` *Moab will reject jobs requesting more than 86400 procs seconds. i.e., 64 processors * 30 minutes will be rejected, while a 2 processor * 12 hour job will be allowed to run.*

SYSTEMMAXJOBWALLTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*-1* (NO LIMIT)
Description	Specifies the maximum amount of wallclock time that can be requested by any single job.
Example	`SYSTEMMAXJOBWALLTIME 1:00:00:00` *Moab will reject jobs requesting more than 1 day of walltime.*

TARGETQUEUETIMEWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to the time remaining until the queuetime is reached.
Example	`TARGETQUEUETIMEWEIGHT 10`

TARGETWEIGHT	
Format	*<INTEGER>*
Default	*1*
Description	Specifies the weight to be applied to a job's queuetime and expansion factor target components (see Job Prioritization).
Example	`TARGETWEIGHT 1000`

TARGETXFACTORWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to the distance to the target expansion factor.
Example	`TARGETXFACTORWEIGHT  10`

TASKDISTRIBUTIONPOLICY	
Format	One of *DEFAULT*, *PACK*, *RR* (round-robin)
Default	---
Description	Specifies how job tasks should be mapped to allocated resources. *DEFAULT* allows the resource manager to determine how the tasks are placed on the nodes. When *PACK* is used, a node is filled up with tasks before the next node is used. When *RR* is used, tasks are cycled through nodes, one task at a time, until there are no more tasks. See Task Distribution Overview for more information.
Example	`TASKDISTRIBUTIONPOLICY DEFAULT` *Moab should use standard task distribution algorithms.*

THREADPOOLSIZE	
Format	*<INTEGER>*
Default	*2X number of core processors* (MAX: 64)
Description	Governs the number of threads used when processing job scheduling. Scalability and performance may improve with multi-threading; to throttle, limit the number of threads used.
Example	`THREADPOOLSIZE 10`

TOOLSDIR	
Format	*<STRING>*
Default	*tools*
Description	Specifies the directory in which Moab tools will be maintained (commonly used in conjunction with Native Resource Managers, and Triggers).
Example	`TOOLSDIR /var/adm/moab/tools`

TRAPFUNCTION	
Format	*<STRING>*
Default	---
Description	Specifies the functions to be trapped.
Example	`TRAPFUNCTION UpdateNodeUtilization|GetNodeSResTime`

TRAPJOB	
Format	*<STRING>*
Default	---
Description	Specifies the jobs to be trapped.
Example	`TRAPJOB pros23.0023.0`

TRAPNODE	
Format	*<STRING>*
Default	---
Description	Specifies the nodes to be trapped.
Example	`TRAPNODE node001|node004|node005`

TRAPRES	
Format	*<STRING>*
Default	---
Description	Specifies the reservations to be trapped.
Example	`TRAPRES interactive.0.1`

TRIGCHECKTIME	
Format	*<INTEGER>* (milliseconds)
Default	*2000*
Description	Each scheduling iteration, Moab will have a period of time where it handles commands and other UI requests. This time period is controlled by RMPOLLINTERVAL. During this time period, known as the UI phase, Moab will periodically evaluate triggers. Usually this only takes a fraction of a second, but if the number of triggers are large it could take up substantially more time (up to several seconds). While Moab is evaluating triggers, it doesn't respond to UI commands. This makes Moab feel sluggish and unresponsive. To remedy this, use the parameter **TRIGCHECKTIME**. This parameter tells Moab to only spend up to X milliseconds processing triggers during the UI phase. After *X* milliseconds has gone by, Moab will pause the evaluating of triggers, handle any pending UI events, and then restart the trigger evaluations where it last left off.
Example	`TRIGCHECKTIME 4000`

TRIGEVALLIMIT

Format	*<INTEGER>*
Default	*1*
Description	Each scheduling iteration, Moab will have a period of time where it handles commands and other UI requests. This time period is controlled by RMPOLLINTERVAL. During this time period, known as the UI phase, Moab will periodically evaluate triggers. The number of times Moab evaluates all triggers in the system is controlled by the **TRIGEVALLIMIT** parameter. By default, this is set to *1*. This means that Moab will evaluate all triggers at most once during the UI phase. Moab will not leave the UI phase and start other scheduling tasks until ALL triggers are evaluated at least one time. If TrigEvalLimit is set to *5*, then Moab will wait until all triggers are evaluated five times.
Example	`TRIGEVALLIMIT   3`

UJOBWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Weight assigned by jobs per user. *-1* will reduce priority by number of active jobs owned by user.
Example	`UJOBWEIGHT 10`

UMASK

Format	*<INTEGER>*
Default	*0022* (octal) (produces 0644 permissions)
Description	Specifies the file permission mask to use when creating new fairshare, stats, and event files. See the `umask` man page for more details.
Example	`UMASK 0127` *Create statistics and event files which are 'read-write' by owner and 'read' by group only.*

UNSUPPORTEDDEPENDENCIES

Format	Comma delimited string
Default	---
Description	Specifies dependencies that are not supported and should not be accepted by job submissions. A maximum of 30 dependencies is supported.
Example	`# moab.cfg` `UNSUPPORTEDDEPENDENCIES before,beforeok,beforenotok,on` `> msub -l depend=before:105 cmd.sh` `ERROR: cannot submit job - error in extension string`

UPROCWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Weight assigned by processors per user. *-1* will reduce priority by number of active procs owned by user.
Example	`UPROCWEIGHT 10`

USAGECONSUMEDWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to per job processor second consumption.
Example	`USAGECONSUMEDWEIGHT 10`

USAGEEXECUTIONTIMEWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight assigned to the total job execution time (measured in seconds since job start). See Preemption Overview.
Example	`USAGEEXECUTIONTIMEWEIGHT 10`

USAGEPERCENTWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to total requested resources consumed.
Example	`USAGEPERCENTWEIGHT 5`

USAGEREMAININGWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to remaining usage.
Example	`USAGEREMAININGWEIGHT 10`

USAGEWEIGHT	
Format	*<INTEGER>*
Default	*1*
Description	Specifies the weight assigned to the percent and total job usage subfactors.
Example	`USAGEWEIGHT 100`

USEANYPARTITIONPRIO	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	The FSTREE data from the first feasible FSTREE will be used when determining a job's start priority, rather than having no FSTREE data considered. Do not set **USEANYPARTITIONPRIO** if you use per-partition scheduling. Doing so causes to schedule jobs to the first partition listed, even if nodes from another partition will be available sooner.
Example	`USEANYPARTITIONPRIO TRUE`

USECPRSVNODELIST	
Format	*<BOOLEAN>*
Default	*TRUE*
Description	Specifies whether Moab should use the checkpointed reservation node list when rebuilding reservations on startup. If this is not used then Moab will use the reservation's specified host expression during rebuilding.
Example	`USECPRSVNODELIST FALSE`

USEDATABASE	
Format	INTERNAL
Default	-
Description	Specifies whether Moab should store profile statistics, checkpoint information, and event information in an integrated database. See Layout of Scheduler Components with Integrated Database Enabled for more information.
Example	`USEDATABASE INTERNAL`

USEJOBREGEX	
Format	BOOLEAN
Default	FALSE
Description	Specifies whether mjobctl supports regular expressions.
Example	`USEJOBREGEX TRUE` `[user@linux]$ mjobctl -c 8[1-3]` `job '81' cancelled` `job '82' cancelled` `job '83' cancelled`

USEMOABCTIME	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	When Moab finds new jobs on the resource manager, it creates a job inside of Moab for each job in the resource manager. By default, when Moab creates a new job, it uses the time the job was submitted to the resource manager to calculate how long the job has been in the queue (Moab processing time - job creation in resource manager), which is then used in determining the job's priority. In a system where more jobs are submitted to a resource manager than Moab can handle in one iteration, there is the possibility of jobs running out of order. For example, two jobs are both submitted at time 5. The first submitted job is processed first at time 6. So the first job's effective queue duration would be 1 (6-5). On the next iteration, the second job is processed at time 8. So the second job's effective queue duration would be 3 (8-5), indicating that it has been in the queue longer than the other job. Since the later job has a higher effective queue duration it will get a higher priority and could be scheduled to run before earlier submitted jobs. Setting **USEMOABCTIME** to *TRUE* tells Moab to use the creation time of the job in Moab rather than the creation time in the resource manager. This corrects the possible problem of having later submitted jobs having higher priorities and starting before earlier submitted jobs.
Example	`USEMOABCTIME TRUE`

USEMOABJOBID	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether to use the Moab job ID, or the resource manager's job ID.
Example	`USEMOABJOBID TRUE`

USERCFG[<USERID>]

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: General Credential Flags, **CDEF**, **DEFAULT.TPN**, **DEFAULT.WCLIMIT**, **EMAILADDRESS**, ENABLEPROFILING, **FSCAP**, **FSTARGET**, JOBFLAGS, **MAX.WCLIMIT**, **QLIST**, **QDEF**, **NOEMAIL**, **OVERRUN**, **PLIST**, PRIORITY, or a usage limit.
Default	---
Description	Specifies user specific attributes. For general user attribute information, See the Credential Overview. For a description of legal flag values, see flag overview.
Example	`USERCFG[john] MAXJOB=50 QDEF=highprio` `USERCFG[john] EMAILADDRESS=john@company.com` *Up to 50 jobs submitted under the user ID john will be allowed to execute simultaneously and will be assigned the QOS highprio.*

USERPRIOCAP

Format	*<INTEGER>*
Default	---
Description	Specifies the priority cap to be applied to the user specified job priority factor. Under Moab, only negative user priorities may be specified. See Credential (Service) Factor.
Example	`USERPRIOWEIGHT 10` `USERPRIOCAP    -10000`

USERPRIOWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight to be applied to the user specified job priority. Under Moab, only negative user priorities may be specified. If this weight is set, users may reduce the priority of some of their jobs to allow other jobs to run earlier. See Credential (Service) Factor and User Selectable Prioritization.
Example	`USERPRIOWEIGHT 10`

USERWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the weight to be applied to the user priority of each job. See Credential (CRED) Factor.
Example	`USERWEIGHT 10`

USESYSLOG

Format	*<BOOLEAN>[<FACILITY>]*
Default	*FALSE:daemon*
Description	Specifies whether or not the scheduler will report key events to the system syslog facility. If the *<FACILITY>* is specified, Moab will report events to this syslog facility. See Logging Facilities for more information.
Example	`USESYSLOG TRUE:local3` *Moab will report key events, commands, and failures to syslog using the local3 facility.*

USESYSTEMQUEUETIME	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether or not job prioritization should be based on the time the job has been eligible to run, i.e., idle and meets all fairness policies (*TRUE*) or the time the job has been idle (*FALSE*). See Priority Factors for more info. **Note**: This parameter has been superseded by the JOBPRIOACCRUALPOLICY parameter.
Example	`USESYSTEMQUEUETIME FALSE` *The queuetime and expansion factor components of a job's priority will be calculated based on the length of time the job has been in the idle state.*

USEUSERHASH	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Enables searching of the user buffer using the user hash key instead of doing sequential searches of the user buffer.
Example	`USEUSERHASH TRUE`

VMCALCULATELOADBYVMSUM	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	When false, vmmigrate using overcommits uses the CPU load from the node to determine if VM's need to be migrated off the hypervisor. When true, overcommit vmmigrates calculates the total node load using the total sum reported by each VM on the hypervisor.
Example	`VMCALCULATELOADBYVMSUM  TRUE`

VMCPURGETIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*5:00*
Description	When a VM completes, Moab stores it in a completed VM table for the specified amount of time. This prevents it from starting again if an RM reports it late. It also prevents a user from creating a VM with the same ID for a certain amount of time. The VM will remain in the completed VM table for more than the specified amount of time if **VMSTALETIME** is greater than **VMCPURGETIME**. Both parameters must expire before Moab will remove the VM from the table.
Example	`VMCPURGETIME 10:00` *Moab holds completed VMs for 10 minutes to prevent a late RM from reporting and restarting it.*

VMMIGRATETOZEROLOADNODES	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Allows VM migrations to occur to and from hypervisors that do not report a CPULoad or memory load.
Example	`VMMIGRATETOZEROLOADNODES      TRUE`

VMMIGRATETHROTTLE	
Format	*<INTEGER>*
Default	---
Description	Sets the maximum allowable 'VM migrate' jobs at any given time.
Example	`VMMIGRATETHROTTLE  20` *Only 20 VM migrate jobs are allowed in the system at any given time.*

VMMIGRATIONPOLICY	
Format	*<STRING>*; values include *CONSOLIDATION* and *OVERCOMMIT*
Default	*NONE*
Description	Choose only one of these values: • *CONSOLIDATION*- If the *CONSOLIDATION* flag is set, Moab consolidates VMs to allow nodes to go idle. This flag also ensures that no hypervisors are overloaded. • *OVERCOMMIT*- If the *OVERCOMMIT* flag is set, VMs to be migrated will be selected from overloaded hypervisors to bring them below the selected thresholds. This flag must be set for the VMOCTHRESHOLD parameter to function.
Example	`VMMIGRATIONPOLICY OVERCOMMIT`

VMMINOPDELAY	
Format	*[HH[:MM[:SS]*
Default	---
Description	The minimum time between automatic VM node operations, such as creating, modifying, and destroying VMs. May prevent thrashing.
Example	`VMMINOPDELAY 30`

VMOCTHRESHOLD	
Format	*MEM:<0-1>,PROCS:<0-1>,DISK:<0-1>,SWAP:<0-1>,GMETRIC:<metric>:value*
Default	---
Description	Percentage threshold at which Moab begins to migrate virtual machines to other nodes. VMMIGRATIONPOLICY must be set to OVERCOMMIT for this to occur.
Example	`NODECFG[DEFAULT]  VMOCTHRESHOLD=PROC:.7,MEM:.9,GMETRIC:mem_io:6000    # This is the default global policy` `NODECFG[node42]   VMOCTHRESHOLD=PROC:.2,MEM:.1,GMETRIC:mem_io:12000   # This is a node-specific policy for node42` *When a node surpasses .7 (70%) load of CPU or .9 (90%) of memory, Moab begins to migrate virtual machines to other nodes. When node42surpasses .2 (20%) load of CPU or .1 (10%) of memory, Moab begins to migrate virtual machines to other nodes.*

VMPROVISIONSTATUSREADYVALUE	
Format	*<INTEGER>*
Default	---
Description	Checks a VM for a special value or values (which Moab gets from the resource manager) and, based on the value, tells Moab that a VM was created..
Example	`VMProvisionStatusReadyValue 2` `VMProvisionStatusReadyValue 1-4,6,16`

VMSARESTATIC	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	When set to true, informs Moab that it can schedule under the assumption that no VMs will be migrated and no new VMs will be created, and disables Moab from scheduling any VM creations or migrations.
Example	`VMSARESTATIC TRUE`

VMSTALEACTION	
Format	One of the following: *IGNORE*, *CANCELTRACKINGJOB*, or *DESTROY*
Default	*IGNORE*
Description	Specifies the action that is applied to a stale VM, or a VM that the resource manager has not reported to Moab recently (see VMSTALETIME). • *IGNORE* (default) specifies that Moab will take no action. • *CANCELTRACKINGJOB* specifies that Moab will remove the tracking job for stale VMs, but will not remove the actual VM (not recommended). • *DESTROY* specifies that Moab destroys stale VMs. If you specify *DESTROY*, you must also set the ENABLEVMDESTROY parameter to TRUE.
Example	`VMSTALEACTION DESTROY`

VMSTALETIME	
Format	*[[HH:]MM:]SS*
Default	*10:00*
Description	Specifies the amount of time a VM must be unreported by any resource manager before it is considered "stale." To specify what happens with the VM after it has become stale, see VMSTALEACTION.
Example	`VMSTALETIME 5:00` *5 minutes must pass without a resource manager reporting a VM for it to be considered stale.*

VMSTORAGEMOUNTDIR	
Format	*<PATH>*
Default	---
Description	The specified path is used as the default location for storage mounts in all newly created VMs (created via the mvmctl command). This parameter defines the default storage mount directory if one is not specified.
Example	`VMSTORAGEMOUTDIR   /var/spool` *Moab uses /var/spool as a storage mount directory if a storage directory is not submitted (but additional storage is requested) at VM creation.*

VMTRACKING	
Format	*<STRING>*
Default	---
Description	When set to *TRUE*, VMTracking jobs are used to represent VMs in the queue.
Example	`VMTRACKING TRUE`

WALLTIMECAP	
Format	*<DOUBLE>*
Default	*0* (NO CAP)
Description	Specifies the maximum total pre-weighted absolute contribution to job priority which can be contributed by the walltime component. This value is specified as an absolute priority value, not as a percent.
Example	`WALLTIMECAP 10000` *Moab will bound a job's pre-weighted walltime priority component within the range +/- 10000.*

WALLTIMEWEIGHT

Format	*<INTEGER>*
Default	0
Description	Specifies the priority weight to be applied to the amount of walltime requested by a job (in seconds) (see Resource (RES) Factor).
Example	`RESWEIGHT      10` `WALLTIMEWEIGHT 100` *Increase the priority of longer duration jobs.*

WCACCURACYCAP

Format	*<DOUBLE>*
Default	0 (NO CAP)
Description	Specifies the maximum total pre-weighted absolute contribution to job priority which can be contributed by the wallclock accuracy component. This value is specified as an absolute priority value, not as a percent.
Example	`WCACCURACYCAP 10000` *Moab will bound a job's pre-weighted wallclock accuracy priority component within the range +/- 10000.*

WCACCURACYWEIGHT

Format	*<INTEGER>*
Default	0
Description	Specifies the priority weight to be applied to the job's historical user wallclock accuracy (range 0.0 to 1.0) (see Fairshare (FS) Factor).
Example	`FSWEIGHT         10` `WCACCURACYWEIGHT 100` *Favor jobs with good wallclock accuracies by giving them a priority increase.*

WCVIOLATIONACTION

Format	one of *CANCEL* or *PREEMPT*
Default	*CANCEL*
Description	Specifies the action to take when a job exceeds its wallclock limit. If set to *CANCEL*, the job will be terminated. If set to *PREEMPT*, the action defined by PREEMPTPOLICY parameter will be taken. See JOBMAXOVERRUN or Usage-based limits.
Example	`WCVIOLATIONACTION   PREEMPT` `PREEMPTPOLICY       REQUEUE` *Moab will requeue jobs which exceed their wallclock limit.*

WEBSERVICESURL

Format	*<URL>*
Default	---
Description	If specified, Moab sends data to Moab Web Services (MWS) to be stored in a database. This allows Moab to spend more cycles on scheduling instead of database interaction. The sending occurs via HTTP PUT.
Example	`WEBSERVICESURL http://mws-staging.ac:8080/mws/rm/moab/dump` *Moab sends data that needs to be stored in a database to the specified URL.*

WIKIEVENTS	
Format	*<BOOLEAN>*
Default	*TRUE*
Description	When set to true, Moab events are set to native wiki format (ATTR=VALUE pairs) to facilitate easier readability .
Example	`WIKIEVENTS TRUE` Moab events will generate output in the format of the following sample: `09:26:40 1288279600:5 job 58 JOBEND 58 REQUESTEDNC=1 REQUESTEDTC=3 UNAME=wightman GNAME=wightman WCLIMIT=60 STATE=Completed RCLASS=[batch:1] SUBMITTIME=1288279493 RMEMCMP=>= RDISKCMP=>= RFEATURES=[NONE] SYSTEMQUEUETIME=1288279493 TASKS=1 FLAGS=RESTARTABLE PARTITION=pbs DPROCS=1 ENDDATE=2140000000 TASKMAP=proxy,GLOBAL SRM=pbs EXITCODE=0 SID=2357 NODEALLOCATIONPOLICY=SHARED EFFECTIVEQUEUEDURATION=107`

XFACTORCAP	
Format	*<DOUBLE>*
Default	*0* (NO CAP)
Description	Specifies the maximum total pre-weighted absolute contribution to job priority which can be contributed by the expansion factor component. This value is specified as an absolute priority value, not as a percent.
Example	`XFACTORCAP 10000` *Moab will bound a job's pre-weighted XFactor priority component within the range +/- 10000.*

XFACTORWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight to be applied to a job's minimum expansion factor before it is added to the job's cumulative priority.
Example	`XFACTORWEIGHT 1000` *Moab will multiply a job's XFactor value by 1000 and then add this value to its total priority.*

XFMINWCLIMIT	
Format	*[[[DD:]HH:]MM:]SS*
Default	*-1* (NO LIMIT)
Description	Specifies the minimum job wallclock limit that will be considered in job expansion factor priority calculations.
Example	`XFMINWCLIMIT 0:01:00` *Jobs requesting less than 1 minute of wallclock time will be treated as if their wallclock limit was set to 1 minute when determining expansion factor for priority calculations.*

Appendix B: Multi-OS Provisioning

Introduction

Moab can dynamically provision compute machines to requested operating systems and power off compute machines when not in use. Moab can intelligently control xCAT and use its advanced system configuration mechanisms to adapt systems to current workload requirements. Moab communicates with xCAT using the Moab Service Manager (MSM). MSM is a translation utility that resides between Moab and xCAT and acts as aggregator and interpreter. The Moab Workload Manager will query MSM, which in turn queries xCAT, about system resources, configurations, images, and metrics. After learning about these resources from MSM, Moab then makes intelligent decisions about the best way to maximize system utilization.

In this model Moab gathers system information from two resource managers. The first is TORQUE, which handles the workload on the system; the second is MSM, which relays information gathered by xCAT. By leveraging these software packages, Moab intelligently adapts clusters to deliver on-site goals.

This document assumes that xCAT has been installed and configured. It describes the process of getting MSM and xCAT communicating, and it offers troubleshooting guidance for basic integration. This document offers a description for how to get Moab communicating with MSM and the final steps in verifying a complete software stack.

xCAT Configuration Requirements

Observe the following xCAT configuration requirements before installing MSM:

- Configure xCAT normally for your site.
  - Test the following commands to verify proper function:
    - **rpower**
    - **nodeset**
    - **makedhcp**
    - **makedns**
    - **nodestat**
    - **rvitals**
  - If MSM will run on a different machine than the one on which xCAT runs, install the xCAT client packages on that machine, and test the previously listed commands on that machine as well.
  - Configure and test all stateful/stateless images you intend to use.
- Configure xCAT to use either PostgreSQL or MySQL. Note that the default of SQLite may not function properly when MSM drives xCAT.

- PostgreSQL: See xCATSetupPostgreSQL.pdf for more information.
- MySQL: See xCAT2.SetupMySQL.pdf for more information.

You must have a valid Moab license file (`moab.lic`) with provisioning and green enabled. For information on acquiring an evaluation license, please contact info@adaptivecomputing.com.

MSM Installation

- Determine the installation directory (usually `/opt/moab/tools/msm`)
- Untar the MSM tarball into the specified directory (making it the MSM home directory, or *$MSMHOMEDIR*)
- Verify the required Perl modules and version are available

```
> perl -e 'use Storable 2.18'
> perl -MXML::Simple -e 'exit'
> perl -MProc::Daemon -e 'exit'
> perl -MDBD::SQLite -e 'exit'
```

Integrating MSM and xCAT

Copy the `x_msm` table schema to the xCAT schema directory:

```
> cp $MSMHOMEDIR/contrib/xcat/MSM.pm $XCATROOT/lib/perl/xCAT_schema
```

Restart xcatd and check the `x_msm` table is correctly created:

```
> service xcatd restart
```

```
> tabdump x_msm
```

Prepare xCAT images and ensure they provision correctly (see xCAT documentation)

Populate the `x_msm` table with your image definitions:

```
> tabedit x_msm

#flavorname,arch,profile,os,nodeset,features,vmoslist,hvtype,hvgroupname,vmgroupname,co
mments,disable
   "compute","x86_64","compute","centos5.3","netboot","torque",,,,,,
   "science","x86","compute","scientific_linux","netboot","torque",,,,,,
```

- **flavorname** - A user specified name for the image and settings; also an xCAT group name, nodes are added to this group when provisioned
- **arch** - Architecture as used by xCAT
- **profile** - Profile as used by xCAT
- **os** - Operating system as used by xCAT
- **nodeset** - One of netboot|install|statelite

- **features -** Names of xCAT groups that identify special hardware features ('torque' and 'paravirt' are special cases)
- **vmoslist -** Note: Not used. List of flavorname's this image may host as VMs (hypervisor images only)
- **hvtype -** Note: Not used. One of esx|xen|kvm (hypervisor images only)
- **hvgroupname -** Note: Not used. Name of xCAT group nodes will be added to when provisioned to this image
- **vmgroupname -** Note: Not used. Name of xCAT group VMs will be added to when hosted on a hypervisor of this image
- **comments -** User specified comments
- **disable -** Flag to temporarily disable use of this image

Ensure all xCAT group names in the x_msm table exist in the xCAT nodegroup table

```
> tabedit nodegroup
```

Edit as necessary to simulate the following example:

```
#groupname,grouptype,members,wherevals,comments,disable
"compute",,,,,
"esxi4",,,,,
"esxhv",,,,,
"esxvmmgt",,,,,
```

After making any necessary edits, run the following command:

```
> nodels compute,esxi4,esxhv,esxvmmgt
  # should complete without error, ok if doesn't return anything
```

MSM Configuration

Edit *$MSMHOMEDIR*/msm.cfg and configure the xCAT plug-in. Below is a generic example for use with TORQUE without virtualization. See the section on configuration parameters for a complete list of parameters and descriptions.

```
# MSM configuration options
RMCFG[msm]        PORT=24603
RMCFG[msm]        POLLINTERVAL=45
RMCFG[msm]        LOGFILE=/opt/moab/log/msm.log
RMCFG[msm]        LOGLEVEL=8
RMCFG[msm]        DEFAULTNODEAPP=xcat

# xCAT plugin specific options
APPCFG[xcat]      DESCRIPTION="xCAT plugin"
APPCFG[xcat]      MODULE=Moab::MSM::App::xCAT
APPCFG[xcat]      LOGLEVEL=3
APPCFG[xcat]      POLLINTERVAL=45
APPCFG[xcat]      TIMEOUT=3600
APPCFG[xcat]      _USEOPIDS=0
APPCFG[xcat]      _NODERANGE=moab,esxcompute
APPCFG[xcat]      _USESTATES=boot,netboot,install
APPCFG[xcat]      _LIMITCLUSTERQUERY=1
APPCFG[xcat]      _RPOWERTIMEOUT=120
APPCFG[xcat]      _DONODESTAT=1
APPCFG[xcat]      _REPORTNETADDR=1
APPCFG[xcat]      _CQXCATSESSIONS=4
```

Configuration Validation

Set up environment to manually call MSM commands:

```
# substitute appropriate value(s) for path(s)
  export MSMHOMEDIR=/opt/moab/tools/msm
  export MSMLIBDIR=/opt/moab/tools/msm
  export PATH=$PATH:/$MSMLIBDIR/contrib:$MSMLIBDIR/bin
```

Verify that MSM starts without errors:

```
> msmd
```

Verify that the expected nodes are listed, without errors, using the value of **_NODERANGE** from `msm.cfg`.

```
> nodels <_NODERANGE>
```

Verify that the expected nodes, are listed in the cluster query output from MSM:

```
> cluster.query.pl
```

Provision all nodes through MSM for the first time (pick and image name from `x_msm`):

```
> for i in `nodels <_NODERANGE>; do node.modify.pl $i --set os=<image_name>;done
```

Verify the nodes correctly provision and that the correct OS is reported (which may take some time after the provisioning requests are made):

```
> cluster.query.pl
```

Troubleshooting

- **msmctl -a does not report the xCAT plugin -** Check the log file (path specified in `msm.cfg`) for error messages. A common cause is missing Perl modules (Storable, DBD::SQLite, xCAT::Client).
- **cluster.query.pl does not report any nodes -** Check that the xCAT command nodels *<noderange>*, where *<noderange>* is the value configured for _NODERANGE in msm.cfg, outputs the nodes expected.
- **cluster.query.pl does not report OS -** MSM must provision a node to recognize what the current operating system is. It is not sufficient to look up the values in the `nodetype` table because MSM has no way of recognizing whether nodeset and rpower were run with the current values in the `nodetype` table.
- **cluster.query.pl does not report OSLIST, or does not report the expected OSLIST for a node -** Check that the node belongs to the appropriate groups, particularly any listed in the features field of the `x_msm` table for the missing image name.

Deploying Images with TORQUE

When using MSM + xCAT to deploy images with TORQUE, there are some special configuration considerations. Most of these also apply to other workload resource managers.

Note that while the MSM xCAT plugin contains support for manipulating TORQUE directly, this is not an ideal solution. If you are using a version of xCAT that supports prescripts, it is more appropriate to write prescripts that manipulate TORQUE based on the state of the xCAT tables. This approach is also applicable to other workload resource managers, while the xCAT plugin only deals with TORQUE.

Several use cases and configuration choices are discussed in what follows.

Each image should be configured to report its image name through TORQUE. In the TORQUE pbs_mom `mom_config` file the **opsys** value should mirror the name of the image. See **Appendix C: Node Manager (MOM) Configuration** on page 2435 in the TORQUE Administrator's Guide for more information.

Installing Moab on the Management Node

Moab is the intelligence engine that coordinates the capabilities of xCAT and TORQUE to dynamically provision compute nodes to the requested operating system. Moab also schedules workload on the system and powers off idle nodes. Download and install Moab.

Moab Configuration File Example

Moab stores its configuration in the `moab.cfg` file: `/opt/moab/etc/moab.cfg`. A sample configuration file, set up and optimized for adaptive computing follows:

```
SCHEDCFG[Moab]          SERVER=gpc-sched:42559
ADMINCFG[1]             USERS=root,egan
LOGLEVEL                7

# How often (in seconds) to refresh information from TORQUE and MSM
RMPOLLINTERVAL           60,60
RESERVATIONDEPTH        10
DEFERTIME               0
TOOLSDIR                /opt/moab/tools

###############################################################################
# TORQUE and MSM configuration                                                #
###############################################################################
RMCFG[torque]           TYPE=PBS
RMCFG[msm]        TYPE=NATIVE:msm FLAGS=autosync,NOCREATERESOURCE RESOURCETYPE=PROV
RMCFG[msm]        TIMEOUT=60
RMCFG[msm]        PROVDURATION=10:00
AGGREGATENODEACTIONS    TRUE

###############################################################################
# ON DEMAND PROVISIONING SETUP                                                #
###############################################################################
QOSCFG[od]              QFLAGS=PROVISION
USERCFG[DEFAULT]        QLIST=od
NODEALLOCATIONPOLICY    PRIORITY
NODECFG[DEFAULT]        PRIORITYF=1000*OS+1000*POWER
NODEAVAILABILITYPOLICY  DEDICATED
CLASSCFG[DEFAULT]       DEFAULT.OS=scinetcompute

###############################################################
# GREEN POLICIES                                              #
###############################################################
NODECFG[DEFAULT]        POWERPOLICY=ONDEMAND
PARCFG[ALL]             NODEPOWEROFFDURATION=20:00
NODEIDLEPOWERTHRESHOLD  600
# END Example moab.cfg
```

Verifying the Installation

When Moab starts it immediately communicates with its configured resource managers. In this case Moab communicates with TORQUE to get compute node and job queue information. It then communicates with MSM to determine the state of the nodes according to xCAT. It aggregates this information and processes the jobs discovered from TORQUE.

When a job is submitted, Moab determines whether nodes need to be provisioned to a particular operating system to satisfy the requirements of the job. If any nodes need to be provisioned Moab performs this action by creating a provisioning system job (a job that is internal to Moab). This system job communicates with xCAT to provision the nodes and remain active while the nodes are provisioning. Once the system job has provisioned the nodes it informs the user's job that the nodes are ready at which time the user's job starts running on the newly provisioned nodes.

When a node has been idle for a specified amount of time (see NODEIDLEPOWERTHRESHOLD), Moab creates a power-off system job. This job communicates with xCAT to power off the nodes and remains active in the job queue until the nodes have powered off. Then the system job informs Moab that the nodes are powered off but are still available to run jobs. The power off system job then exits.

To verify correct communication between Moab and MSM run the mdiag -R -v msm command.

```
$ mdiag -R -v msm
diagnosing resource managers
RM[msm]       State: Active  Type: NATIVE:MSM  ResourceType: PROV
  Timeout:            30000.00 ms
  Cluster Query URL:  $HOME/tools/msm/contrib/cluster.query.xcat.pl
  Workload Query URL: exec://$TOOLSDIR/msm/contrib/workload.query.pl
  Job Start URL:      exec://$TOOLSDIR/msm/contrib/job.start.pl
  Job Cancel URL:     exec://$TOOLSDIR/msm/contrib/job.modify.pl
  Job Migrate URL:    exec://$TOOLSDIR/msm/contrib/job.migrate.pl
  Job Submit URL:     exec://$TOOLSDIR/msm/contrib/job.submit.pl
  Node Modify URL:    exec://$TOOLSDIR/msm/contrib/node.modify.pl
  Node Power URL:     exec://$TOOLSDIR/msm/contrib/node.power.pl
  RM Start URL:       exec://$TOOLSDIR/msm/bin/msmd
  RM Stop URL:        exec://$TOOLSDIR/msm/bin/msmctl?-k
  System Modify URL:  exec://$TOOLSDIR/msm/contrib/node.modify.pl
  Environment:
MSMHOMEDIR=/home/wightman/test/scinet/tools//msm;MSMLIBDIR=/home/wightman/test/scinet/
tools//msm
  Objects Reported:   Nodes=10 (0 procs)  Jobs=0
  Flags:              autosync
  Partition:          SHARED
  Event Management:   (event interface disabled)
  RM Performance:     AvgTime=0.10s  MaxTime=0.25s  (38 samples)
  RM Languages:       NATIVE
  RM Sub-Languages:   -
```

To verify nodes are configured to provision use the checknode -v command. Each node will have a list of available operating systems.

```
$ checknode n01
node n01
State:      Idle  (in current state for 00:00:00)
Configured Resources: PROCS: 4  MEM: 1024G  SWAP: 4096M  DISK: 1024G
Utilized   Resources: ---
Dedicated  Resources: ---
Generic Metrics:    watts=25.00,temp=40.00
Power Policy:       Green (global policy)   Selected Power State: Off
Power State:   Off
Power:      Off
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      compute   Arch:      ---
  OS Option: compute
  OS Option: computea
  OS Option: gpfscompute
  OS Option: gpfscomputea
Speed:      1.00      CPULoad:   0.000
Flags:      rmdetected
RM[msm]:    TYPE=NATIVE:MSM  ATTRO=POWER
EffNodeAccessPolicy: SINGLEJOB
Total Time: 00:02:30  Up: 00:02:19 (92.67%)  Active: 00:00:11 (7.33%)
```

To verify nodes are configured for Green power management, run the mdiag -G command. Each node will show its power state.

```
$ mdiag -G
NOTE:  power management enabled for all nodes
Partition ALL:  power management enabled
  Partition NodeList:
Partition local:  power management enabled
  Partition NodeList:
  node n01 is in state Idle, power state On (green powerpolicy enabled)
  node n02 is in state Idle, power state On (green powerpolicy enabled)
  node n03 is in state Idle, power state On (green powerpolicy enabled)
  node n04 is in state Idle, power state On (green powerpolicy enabled)
  node n05 is in state Idle, power state On (green powerpolicy enabled)
  node n06 is in state Idle, power state On (green powerpolicy enabled)
  node n07 is in state Idle, power state On (green powerpolicy enabled)
  node n08 is in state Idle, power state On (green powerpolicy enabled)
  node n09 is in state Idle, power state On (green powerpolicy enabled)
  node n10 is in state Idle, power state On (green powerpolicy enabled)
Partition SHARED:  power management enabled
```

To submit a job that dynamically provisions compute nodes, run the msub -l os=<*image*> command.

```
$ msub -l os=computea job.sh
yuby.3
$ showq
active jobs------------------------
JOBID              USERNAME      STATE PROCS   REMAINING            STARTTIME
provision-4            root    Running     8    00:01:00  Fri Jun 19 09:12:56
1 active job               8 of 40 processors in use by local jobs (20.00%)
                           2 of 10 nodes active      (20.00%)
eligible jobs----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME
yuby.3             wightman       Idle     8    00:10:00  Fri Jun 19 09:12:55
1 eligible job
blocked jobs-----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME

0 blocked jobs
Total jobs:  2
```

Notice that Moab created a provisioning system job named `provision-4` to provision the nodes. When `provision-4` detects that the nodes are correctly provisioned to the requested OS, the submitted job `yuby.3` runs:

```
$ showq
active jobs------------------------
JOBID              USERNAME      STATE PROCS   REMAINING            STARTTIME
yuby.3             wightman    Running     8    00:08:49  Fri Jun 19 09:13:29
1 active job               8 of 40 processors in use by local jobs (20.00%)
                           2 of 10 nodes active      (20.00%)
eligible jobs----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME

0 eligible jobs
blocked jobs-----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME

0 blocked jobs
Total job:  1
```

The checkjob command shows information about the provisioning job as well as the submitted job. If any errors occur, run the checkjob -v <*jobid*> command to diagnose failures.

xCAT Plug-in Configuration Parameters

Plugin parameters that begin with an underscore character are specific to the xCAT plug-in; others are common to all plug-ins and may either be set in the **RMCFG[msm]** for all plug-ins, or per plug-in in the **APPCFG[*<plugin_name>*]**.

Description	
Format	Double quoted string containing brief description of plugin.
Default	---
Description	This information is not visible in Moab, but shows up in msmctl -a.

Module	
Format	Moab::MSM::App::xCAT
Default	---
Description	Name of the plugin module to load.

LogLevel	
Format	1-9
Default	*5*

LogLevel	
Description	Used to control the verbosity of logging, 1 being the lowest (least information logged) and 9 being the highest (most information logged). For initial setup and testing, 8 is recommended, then lowering to 3 (only errors logged) for normal operation. Use 9 for debugging, or when submitting a log file for support.

PollInterval	
Format	Integer > 0
Default	*60*
Description	MSM will query xCAT every **POLLINTERVAL** seconds to update general node status. This number will likely require tuning for each specific system. In general, to develop this number, you should pick a fraction of the total nodes MSM will be managing (1/_CQXCATSESSIONS), and time how long it takes run nodestat, rpower stat, and optionally rvitals on these nodes, and add ~15%. Increasing the **POLLINTERVAL** will lower the overall load on the xCAT headnode, but decrease the responsiveness to provisioning and power operations.

TimeOut	
Format	Integer value > POLLINTERVAL
Default	*300*
Description	This parameter controls how long MSM will wait for child processed to complete (all xCAT commands are run in child processes). After **TIMEOUT** seconds, if a child has not returned it will be killed, and an error reported for the operation.

_NodeRange	
Format	Any valid noderange (see the xCAT noderange man page).
Default	All
Description	When MSM queries xCAT this is the noderange it will use. At sites where xCAT manages other hardware that Moab is not intended to control, it is important to change this.

_CQxCATSessions	
Format	Positive integer > 1
Default	*10*
Description	MSM will divide the node list generated by nodels into this many groups and simultaneously query xCAT for each group. The value may need tuning for large installations, higher values will cause the time to complete a single cluster query to go down, but cause a higher load on the xCAT headnode.

_DORVitals	
Format	*0* or *1*
Default	*0*
Description	When set to *1*, MSM will poll rvitals power and led status (see the xCAT rvitals man page). This only works with IBM BMCs currently. In order to use this, xCAT should respond without error to the rvitals *<noderange>* watts and rvitals *<noderange>* leds commands. Status is reported as **GMETRTIC[watts]** and **GMETRIC[leds]**. See also the **_PowerString** on page 1068 configuration parameter.

_PowerString	
Format	single quote delimited string
Default	*'AC Avg Power'*
Description	Only meaningful when used with **_DORVitals** on page 1068=*1*. Some BMCs return multiple responses to the rvitals command, or use slightly different text to describe the power metrics. Use this parameter to control what is reported to Moab. You can use *'`$MSMLIBDIR/contrib/xcat/dump.xcat.cmd.pl` rvitals <node_name> power'* and examine the output to determine what the appropriate value of this string is.

_DoNodeStat	
Format	*0* or *1*
Default	*1*

_DoNodeStat	
Description	If set to *0*, MSM will not call nodestat to generated a substate. This can be used to speed up the time it takes to query xCAT, and you do not need the substate visible to Moab.

_DoxCATStats	
Format	*0* or *1*
Default	*0*
Description	If Set to *1*, MSM will track performance statistics about calls to xCAT, and the performance of higher level operations. The information is available via the script `$MSMHOMEDIR/contrib/xcat/xcatstats.pl`. This parameter is useful for tuning the **POLLINTERVAL** and **_CQxCATSessions** on page 1068 configuration parameters.

_LockDir	
Format	Existing path on MSM host
Default	*$MSMHOMEDIR/lck*
Description	This is a path to where MSM maintains lock files to control concurrency with some Xen and KVM operations.

_HVxCATPasswdKey	
Format	key value in the xCAT passwd table
Default	*vmware*
Description	This is where MSM gets the user/password to communicate with ESX hypervisors.

_FeatureGroups	
Format	Comma delimited string of xCAT group names.
Default	---

_FeatureGroups	
Description	MSM builds the OSLIST for a node as the intersection of _FEATUREGROUPS, features specified in `x_msm` for that image, and the nodes group membership. The value 'torque' is special, and indicates that the image uses TORQUE, and the node should be added/removed from TORQUE during provisioning when used in conjunction with the _ModifyTORQUE on page 1074 parameter.

_DefaultVMCProc	
Format	1-?
Default	*1*
Description	If not explicitly specified in the create request, MSM will create VMs with this many processors.

_DefaultVMDisk	
Format	Positive integer values, minimum is determined by your vm image needs
Default	*4096*
Description	If not explicitly specified in the create request, MSM will create VMs with this much disk allocated.

_DefaultVMCMemory	
Format	Positive integer values, minimum is determined by your vm image needs
Default	*512*
Description	If not specified, MSM will create VMs with this much memory allocated.

_KVMStoragePath	
Format	Existing path on MSM host
Default	*/vms*
Description	File backed disk location for stateful KVM VMS will be placed here.

_ESXStore	
Format	Mountable NFS Path
Default	---
Description	Location of ESX stores.

_ESXCFGPath	
Format	Mountable NFS Path
Default	*ESXStore*
Description	Location of ESX VM configuration files.

_VMInterfaces	
Format	Name of bridge device in your VM image
Default	*br0*
Description	Bridge device name passed to libvirt for network configuration of VMs (overrides _ XENHOSTINTERFACES and _KVMHOSTINTERFACES if specified).

_XenHostInterfaces	
Format	Name of bridge device in your VM image
Default	*xenbr0*
Description	Bridge device name passed to libvirt for network configuration of Xen VMs.

_KVMHostInterfaces	
Format	Name of bridge device in your VM image
Default	*br0*

_KVMHostInterfaces	
Description	Bridge device name passed to libvirt for network configration of KVM VMs.

_VMSovereign	
Format	*0* or *1*
Default	*0*
Description	Setting this attribute will cause Moab to reserve VMs' memory and procs on the hypervisor and treat the VM as the workload — additional workload cannot be scheduled on the VM.

_UseStates	
Format	Valid xCAT chain.currstate values (see the xCAT chain man page)
Default	*boot,netboot,install*
Description	Nodes that do not have one of these values in the xCAT `chain.currstate` field will reported with **STATE**=*Updating*. Use this configuration parameter to prevent Moab from scheduling nodes that are updating firmware, etc.

_ImagesTabName	
Format	Existing xCAT table that contains your image definitions.
Default	*x_msm*
Description	This table specifies the images that may be presented to Moab in a node's `OSLIST`. The xCAT schema for this table is defined in *$MSMHOMEDIR*`/contrib/xcat/MSM.pm`, which needs to be copied to the *$XCATROOT*`/lib/perl/xCAT_schema` directory.

_VerifyRPower	
Format	*0* or *1*
Default	*0*

_VerifyRPower	
Description	If set, MSM will attempt to confirm that rpower requests were successful by polling the power state with rpower stat until the node reports the expected state, or **_RPowerTimeOut** on page 1073 is reached. **NOTE**: This can create significant load on the xCAT headnode.

_RPowerTimeOut	
Format	Positive integer values
Default	*60*
Description	Only meaningful when used with **_VerifyRPower** on page 1072. If nodes do not report the expected power state in this amount of time, a GEVENT will be produced on the node (or system job).

_QueueRPower	
Format	*0* or *1*
Default	*0*
Description	When set, this parameter will cause MSM to aggregate rpower requests to xCAT into batches. The timing and size of these batches is controlled with the **_RPowerQueueAge** on page 1073 and **_RPowerQueueSize** on page 1074 parameters. **NOTE**: This can significantly reduce load on the xCAT headnode, but will cause the power commands to take longer, and MSM shutdown to take longer.

_RPowerQueueAge	
Format	Positive integer values
Default	*30*
Description	Only meaningful when used with **_QueueRPower** on page 1073. MSM will send any pending rpower requests when the oldest request in the queue exceeds this value (seconds).

_RPowerQueueSize	
Format	Positive integer values
Default	*200*
Description	Only meaningful when used with **_QueueRPower** on page 1073. MSM will send any pending rpower requests when the queue depth exceeds this value.

_MaskOSWhenOff	
Format	*0* or *1*
Default	*0*
Description	When set, this parameter will cause MSM to report **OS**=*None* for nodes that are powered off. This may be useful when mixing stateless and stateful images, forcing Moab to request provisioning instead of just powering on a node.

_ModifyTORQUE	
Format	*0* or *1*
Default	*0*
Description	When set, this parameter will cause MSM to add and removes nodes and VMs from TORQUE as required by provisioning. See the **_FeatureGroups** on page 1069 parameter as well.

_ReportNETADDR	
Format	*0* or *1*
Default	*0*
Description	When set, this parameter will cause MSM to report NETADDR=<hosts.ip from xCAT>.

_UseOpIDs	
Format	*0* or *1*

_UseOpIDs	
Default	*0*
Description	When set, this parameter will cause errors to be reported as GEVENTs on the provided system job, instead of a node (Moab 5.4 only, with appropriate Moab CFG)

_VMIPRange	
Format	Comma separated list of dynamic ranges for VM (ex '10.10.23.100-200,10.10.24.1-255')
Default	---
Description	Use this parameter to specify a pool of IPs that MSM should assign to VMs at creation time. IPs are selected sequentially from this list as available. Omit this configuration parameter if an external service is managing IP assignment, or if they are all previously statically assigned.

_xCATHost	
Format	*<xcat_headnode>*:*<xcatd_port>*
Default	*localhost:3001*
Description	Use to configure MSM to communicate with xCAT on another host.

_NoRollbackOnError	
Format	*0* or *1*
Default	*0*
Description	When an error occurs and rollback is activated (as it is by default), rollback causes a reversion to the previous successful request. _NoRollbackOnError is useful for debugging to determine the xCAT state if no rollback occurred. If set to 1 and an error occurs between MSM and xCAT when creating a node, assigning a name (DNS) to a node, or assigning an IP address (DHCP) to a node, then no rollback occurs.

Event Dictionary

See "Logging Overview" for more information about Moab logging.

Moab Event Dictionary

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-0005	USER	system.moab	INFO	MWM_TESTING_INFO	Testing with argument1: %s. and argument2: %s and argument3: %s and argument4: %s	Internal error for testing diagnostics.
0x100-0065	USER	domain.lifecycle	INFO	MWM_JOB_CANCEL	Job %s was canceled. %s	The job was canceled.
0x100-0066	USER	domain.lifecycle	INFO	MWM_JOB_END_SUCCESSFUL	Job %s finished successfully. %s	The job finished successfully.
0x100-0068	USER	domain.lifecycle	INFO	MWM_JOB_USER_HOLD	Job %s had a user hold applied.	A user hold was applied to the job.
0x100-0069	USER	domain.lifecycle	INFO	MWM_JOB_SYSTEM_HOLD	Job %s had a system hold applied.	A system hold was applied to the job.
0x100-006a	USER	domain.lifecycle	INFO	MWM_JOB_BATCH_HOLD	Job %s had a batch hold applied.	A batch hold was applied to the job.
0x100-006b	USER	domain.lifecycle	INFO	MWM_JOB_DEFER_HOLD	Job %s had a defer hold applied.	A defer hold was applied to the job.
0x100-006c	USER	domain.lifecycle	INFO	MWM_JOB_MODIFY	Job %s was modified. %s	One of the attributes of the job was modified either via a user initiated action or an automated action.
0x100-006d	USER	domain.lifecycle	INFO	MWM_JOB_REJECT	Job %s was rejected. %s	The job was rejected for some reason.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-006e	USER	domain.lifecycle	INFO	MWM_JOB_RELEASE	Job %s was released.	Any holds placed on the job have been released, and the job is not prevented from running due to any hold action. The job may still not be able to run due to other considerations.
0x100-006f	USER	domain.lifecycle	INFO	MWM_JOB_START	Job %s was started. %s	The job was started on its designated node[s].
0x100-0070	USER	domain.lifecycle	INFO	MWM_JOB_SUBMIT	Job %s was submitted. %s	The job has been submitted to Moab and is being evaluated and processed.
0x100-0071	USER	domain.lifecycle	INFO	MWM_JOB_CREATED	Job %s was created.	The job has been created and will be queued for execution.
0x100-0072	USER	domain.lifecycle	INFO	MWM_JOB_REQUEUE	Job %s was requeued. %s	The job has been requeued so it can be executed again.
0x100-0073	USER	domain.lifecycle	INFO	MWM_JOB_CANCEL_CLEANUP_STARTED	Job %s is being cleaned up due to cancel request.	The job has been issued a cancel request and is being cleaned up.
0x100-0074	USER	domain.lifecycle	INFO	MWM_JOB_CLEANUP_STARTED	Job %s is being cleaned up.	The job has ended and is being cleaned up.
0x100-0075	USER	domain.lifecycle	INFO	MWM_JOB_DEFERRED	Job %s has been deferred.	The job has been deferred.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-00c9	USER	domain.li-fecycle	INFO	MWM_NODE_EVAC_VMS	Evacuating VMs off node %s.	Evacuating VMs off the node.
0x100-012c	USER	domain.li-fecycle	INFO	MWM_RSV_CREATE	Reservation %s was created. %s	The reservation has been created and is stored in the system.
0x100-012d	USER	domain.li-fecycle	INFO	MWM_RSV_START	Reservation %s has started.	The reservation has started.
0x100-012e	USER	domain.li-fecycle	INFO	MWM_RSV_END	Reservation %s has ended.	The reservation has ended.
0x100-0190	USER	sys-tem.-moab	INFO	MWM_SCHED_COMMAND	The following sched-uler command was submitted: %s	External commands are submitted to Moab in a variety of ways. This event doc-uments the com-mand line and possibly other inform-ation associated with the command. These commands typically have the ability to change beha-vior/state within Moab. Commands that are typically queries are not included.
0x100-0192	USER	sys-tem.-moab	INFO	MWM_SCHED_CYCLE_START	A scheduler iteration is beginning. %s	Moab periodically checks through sub-mitted jobs and makes decisions regarding which jobs are scheduled. One of these iterations is beginning now.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-0193	USER	sys-tem.-moab	INFO	MWM_SCHED_CYCLE_END	A scheduler iteration is ending.	Moab periodically checks through sub-mitted jobs and makes decisions regarding which jobs are scheduled. One of these iterations is ending now.
0x100-0194	USER	sys-tem.-moab	INFO	MWM_SCHED_PAUSE	The scheduler has been paused. %s	The Moab scheduler has been admin-istratively paused. New jobs may be sub-mitted and existing jobs will continue to run, but no new jobs will be scheduled as long as Moab is paused.
0x100-0195	USER	sys-tem.-moab	INFO	MWM_SCHED_RECYCLE	The scheduler has been recycled. %s	The Moab scheduler has been admin-istratively recycled. The process will cleanly exit and save the state data. It will then restart, read in the data, and resume scheduling.
0x100-0196	USER	sys-tem.-moab	INFO	MWM_SCHED_RESUME	The scheduler has been resumed.	The Moab scheduler has been admin-istratively resumed. A new scheduling iter-ation will begin imme-diately and continue regularly.
0x100-0197	USER	sys-tem.-moab	INFO	MWM_SCHED_START	The scheduler has started.	The Moab scheduler has started.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-0198	USER	system.moab	INFO	MWM_SCHED_STOP	The scheduler has stopped. %s	The Moab scheduler has stopped.
0x100-01f4	USER	domain.lifecycle	INFO	MWM_TRIG_CREATE	Trigger %s has been created.	The named trigger has been created and is now recognized in the Moab system.
0x100-01f5	USER	domain.lifecycle	INFO	MWM_TRIG_START	Trigger %s has started.	The named trigger has started its action.
0x100-01f6	USER	domain.lifecycle	INFO	MWM_TRIG_END	Trigger %s has ended. %s	The named trigger has finished its action.
0x100-01f8	USER	domain.lifecycle	INFO	MWM_TRIG_THRESHOLD	Trigger %s threshold event: %s	A trigger threshold has been encountered. Additional details regarding the threshold may be included in the text.
0x100-0258	USER	domain.lifecycle	INFO	MWM_VM_SUBMIT	VM %s has been submitted.	The named VM has been submitted and is now recognized in the Moab system.
0x100-0259	USER	domain.lifecycle	INFO	MWM_VM_DESTROY	VM %s has been terminated.	The named VM has finished its lifecycle and is now removed and added to the completed table.
0x100-025a	USER	domain.lifecycle	INFO	MWM_VM_CANCEL	VM %s has been canceled.	The named VM has been canceled.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-025b	USER	domain.li-fecycle	INFO	MWM_VM_END	VM %s has been ended.	The named VM has been canceled because it has exceeded its alloc-ated walltime.
0x100-025c	USER	domain.li-fecycle	INFO	MWM_VM_ MIGRATE_START	VM %s migration has started. (%s)	The named VM has started its migration. Additional inform-ation may be provided regarding source and des-tination nodes.
0x100-025d	USER	domain.li-fecycle	INFO	MWM_VM_ MIGRATE_END	VM %s migration has finished. (%s)	The named VM has finished its migration. Additional inform-ation may be provided regarding source and des-tination nodes.
0x100-025f	USER	domain.li-fecycle	INFO	MWM_VM_ MANUAL_ MIGRATE_START	VM %s migration started manually. (%s)	The named VM migration has been started manually. Additional inform-ation may be provided regarding source and des-tination nodes.
0x100-0260	USER	domain.li-fecycle	INFO	MWM_VM_ READY	VM %s is ready.	The named VM is ready. It has been linked to an internal tracking job.
0x100-2711	USER	sys-tem.-moab	INFO	MWM_ PARAMETER_ SET_TO_VALUE_ INFO	Parameter '%s' is set to '%s'.	A parameter was set to a specified value. This is usually accom-plished via a con-figuration file.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-2741	USER	system.moab	INFO	MWM_SOCKET_EXCEPTION	Exception detected in select for socket %s.	The select() system call indicated an exception for this socket.
0x100-2742	USER	system.moab	INFO	MWM_SOCKET_EXCEPTION_REASON	Exception identified as '%s' in select for socket %s.	The select() system call indicated an exception for this socket. It has been identified with an error id by getsockopt().
0x100-2748	USER	system.moab	INFO	MWM_MOAB_STARTED_ON_CORRECT_HOST	Server started on host '%s' %s.	Moab is started on either the primary or fallback server.
0x100-2762	USER	system.moab	INFO	MWM_CONFIG_LINE_SUCCESSFUL	Configuration line '%s' successfully processed.	The line in the configuration file was processed without error.
0x100-2935	USER	system.moab	INFO	MWM_ACTIVE_JOB_REMOVED_FROM_QUEUE	Active %s job %s has been removed from the queue, default to successful completion.	The job was removed from the indicated resource manager while it was still active. By default it is assumed to complete successfully unless more information is available (i.e. ENABLEFAILUREFORPURGEDJOB).

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-2936	USER	sys-tem.-moab	INFO	MWM_INACTIVE_JOB_REMOVED_FROM_QUEUE	Inactive %s job %s has been removed from the queue, default to status 'can-celed'.	The job was removed from the indicated resource manager while it was still act-ive. By default it is given status 'can-celed' unless more information is avail-able (i.e. ENABLEFAILUREFOR-PURGEDJOB).
0x100-2937	USER	sys-tem.-moab	INFO	MWM_RM_DOWN_SKIPPING_WORK	RM %s state is %s, skipping %s.	The specified resource manager is not in a good state. Certain actions might be skipped while it is in this state.
0x100-296a	USER	sys-tem.-moab	INFO	MWM_CANNOT_RESUME_JOB	Cannot resume job '%s' (%s).	Check the PBS server log to see reason for failure.
0x100-296b	USER	sys-tem.-moab	INFO	MWM_CANNOT_LOCATE_RESOURCE	Cannot locate %s '%s'.	Unable to find the resource specified.
0x100-296c	USER	sys-tem.-moab	INFO	MWM_CANNOT_SET_JOB_CLASS	Cannot set class on job '%s' to '%s' (%s).	The job could not be modified.
0x100-296d	USER	sys-tem.-moab	INFO	MWM_NATIVE_ACTION_MISSING	%s action not spe-cified for native inter-face. %s.	The native interface allows custom actions to be specified. No value was specified for this action.
0x100-296e	USER	sys-tem.-moab	INFO	MWM_COMMAND_SENT	Command sent to server.	A command was sent.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-296f	USER	system.moab	INFO	MWM_JOB_MAXPREMPT	JOBMAXPREEMPTPERITERATION reached: %s of %s.	The maximum value was reached.
0x100-2970	USER	system.moab	INFO	MWM_JOB_CHANGED_STATES	Job '%s' changed states from '%s' to '%s'.	The state changed.
0x100-2971	USER	system.moab	INFO	MWM_JOB_RELEASING_RESERVATION	Job '%s' was requeued/rejected. Releasing reservation.	The job no longer holds the reservation.
0x100-2972	USER	system.moab	INFO	MWM_NODE_CHANGED_STATES	Node '%s' changed states from '%s' to '%s'.	The node state changed.
0x100-2973	USER	system.moab	INFO	MWM_JOB_ACTION_SUCCESSFUL	Job '%s' successfully %s.	The job action completed.
0x100-2974	USER	system.moab	INFO	MWM_ALLOC_TEMP_MEMORY	Cannot allocate temp memory for %s completed jobs.	The system may be low on memory.
0x100-2975	USER	system.moab	INFO	MWM_ACTION_LAUNCHED	Action '%s' launched with message '%s'.	Scheduler action is about to be executed.
0x100-2976	USER	system.moab	INFO	MWM_JOB_ADJUSTMENT	Adjusting allocated %s to %s for job '%s'.	The value is being changed.
0x100-2977	USER	system.moab	INFO	MWM_ALL_JOBS_LOADED	All jobs loaded.	The jobs have been loaded.
0x100-2978	USER	system.moab	INFO	MWM_ALL_NODES_LOADED	All located non-native nodes loaded (%s).	The nodes have been loaded.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-2979	USER	system.moab	INFO	MWM_BACKFILL_POLICY_DISABLED	Backfill policy disabled.	The policy was disabled.
0x100-297a	USER	system.moab	INFO	MWM_JOB_LOAD	Cannot load job '%s'.	The job failed to load.
0x100-297b	USER	system.moab	INFO	MWM_CANNOT_CREATE_RSV	Cannot create reservation.	The request to create the given reservation has failed.
0x100-297c	USER	system.moab	INFO	MWM_MODIFY_PARTITION	Cannot modify partition of running job '%s'.	Must wait until job completes.
0x100-297f	USER	system.moab	INFO	MWM_CANNOT_CREATE_RSV_IN_PARTITION	Cannot create reservation for job '%s' in partition '%s'.	Job cannot be run on requested partition.
0x100-2982	USER	system.moab	INFO	MWM_CLUSTER_QUERY_GETDATA	Cluster query getdata failed for native interface.	The resource manager may be down or unresponsive.
0x100-2a0b	USER	system.moab	INFO	MWM_SENDING_CLIENT_COMMAND	Sending %s command: '%s'.	The specified command is being sent to the server.
0x100-2a0e	USER	system.moab	INFO	MWM_SCHED_SHUTDOWN_REQUEST	The scheduler has received a user shutdown request.	The Moab scheduler has received a request to shut down. It will be processed as soon as possible.
0x100-2a0f	USER	system.moab	INFO	MWM_SCHED_RECYCLE_REQUEST	The scheduler has received a user recycle request.	The Moab scheduler has received a request to recycle. It will be processed as soon as possible.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-2a10	USER	system.moab	INFO	MWM_SCHED_PAUSE_DESCRIPTION	Scheduling will be disabled, cluster information will continue to be updated.	This is a description of what happens when the scheduler is paused.
0x100-2a11	USER	system.moab	INFO	MWM_SCHED_STOP_TIMESTAMP	Scheduling will stop in %s at iteration %s.	This provides a log message of when the scheduler will stop.
0x100-2a12	USER	system.moab	INFO	MWM_SCHED_RESUME_TIMESTAMP	Scheduling will resume in %s seconds.	This provides a log message of when the scheduler will resume.
0x100-2a13	USER	system.moab	INFO	MWM_SCHED_RESTART_TIME_REACHED	Scheduler restart time reached (scheduler will restart).	The configured restart time was reached. (RESTARTINTERVAL or MEMREFRESHINTERVAL.
0x100-2a14	USER	system.moab	INFO	MWM_SCHED_COMPLETE_SCHEDULING	Scheduling complete. Sleeping for %s seconds.	The scheduling portion of the iteration is complete. Additional jobs will not be scheduled until the next iteration.
0x100-2a17	USER	system.moab	INFO	MWM_ABOUT_TO_EXEC	About to exec() '%s'.	The process is about to be executed.
0x100-2a18	USER	system.moab	INFO	MWM_JOB_ARRAY_CANCEL_POLICY	Sub-job %s exit code %s canceled job array %s with policy %s.	A job within an array job finished and, depending on its exit code and the policy in place, the entire array job might cancel.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-2a19	USER	sys-tem.-moab	INFO	MWM_RESERVATION_COMPLETION_DELAYED	Reservation com-pletion for job '%s' delayed from %s to %s.	The reservation end time is later than ini-tially expected for this job.
0x100-2a1b	USER	sys-tem.-moab	INFO	MWM_VM_ORPHANED	VM '%s' successfully orphaned.	The VM has been sep-arated from its track-ing job.
0x100-2a1c	USER	sys-tem.-moab	INFO	MWM_VM_REPORTED_DESTROYED	VM '%s' reported destroyed via RM - removing VM.	The VM is no longer available from the resource manager, so it is being removed from the scheduler.
0x100-2a1d	USER	sys-tem.-moab	INFO	MWM_VM_STALE_REPORT	VM '%s' has not been reported in %s seconds.	The VM is no longer being reported from the resource man-ager. No action is cur-rently being taken.
0x100-2a1e	USER	sys-tem.-moab	INFO	MWM_WIKI_KEYWORD_NOT_HANDLED	Wiki keyword '%s' (%s) not handled.	The keyword was not recognized, so it will be ignored.
0x100-2a20	USER	sys-tem.-moab	INFO	MWM_ROLLING_LOGFILE	Rolling logfile '%s' to '%s'.	The old logfile will be closed and logging will resume in the new file.
0x100-2a23	USER	sys-tem.-moab	INFO	MWM_NODE_LOCATED	Nodes located for job %s: %s of %s required (%s feas-ible).	List of nodes located for a specific job.
0x100-2a2d	USER	sys-tem.-moab	INFO	MWM_JOB_PAL_SET	Partition access list set to value: %s.	The partition access list (PAL) is set.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-2a2e	USER	system.moab	INFO	MWM_JOB_PREEMPTED_BY_JOB	Job %s preempted job %s - added idle resources (T: %s; N: %s; P: %s)/remaining (T: %s; N: %s; P: %s).	Job was preempted by another job.
0x100-2a2f	USER	system.moab	INFO	MWM_JOB_CAN_START_WITHOUT_PREMEPTION	Job %s would start in %s without preemption (PC: %s).	Job can start without preemption.
0x100-2a32	USER	system.moab	INFO	MWM_PARTITON_RESOURCES	Partition %s nodes/procs available %s/%s (%s jobs examined).	General parition information.
0x100-2a33	USER	system.moab	INFO	MWM_RSV_OPERATION	Performing '%s' operation on reservation expression '%s' (%s matches).	This is operation is caused by a mrsvctl command.
0x100-2a34	USER	system.moab	INFO	MWM_PREEMPTING_JOBS	Preempting jobs to allow job %s to start - required resources T: %s; N: %s; P: %s.	Preempting jobs.
0x100-2a35	USER	system.moab	INFO	MWM_MOABTRACKSUSPEND	Preempt usage tracking enabled (env).	Environment variable MOABTRACKSUSPEND set.
0x100-2a36	USER	system.moab	INFO	MWM_JOB_MAX_PREEMPTEE_LIMIT	Single job max preemptee limit (%s) reached.	Max requirements exceeded on job.
0x100-2a37	USER	system.moab	INFO	MWM_QUEUES_DETECTED	Queues detected: %s.	Resource manager found queues on cluster query update.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-2a38	USER	sys-tem.-moab	INFO	MWM_JOB_START_TIME_CHANGED	Start time changed from %s to %s on job %s.	The job's start time was changed via the resource manager.
0x100-2a39	USER	sys-tem.-moab	INFO	MWM_STORING_CHECKPOINT_INFO	Storing object to checkpoint.	The object's state is being checkpointed.
0x100-2a3a	USER	sys-tem.-moab	INFO	MWM_PBS_DATA_UP_TO_DATE	PBS raw data already up to date.	The resource man-ager is already updated.
0x100-2a3b	USER	sys-tem.-moab	INFO	MWM_PBS_DATA_UPDATED	PBS data updated for iteration %s.	The resource man-ager is now updated.
0x100-2a3c	USER	sys-tem.-moab	INFO	MWM_STARTED_MESSAGE_QUEUE	Started message queue thread.	The message queue is now operational.
0x100-2a3d	USER	sys-tem.-moab	INFO	MWM_JOBS_SELECTED_IN_PARTITION	Total jobs selected in partition %s: %s/%s.	Identifies the selec-ted jobs in a par-tition.
0x100-2a3e	USER	sys-tem.-moab	INFO	MWM_TASKS_LOCATED_FOR_JOB	Tasks located for job %s: %s of %s required (%s feas-ible).	Identifies the tasks available for a job.
0x100-2a3f	USER	sys-tem.-moab	INFO	MWM_CLIENT_REQUEST	Client requesting command '%s'.	Client requested com-mand.
0x100-2a40	USER	sys-tem.-moab	INFO	MWM_REQUEST_TO_CANCEL_JOB	Request to cancel job '%s' sent, but could not confirm can-celation (pending response).	Client did not get a confirmation as expected.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-2a41	USER	system.moab	INFO	MWM_RESERVATION_NOT_ALLOWED_FOR_JOB	Reservation not allowed for job %s in %s.	Reservation not allowed in specified condition.
0x100-2a42	USER	system.moab	INFO	MWM_RESERVED_JOB_STARTED	Reserved job '%s' started.	Reserved job started.
0x100-2a43	USER	system.moab	INFO	MWM_RESOURCES_AVAILABLE_AFTER_SCHEDULING	Resources available after scheduling: N: %s P: %s.	Resources available after scheduling.
0x100-2a44	USER	system.moab	INFO	MWM_RESTORING_DEFERRED_JOB	Restoring job '%s' from deferred state.	Restoring job from deffered state.
0x100-2a45	USER	system.moab	INFO	MWM_RM_DUPLICATE_QUERY	RM %s already has a pending query - skipping get data query.	Duplicate queries may not be performed simultaneously.
0x100-2a46	USER	system.moab	INFO	MWM_RM_PEER_COMMAND	Sending peer server command to %s:%s (Cmd: %s, Requestor: %s, Key: %s...).	A command has been sent to a peer Moab grid server.
0x100-2a47	USER	system.moab	INFO	MWM_SET_ATTRIBUTE_ON_NODE	Setting %s on node %s to %s.	A command has been sent to a peer Moab grid server.
0x100-2a48	USER	system.moab	INFO	MWM_SET_ATTRIBUTE_ON_JOB	Setting %s on job %s to %s (%s).	A command has been sent to a peer Moab grid server.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-a73c	ADMI-N	sys-tem.-moab	INFO	MWM_CANNOT_STAT_FILE_INFO	Cannot stat file '%s', errno: %s (%s).	The stat() system call failed. This is not always significant as it is sometimes used to test the existence of a file that may or may not be there. Use the errno and associated message to determine possible causes.
0x100-a743	ADMI-N	sys-tem.-moab	INFO	MWM_FAILED_SELECT	Select for socket %s failed, errno: %s (%s).	The select() system call failed. Use the errno and associated message to determ-ine possible causes.
0x100-a744	ADMI-N	sys-tem.-moab	INFO	MWM_SELECT_TIMEOUT	Select for socket %s timed out after %s seconds with no valid descriptors.	The select() system call timed out. This may or may not be an error. Check MTU.
0x100-a75d	ADMI-N	sys-tem.-moab	INFO	MWM_CONFIG_VALUE_OUT_OF_RANGE	Configuration para-meter '%s' has an invalid value '%s'. Range is limited by %s.	Check the line in the configuration file for the attribute.
0x100-a76d	ADMI-N	sys-tem.-moab	INFO	MWM_REMOVING_OBJECT_FROM_MONGO	Removing object '%s' from Mongo DB '%s'.	The object is being removed from the database.
0x100-a789	ADMI-N	sys-tem.-moab	INFO	MWM_UNABLE_TO_ALLOCATE_NODES_FOR_RSV	Cannot allocate nodes for reser-vation '%s'. (%s)	Cannot allocate a node list that matches the require-ments for this reser-vation. This may not be serious since mul-tiple passes may occur.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-a8ab	ADMI-N	sys-tem.-moab	INFO	MWM_VM_EXCEED_TTL	VM '%s' has reached TTL (%s). Must be removed manually.	The given VM has reached its time to live.
0x100-a8e1	ADMI-N	sys-tem.-moab	INFO	MWM_PLUGIN_LOADED_SUCCESS	Successfully loaded NodeAllocation plu-gin '%s' for partition '%s'.	A NodeAllocation plu-gin was loaded without error.
0x100-a95b	ADMI-N	sys-tem.-moab	INFO	MWM_RSV_FULL	Full reservation '%s' reserved %s procs in partition '%s' to start in %s at (%s).	The full reservation has been reserved.
0x100-a97d	ADMI-N	sys-tem.-moab	INFO	MWM_RSV_PREREQ_JOB	Cannot create reser-vation for pre-requisite job '%s'.	Could not obtain a reservation for this job.
0x100-a97e	ADMI-N	sys-tem.-moab	INFO	MWM_ANNOTATE_JOB	Cannot annotate job '%s' with message '%s'.	Unable to modify the job with the annota-tion.
0x100-a980	ADMI-N	sys-tem.-moab	INFO	MWM_UPDATE_JOB	Cannot update job '%s'.	The update on the job from XML failed.
0x100-a981	ADMI-N	sys-tem.-moab	INFO	MWM_REMAP_CLASS	Cannot remap class for RM job '%s' (%s).	Unable to modify the job with the new class.
0x100-a983	ADMI-N	sys-tem.-moab	INFO	MWM_COMPLETING_JOB	Completing job '%s'.	The job finished.
0x100-a984	ADMI-N	sys-tem.-moab	INFO	MWM_JOBS_DETECTED	There were %s %s jobs detected on RM '%s'.	The resource man-ager reported these jobs.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-a985	ADMIN	system.moab	INFO	MWM_SUSPEND_JOB	Cannot suspend job '%s' (%s).	Check the PBS server log to see reason of failure.
0x100-a986	ADMIN	system.moab	INFO	MWM_STALE_PARTITION	Attempting to remove stale partition for completed job '%s'.	About to perform the stated operation.
0x100-a987	ADMIN	system.moab	INFO	MWM_STALE_PARTITION_SUCCESS	Successfully removed stale partition for completed job '%s'.	Successfully performed the stated operation.
0x100-a988	ADMIN	system.moab	INFO	MWM_CANCEL_NOQUEUE_JOB	Canceling No-queue job '%s'.	About to perform the stated operation.
0x100-a989	ADMIN	system.moab	INFO	MWM_SIGNAL_JOB	Cannot signal job %s' (%s).	The resource manager did not respond to the signal request.
0x100-a98a	ADMIN	system.moab	INFO	MWM_RSV_JOB_CREDS	Cannot set up reservation job credentials.	The user, account, or group credentials may not be valid.
0x100-a98b	ADMIN	system.moab	INFO	MWM_CP_CORRUPT_NODE_LINE	Corrupt node line detected (%s).	The line does not contain the correct syntax for a checkpoint.
0x100-a98c	ADMIN	system.moab	INFO	MWM_EVALUATING_RSV	Evaluating reservation '%s'.	About to perform the stated operation.
0x100-a98d	ADMIN	system.moab	INFO	MWM_EXPIRING_CHECKPOINT_DATA	Expiring checkpoint data for %s '%s'. Not updated in %s.	The object's checkpoint data has expired.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-a98e	ADMIN	system.moab	INFO	MWM_JOB_PREVIOUSLY_REMOVED	Job '%s' was previously removed.	The job has already been removed.
0x100-a98f	ADMIN	system.moab	INFO	MWM_JOB_STARTED_BY_USER	Job '%s' was started by user '%s'.	The job started.
0x100-a990	ADMIN	system.moab	INFO	MWM_JOB_NOT_STARTED_BY_USER	Job '%s' could not be started by user '%s' (%s).	The job could not be started.
0x100-a991	ADMIN	system.moab	INFO	MWM_RM_JOB_NOT_STARTED	Job '%s' could not be started with %s RM '%s' (%s).	The job could not be started.
0x100-a992	ADMIN	system.moab	INFO	MWM_JOB_CANCELED_EXTERNALLY	Job '%s' appears to have been canceled externally.	The job was canceled.
0x100-a993	ADMIN	system.moab	INFO	MWM_JOB_COMPLETED_SINGLE_ITERATION	Job '%s' appears to have been started and completed in a single iteration.	The job completed.
0x100-a994	ADMIN	system.moab	INFO	MWM_JOB_PROCESSING_COMPLETED	Job processing completed.	The jobs have been processed.
0x100-a995	ADMIN	system.moab	INFO	MWM_PROCESSING_JOB	Processing job '%s' in state '%s'.	Processing a single job.
0x100-a996	ADMIN	system.moab	INFO	MWM_JOB_SUSPENDED	Job '%s' suspended through %s RM.	The resource manager suspended the job.
0x100-a997	ADMIN	system.moab	INFO	MWM_JOB_RESUMED	Job '%s' resumed through %s RM.	The resource manager resumed the job.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-a998	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_FEASIBILITY_CHECK_DISABLED	Job feasibility check disabled (env).	This feature has been disabled.
0x100-a999	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_USAGE_SENT	Job usage sent for job '%s'.	The usage sent as feedback to user.
0x100-a99a	ADMI-N	sys-tem.-moab	INFO	MWM_LOADING_JOBS	Loading %s job(s).	The jobs are about to be loaded.
0x100-a99b	ADMI-N	sys-tem.-moab	INFO	MWM_LOADING_NODE_RECORDS	Loading %s node record(s).	The node records are about to be loaded.
0x100-a99c	ADMI-N	sys-tem.-moab	INFO	MWM_LOADED_WORKLOAD_BUFFER	Loaded %s workload buffer (%s bytes), processing jobs.	The workload buffer was loaded.
0x100-a99d	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_REJECTED_INFINITE_WALLTIME	Job '%s' rejected (requested infinite walltime).	Jobs must have a walltime limit.
0x100-a99e	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_REJECTED_PARTITION	Job '%s' rejected in partition %s (exceeds maximum task size: %s > %s).	Adjust JOBMAXTASKCOUNT in the configuration file.
0x100-a99f	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_ALREADY_EXISTS	Job '%s' already exists but is not a duplicate.	The ID of the job matched a completed job.
0x100-a9a0	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_ALREADY_BATCH_HOLD	Job '%s' is already on batch hold.	Trying to place a job on hold that is already in that state.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-a9a1	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_REQUESTS_RSV	Job '%s' requests reservation '%s' (not deferring).	The job is requesting the reservation.
0x100-a9a2	ADMI-N	sys-tem.-moab	INFO	MWM_RM_CONNECTION_FAILED	Connection to RM '%s' failed. Not defer-ring job '%s' (Reason: %s).	Refer to the reason message.
0x100-a9a3	ADMI-N	sys-tem.-moab	INFO	MWM_DEFER_DISABLED	Defer disabled.	The job cannot be deferred.
0x100-a9a4	ADMI-N	sys-tem.-moab	INFO	MWM_MWS_CLUSTER_QUERY	Cluster query retrieval failed for MWS RM '%s'.	The resource man-ager did not respond to the request.
0x100-a9a5	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_INVALID_PARTITION	Job '%s' specifies an invalid partition.	The job must ref-erence a valid par-tition.
0x100-a9a6	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_INVALID_QOS	Cannot set QoS on job '%s' to '%s' - invalid QoS.	The job must use a valid QoS.
0x100-a9a7	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_INVALID_ACCOUNT	Cannot set account on job '%s' to '%s' - invalid account (%s).	The job must use a valid account.
0x100-a9a8	ADMI-N	sys-tem.-moab	INFO	MWM_CHECKING_IDLE_JOB	Checking idle job '%s' (priority: %s) partition %s.	Checking the job.
0x100-a9a9	ADMI-N	sys-tem.-moab	INFO	MWM_CHECKING_SUSPENDED_JOB	Checking suspended job '%s' (priority: %s).	Checking the job.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-a9aa	ADMIN	system.moab	INFO	MWM_CHECKPOINT_TEST_ENABLED	Checkpoint test enabled (env).	The feature has been enabled with an environment variable.
0x100-a9ab	ADMIN	system.moab	INFO	MWM_ADD_NODE_FAILED	Could not add node because MNodeAdd failed.	The node could not be added to the object.
0x100-a9ac	ADMIN	system.moab	INFO	MWM_ATTEMPTING_RESERVATION	Attempting reservation of %s procs in %s for %s.	The scheduler will try to make the reservation.
0x100-a9ad	ADMIN	system.moab	INFO	MWM_FAIRSHARE_INTERVAL	Fairshare rolled to interval %s.	The interval has changed.
0x100-a9ae	ADMIN	system.moab	INFO	MWM_INVALID_ARCHITECTURE	Invalid architecture.	The architecture is not a valid value.
0x100-a9af	ADMIN	system.moab	INFO	MWM_INVALID_PSEUDOJOB	Invalid pseudo-job.	The pseudo-job is not a valid value.
0x100-a9b0	ADMIN	system.moab	INFO	MWM_HOLD_TYPE	Hold type '%s' selected.	The given hold type was specified.
0x100-a9b1	ADMIN	system.moab	INFO	MWM_MESSAGE_SENT	Message sent to server.	The message was sent.
0x100-a9b2	ADMIN	system.moab	INFO	MWM_JOB_LOCATED	Located job '%s' in partition '%s' reserved to start %s.	The specified job has been located.
0x100-a9b3	ADMIN	system.moab	INFO	MWM_TOTAL_JOBS_DETECTED	Total jobs detected: %s.	Number of counted jobs returned from the workload query.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-a9b4	ADMIN	system.moab	INFO	MWM_NO_WORKLOAD_DETECTED	No workload reported by any RM.	No jobs were reported across all the resource manager queries.
0x100-a9b5	ADMIN	system.moab	INFO	MWM_LOADING_JOB	Loading job '%s' in state '%s' (%s bytes).	The job is being loaded.
0x100-a9b6	ADMIN	system.moab	INFO	MWM_JOB_START_REJECTED	Local constraints rejected the starting of job '%s'.	The job cannot start.
0x100-a9b7	ADMIN	system.moab	INFO	MWM_INVALID_STAT_TYPE	Invalid stat type '%s' requested.	Not a valid value.
0x100-a9b8	ADMIN	system.moab	INFO	MWM_ORPHAN_PARTITION	Creating temporary job to process orphan partition '%s' for job '%s'.	The job was not found in active or completed job tables.
0x100-a9b9	ADMIN	system.moab	INFO	MWM_DISABLING_ACTION_PROGRAM	Disabling action program '%s'.	An invalid action program was requested.
0x100-a9ba	ADMIN	system.moab	INFO	MWM_DISABLING_JOB_FB_PROGRAM	Disabling job feedback program '%s' (%s).	An invalid job feedback program was requested. See documentation for FEEDBACKPROGRAM.
0x100-a9bb	ADMIN	system.moab	INFO	MWM_CP_RESTART_STATE_IGNORED	Checkpoint restart state '%s' ignored.	The restart state specified is being ignored.
0x100-a9bc	ADMIN	system.moab	INFO	MWM_CP_RESTART_STATE_SUCCESS	Starting scheduler with checkpoint restart state '%s'.	The restart state specified is being used.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-a9bd	ADMI-N	sys-tem.-moab	INFO	MWM_DESTROYING_NODE	Destroying node '%s'.	The specified node is being destroyed.
0x100-a9be	ADMI-N	sys-tem.-moab	INFO	MWM_IGNORING_NODE	Ignoring node '%s'.	The specified node is being ignored.
0x100-a9c0	ADMI-N	sys-tem.-moab	INFO	MWM_CANNOT_ADJUST_JOB_HOLDS	Cannot adjust holds on remote peer for job '%s' (%s).	Unable to modify the job.
0x100-a9c1	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_CANNOT_CREATE_RESERVATION	Cannot create reser-vation for job '%s' (previously reserved to start in %s)).	Failed to create reser-vation for job.
0x100-a9c2	ADMI-N	sys-tem.-moab	INFO	MWM_TRIGGER_LOAD_OUTPUT	Cannot load output data for trigger '%s' (File: %s).	The file may not exist or may be inac-cessible.
0x100-a9c3	ADMI-N	sys-tem.-moab	INFO	MWM_PBS_SERVER_CONNECT	Connected to PBS server %s:%s on sd %s.	Connection estab-lished.
0x100-a9c4	ADMI-N	sys-tem.-moab	INFO	MWM_NO_JOB_DATA	No job data was sent by %s RM.	The data sent by the resource manager did not contain job information.
0x100-a9c5	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_RESUMED_WITH_PROCS	Job '%s' resumed on %s processors.	The resource man-ager resumed the job.
0x100-a9c6	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_SIGNALED	Job %s' successfully signaled (action: %s, signal: %s).	The job responded to the signal request.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-a9c7	ADMIN	system.moab	INFO	MWM_JOB_CANCELED_RM	Job '%s' canceled through %s RM.	The job was canceled.
0x100-a9c8	ADMIN	system.moab	INFO	MWM_JOB_ASSIGNED_DEFAULT_GROUP	Job '%s' assigned default group '%s'.	The job was modified.
0x100-a9c9	ADMIN	system.moab	INFO	MWM_FILE_EXECUTE_PERMISSION	File '%s' does not have user execute permission (st_mode = %s).	The permissions must be modified.
0x100-a9ca	ADMIN	system.moab	INFO	MWM_INSUFFICIENT_PREEMPT_JOBS	Inadequate preempt jobs (%s) located for %s job (P: %s of %s, N: %s of %s).	Not enough jobs could be preempted.
0x100-a9cb	ADMIN	system.moab	INFO	MWM_READ_STAT_INDEX	Cannot read stat index for location %s:%s:%s.	The checkpoint did not have the stat information.
0x100-a9cc	ADMIN	system.moab	INFO	MWM_BACKFULL_JOB_PREEMPT	Backfill job '%s' no longer preemptible (%s > %s) in partition '%s'.	The job cannot be preempted.
0x100-a9cd	ADMIN	system.moab	INFO	MWM_STARTTIME_UNAVAILABLE	Cannot obtain desired starttime (%s != %s).	The job cannot be adjusted to the given start time.
0x100-a9ce	ADMIN	system.moab	INFO	MWM_STARTTIME_ADJUSTED	Timeframe for reservation %s adjusted forward by %s seconds.	The reservation has been adjusted to the given start time.
0x100-a9cf	ADMIN	system.moab	INFO	MWM_RESERVATION_ROLLBACK	Time: %s RollbackOffset: %s RsvStartTime: %s RsvDuration %s.	The reservation is being considered for rollback.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-a9d0	ADMI-N	sys-tem.-moab	INFO	MWM_RESERVATION_NOT_REQUIRED	Reservation '%s' not required for spe-cified period.	The reservation is not required for this time period.
0x100-a9d1	ADMI-N	sys-tem.-moab	INFO	MWM_RM_INTERFACE_RECOVERED	The interface for RM '%s' has been recovered.	A previously corrupt interface is now work-ing.
0x100-a9d2	ADMI-N	sys-tem.-moab	INFO	MWM_NTR_JOB_FOUND	Found an NTR (next to run) job - stop-ping idle job schedul-ing.	The job will now be run.
0x100-a9d3	ADMI-N	sys-tem.-moab	INFO	MWM_GRES_KEYWORD_NO_VALUE	GRes keyword '%s' passed in with no value.	A value must be spe-cified.
0x100-a9d4	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_EXTENSION_STRING	Job '%s' has invalid extension string - '%s'.	The system is unable to process the string.
0x100-a9d5	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_PROCESS_FAILURE	Job '%s' is invalid. It cannot be processed (%s).	There was an error loading the job. It will be rejected.
0x100-a9d6	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_MODIFIED_RM	Job '%s' has been modified through %s RM.	The job was mod-ified.
0x100-a9d7	ADMI-N	sys-tem.-moab	INFO	MWM_IDLE_BACKLOG_SIZE	Idle backlog: %s seconds (%s hours).	The idle backlog status is given.
0x100-a9d8	ADMI-N	sys-tem.-moab	INFO	MWM_SET_RESOURCES	Inadequate resources found in any set (%s < %s).	None of the node sets have the resources needed.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-a9d9	ADMIN	system.moab	INFO	MWM_JOB_PREEMPTING_JOB	Job %s preempting job %s (statemtime: %s) (preempted this iteration: %s).	One job preempted another.
0x100-a9da	ADMIN	system.moab	INFO	MWM_UPDATE_SCHEDULER_STATS	Iteration: %s; scheduling time: %s seconds.	Normal statistics update.
0x100-a9db	ADMIN	system.moab	INFO	MWM_JOB_STARTED_RM	Job '%s' started through %s RM on %s procs.	The job has started.
0x100-a9dc	ADMIN	system.moab	INFO	MWM_JOB_DELAY	Job delay: %s; reservation retry time: %s (StateDelayNC: %s; JobRsvDelayNC: %s).	The job has been delayed.
0x100-a9dd	ADMIN	system.moab	INFO	MWM_JOB_COMPLETED	Job '%s' completed. X: %s; T: %s; PS: %s; A: %s (RM: %s/%s).	The job completed.
0x100-a9de	ADMIN	system.moab	INFO	MWM_JOB_RESERVED_TASKS	Job '%s' reserved %s tasks (partition %s) to start in %s on %s (WC: %s).	The job has reserved the tasks.
0x100-a9df	ADMIN	system.moab	INFO	MWM_EVENT_INTERFACE_ENABLED	Event interface enabled for wiki RM %s on port %s.	The interface is now functional.
0x100-a9e0	ADMIN	system.moab	INFO	MWM_RM_RESOURCES_DETECTED	There were %s %s resources detected on RM '%s'.	The given resources were found.
0x100-a9e3	ADMIN	system.moab	INFO	MWM_EXTENDING_RESERVATION	Extending reservation by %s seconds (trigger still active).	The reservation is being extended.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-a9e4	ADMI-N	sys-tem.-moab	INFO	MWM_EXTENDING_RESERVATION_OVERRUN_JOB	Extending reser-vation for overrun job '%s' by %s seconds.	The reservation is being extended.
0x100-a9e5	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_LOCATED_BESTFIT	Located bestfit job '%s' (size: %s; dur-ation: %s).	Backfill found a job that best fits the available resources.
0x100-a9e6	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_BEST_PARTITION	The best partition for job '%s' is '%s'.	Backfill found a job that best fits the available resources.
0x100-a9e7	ADMI-N	sys-tem.-moab	INFO	MWM_CPA_PARTITION_DESTROY	Destroying CPA par-tition' %s' for job '%s' with cookie %s (%s).	The partition is being destroyed.
0x100-a9e8	ADMI-N	sys-tem.-moab	INFO	MWM_RESOURCES_LOCATED	Located resources for %s tasks (%s) in best partition '%s' for job '%s' at time offset %s.	The listed resources have been located.
0x100-a9e9	ADMI-N	sys-tem.-moab	INFO	MWM_MINIMUM_EFFICIENCY_REACHED	Minimum efficiency reached (%s per-cent) on iteration %s.	The threshold has been reached.
0x100-a9ea	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_START_PARTITION	Cannot start job '%s' in partition '%s' (scheduler mode: %s).	The job could not be started.
0x100-a9eb	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_FEASIBLE_NODES	Inadequate feasible nodes found for job '%s':%s in partition '%s' (%s < %s).	The job could not be scheduled.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-a9ec	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_LOADED	Job '%s' loaded: TC=%s UGC-C=%s,%s,%s WC=%s ST=%s %s %s.	The job was loaded.
0x100-a9ed	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_RSV_CREATE	Cannot create new reservation for job %s (shape[%s] %s).	Check the reser-vation time, nodes, and account.
0x100-a9ee	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_LOCATE_NODES	Cannot locate nodes for job '%s' req[%s] (%s additional needed).	Not enough nodes are available to run the job.
0x100-a9ef	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_START_RM_DISABLED	Cannot start job '%s' since RM '%s' is dis-abled.	Not enough nodes are available to run the job.
0x100-a9f0	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_START_RESERVE_TIME	Cannot start job '%s' reserve time in %s.	The time to schedule has already arrived.
0x100-a9f1	ADMI-N	sys-tem.-moab	INFO	MWM_ERROR_IN_EXE_STDERR	Error detected in '%s' due to presence of the word 'ERROR' in stderr (%s).	The executable failed.
0x100-a9f2	ADMI-N	sys-tem.-moab	INFO	MWM_ERROR_IN_STDERR	Error detected due to presence of the word 'ERROR' in stderr.	The child process failed.
0x100-a9f3	ADMI-N	sys-tem.-moab	INFO	MWM_CHECKJOB_STATE	Job '%s' State: %s Expected State: %s QueueTime: %s.	The job is in the lis-ted state. The expec-ted state may not be the same.
0x100-a9f4	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_NODELIST	Cannot obtain nodel-ist for job '%s':%s in range %s.	The nodes are not available.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-a9f5	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_RESUME	Job '%s' cannot be resumed since alloc-ated nodes are not available (node '%s' state '%s').	The resource man-ager resumed the job.
0x100-a9f6	ADMI-N	sys-tem.-moab	INFO	MWM_CLEARING_EXPIRED_RESERVATION	Clearing expired %s reservation '%s' on iteration %s (start: %s end: %s).	The reservation has expired.
0x100-a9f7	ADMI-N	sys-tem.-moab	INFO	MWM_CPA_RETRY	CPA retry detected - will re-attempt par-tition creation in 2 seconds.	The partition may be created.
0x100-a9f9	ADMI-N	sys-tem.-moab	INFO	MWM_JOBS_STARTED	There were %s %s jobs started in par-tition '%s' on iter-ation %s.	The jobs were star-ted.
0x100-a9fa	ADMI-N	sys-tem.-moab	INFO	MWM_TASKS_ALLOCATED	There were %s of %s tasks allocated for job '%s':%s.	The tasks were alloc-ated.
0x100-a9fb	ADMI-N	sys-tem.-moab	INFO	MWM_CLASSES_DETECTED	There were %s %s classes/queues detected on RM '%s'.	The classes were detected.
0x100-a9fd	ADMI-N	sys-tem.-moab	INFO	MWM_JOB_DELAYED_RSV	Delayed reservation detected for reserved job '%s' (%s seconds) attempting squeeze.	Attempting to fit the job into the reser-vation.
0x100-a9fe	ADMI-N	sys-tem.-moab	INFO	MWM_DUMPING_RESERVATIONS	Dumping reser-vations on iteration %s.	All the reservations will be dumped to the log.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-a9ff	ADMI-N	sys-tem.-moab	INFO	MWM_ALLOCPARTITIO-N_MISSING	ALLOCPARTITION missing from com-pleted job '%s' - restoring variable with value '%s'.	The value is being substituted.
0x100-aa00	ADMI-N	sys-tem.-moab	INFO	MWM_RECEIVED_NODELIST	Received nodelist through %s RM.	The nodelist was received.
0x100-aa01	ADMI-N	sys-tem.-moab	INFO	MWM_SERVICE_REQUEST_FROM_HOST	Received service request from host '%s'.	The request was received.
0x100-aa02	ADMI-N	sys-tem.-moab	INFO	MWM_RECEIVED_WORKLOAD	Received workload info through %s RM '%s' (%s bytes).	The workload was received.
0x100-aa03	ADMI-N	sys-tem.-moab	INFO	MWM_RSV_REMOVED_FROM_CACHE	Removing reser-vation '%s' from cache.	The cached reser-vation is being removed.
0x100-aa04	ADMI-N	sys-tem.-moab	INFO	MWM_RECOVER_READ_SOCKET	RECOVER: attempt-ing to read socket connection.	The recovery func-tion is attempting to communicate via sockets.
0x100-aa05	ADMI-N	sys-tem.-moab	INFO	MWM_GREEDY_BACKFILL	Improved list found by greedy backfill in %s searches (utility: %s; processors avail-able: %s).	The object is being removed from the database.
0x100-aa06	ADMI-N	sys-tem.-moab	INFO	MWM_RESERVATION_NAME_AND_GROUP	Name='%s' RsvGroup='%s'.	The object is being removed from the database.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x100-aa15	ADMI-N	sys-tem.-moab	INFO	MWM_TRIGGERS_DISABLED	Triggers disabled. %s.	Triggers are disabled. This message indicates when this flag is being set, and when an action is being skipped because the flag is set.
0x100-aa16	ADMI-N	sys-tem.-moab	INFO	MWM_USER_NOT_AUTHORIZED	User %s is not authorized to %s.	This user does not have permissions to accomplish the listed task.
0x100-aa1f	ADMI-N	sys-tem.-moab	INFO	MWM_SUCCESSFULLY_OPENED_SOCKET	Opened service socket on port %s.	A socket was sucessfully opened listeneing on the remote port.
0x100-aa21	ADMI-N	sys-tem.-moab	INFO	MWM_NO_CHECKPOINT_INFO	No checkpoint information available for '%s'.	Checkpoint information was not available.
0x100-aa22	ADMI-N	sys-tem.-moab	INFO	MWM_NODE_INDEX_TABLE_ENABLED	Node index table enabled.	Enabled by environment variable: MOABUSENODEINDEX.
0x100-aa24	ADMI-N	sys-tem.-moab	INFO	MWM_UNKNOWN_NODE_SLOT	Node slot not yet set on node '%s'.	Delaying setting rack until slot is known.
0x100-aa25	ADMI-N	sys-tem.-moab	INFO	MWM_NODE_STATUS	Node '%s' status: state='%s' rsvlist='%s' joblist='%s'.	General node status.
0x100-aa26	ADMI-N	sys-tem.-moab	INFO	MWM_NO_JOBS_IN_QUEUE	No jobs in queue.	There were no jobs in the scheduler queue indicating the scheduler has nothing to process.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-aa27	ADMIN	system.moab	INFO	MWM_NO_NODE_DATA	No node data sent by %s RM.	The resource manager did not receive any node data in cluster query.
0x100-aa28	ADMIN	system.moab	INFO	MWM_NO_PREEMPTIBLE_RESOURCES	No preemptible resources found for job %s (tc: %s; class: '%s'; qos: %s; priority: %s; partition %s.	Indicates the scheduler could not find any jobs for preemption.
0x100-aa29	ADMIN	system.moab	INFO	MWM_NO_PRIORITY_RESERVATION_CREATED_FOR_POLICY	No priority reservations created for policy '%s' for job %s.	Job reservation for a specific policy was unable to be created.
0x100-aa2a	ADMIN	system.moab	INFO	MWM_NO_QUEUES_DETECTED	No queues detected.	Resource manager attempted to obtain queue information. Check resource manager for configured queues.
0x100-aa2b	ADMIN	system.moab	INFO	MWM_NOT_ADDING_RM	Not adding RM '%s'.	The partition is not adding the specified resource manager. This situation is most common in grid configurations where resource manager names are similiar.
0x100-aa30	ADMIN	system.moab	INFO	MWM_ORPHAN_PARTITION_REPORTED_FOR_JOB	Orphan partition %s reported for job %s. %s.	The resource manager reported the partition as orphaned.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x100-aa31	ADMIN	system.moab	INFO	MWM_PARAMETER_CANNOT_BE_CHANGED	Parameter '%s' cannot be changed while Moab is running.	Configuration file must be changed, and Moab must be restarted.
0x100-e72f	INTERNAL	system.moab	INFO	MWM_SOCKET_REMOTE_DISCONNECT	Reading from a socket failed. It appears the client disconnected, errno: %s (%s).	The recv() system call failed. Use the errno and associated message to determine possible causes.
0x100-e7f0	INTERNAL	system.moab	INFO	MWM_VM_LINKED_TO_NEW_TRACKING_JOB	Setting VMTracking job for VM '%s' to job '%s.	A VM is associated with a tracking job.
0x110-001f7	USER	domain.lifecycle	WARN	MWM_TRIG_FAILURE	Trigger %s has failed.	The named trigger has finished its action, but it returned with a failure status.
0x110-0025e	USER	domain.lifecycle	WARN	MWM_VM_MIGRATE_END_ERROR	VM %s migration has finished with an error: (%s)	The named VM has finished its migration. There was a problem during the migration. Additional information may be provided regarding the error specifics.
0x110-00261	USER	domain.lifecycle	WARN	MWM_VM_MIGRATE_SUBMIT	Failed to submit VM migration job for VM %s. (%s)	Failed to submit VM Migration job.
0x110-00262	USER	domain.lifecycle	WARN	MWM_VM_NO_FEASIBLE_NODES	Failed to find a feasible node/hypervisor on which to run VM %s. Check setup job %s for details.	The named VM has been submitted, but no node/hypervisor meets all requirements.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-00384	USER	sys-tem.-moab	WAR-N	MWM_VM_LICENSE_ERROR	There is an error with the Moab license: (%s)	There was a licensing error. Additional information may be provided regarding the error specifics.
0x110-0284a	USER	sys-tem.-moab	WAR-N	MWM_BAD_COMMANDLINE_FLAG	Unexpected flag detected: '%s'.	The command line syntax that was received contains an invalid flag. Check the documentation and retry.
0x110-02858	USER	sys-tem.-moab	WAR-N	MWM_NODESETMAXU-SAGE_FAILURE	Ignoring incorrect NODESETMAXUSAGE value '%s'.	Valid range is from 0.0 to 1.0 inclusive.
0x110-02883	USER	sys-tem.-moab	WAR-N	MWM_CANNOT_DESTROY_STATIC_RM	Trigger cannot des-troy static RM.	A trigger cannot des-troy a static resource manager. Refer to trigger 'destroy'.
0x110-02966	USER	sys-tem.-moab	WAR-N	MWM_RM_JOB_SUBMIT_FAILURE	RM %s job submit failed: %s.	Error while sub-mitting the job to the resource manager.
0x110-02a0c	USER	sys-tem.-moab	WAR-N	MWM_CANNOT_MODIFY_RM_JOB	Cannot modify %s for RM job %s - '%s'.	The listed attribute of the job could not be changed.
0x110-04004	POWE-R_USER	sys-tem.-moab	WAR-N	MWM_TESTING_WARNING	Testing with argu-ment1: %s. and argu-ment2: %s and argument3: %s	Internal error for testing diagnostics.
0x110-0a713	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_LOAD_FILE	Cannot load %s file %s.	Failed to load a file into Moab. Make sure it exists and that per-missions are correct.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a71a	ADMI-N	sys-tem.-moab	WAR-N	MWM_FAILED_TO_WAIT_FOR_CHILD	Failed to wait for child, pid: %s, errno: %s (%s).	The wait() system call failed. Use the errno and associated message to determ-ine possible causes.
0x110-0a71d	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_CHMOD_FILE	Failure changing per-missions of file: '%s' to mode:'%s', errno: %s (%s).	The chmod() system call failed. Use the errno and associated message to determ-ine possible causes.
0x110-0a71e	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_OPEN_FILE_WARNING	Cannot open %s file '%s', errno: %s (%s).	The fopen() system call failed. Use the errno and associated message to determ-ine possible causes.
0x110-0a722	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_WRITE_FILE_WARNING	Cannot write to file '%s', errno: %s (%s).	The fwrite() system call failed. Use the errno and associated message to determ-ine possible causes.
0x110-0a723	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_CLOSE_FILE_DESCRIPTOR	Cannot close file descriptor %s, errno: %s (%s).	The close() system call failed. Use the errno and associated message to determ-ine possible causes.
0x110-0a724	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_RENAME_FILE	Failure renaming file '%s' to '%s', errno: %s (%s).	The rename() system call failed. Use the errno and associated message to determ-ine possible causes.
0x110-0a726	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_BIND_TO_PORT	Cannot bind to port %s, errno: %s (%s).	The bind() system call failed. Use the errno and associated message to determ-ine possible causes.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a72a	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_SEND_TO_SOCKET	Cannot send %s byte packet, errno: %s (%s).	The send() system call failed. Use the errno and associated message to determ-ine possible causes.
0x110-0a72c	ADMI-N	sys-tem.-moab	WAR-N	MWM_FAILED_GETSOCKOPT_WARNING	Cannot get socket %s option, errno: %s (%s).	The getsockopt() sys-tem call failed. Use the errno and asso-ciated message to determine possible causes.
0x110-0a72d	ADMI-N	sys-tem.-moab	WAR-N	MWM_FAILED_SETSOCKOPT_WARNING	Cannot set socket %s option, errno: %s (%s).	The setsockopt() sys-tem call failed. Use the errno and asso-ciated message to determine possible causes.
0x110-0a738	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_SET_UMASK	Failure setting umask on file '%s', errno: %s (%s).	The umask() system call failed. Use the errno and associated message to determ-ine possible causes.
0x110-0a73a	ADMI-N	sys-tem.-moab	WAR-N	MWM_FAILED_FCNTL_WARNING	Cannot set %s option on file descriptor, errno: %s (%s).	The fcntl() system call failed. Use the errno and associated message to determ-ine possible causes.
0x110-0a73b	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_STAT_FILE_WARNING	Cannot get stats on file '%s', errno: %s (%s).	The stat() system call failed. Use the errno and associated mes-sage to determine possible causes.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x110-0a73e	ADMIN	system.moab	WARN	MWM_GET_HOSTNAME_CLIENT	Cannot get hostname of the client, errno: %s (%s).	The getnameinfo() system call failed. Use the errno and associated message to determine possible causes.
0x110-0a74d	ADMIN	system.moab	WARN	MWM_CONFIG_FILE_NOT_FOUND_WARNING	Cannot locate configuration file '%s' in '%s'.	Check for the existence of this file.
0x110-0a753	ADMIN	system.moab	WARN	MWM_RM_CONFIG_INVALID_VALUE	Invalid %s value '%s' specified for RM '%s'.	Check the line in the configuration file for the parameter.
0x110-0a754	ADMIN	system.moab	WARN	MWM_RM_CONFIG_PROCESS_ATTR	Failed to process attribute '%s' for resource manager '%s'.	Check the line in the configuration file for the parameter.
0x110-0a755	ADMIN	system.moab	WARN	MWM_RM_CONFIG_ATTR	RM attribute '%s' not handled.	Check the line in the configuration file for the parameter.
0x110-0a756	ADMIN	system.moab	WARN	MWM_RM_CONFIG_TIMEOUT	Resource manager '%s' has a timout of less than 50 ms.	Check the line in the configuration file for the parameter.
0x110-0a758	ADMIN	system.moab	WARN	MWM_CONFIG_PARAM_DEFAULT_VALUE	Configuration parameter '%s[%s]' was not assigned a value. Using default.	Check the line in the configuration file to see if this behaviour is desired.
0x110-0a759	ADMIN	system.moab	WARN	MWM_CONFIG_PARAM_INTEGER_DEFAULT_VALUE	Configuration parameter '%s[%s]' has a value '%s' that is not an integer. Using default.	Check the line in the configuration file for the integer value.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x1100a75a	ADMIN	system.moab	WARN	MWM_CONFIG_PARAM_DOUBLE_DEFAULT_VALUE	Configuration parameter '%s[%s]' has a value '%s' that is not a double. Using default.	Check the line in the configuration file for the double value.
0x1100a75b	ADMIN	system.moab	WARN	MWM_CONFIG_PARAM_NULL_VALUE	Configuration parameter '%s[%s]' has a NULL value.	Check the line in the configuration file.
0x1100a75c	ADMIN	system.moab	WARN	MWM_CONFIG_PARAM_INVALID	Configuration parameter '%s' has an invalid value.	Check the line in the configuration file for the attribute.
0x1100a75e	ADMIN	system.moab	WARN	MWM_CONFIG_PARAM_UNKNOWN	Configuration parameter '%s[%s]' is not defined.	Check the line in the configuration file for the undefined parameter.
0x1100a75f	ADMIN	system.moab	WARN	MWM_CONFIG_ATTR_EXTRACTION	Configuration parameter '%s[%s]' attribute value '%s' cannot be extracted.	Check the line in the configuration file for the attribute.
0x1100a760	ADMIN	system.moab	WARN	MWM_INVALID_CONFIG_LINE	Cannot process line '%s'.	Check the line syntax against the documentation.
0x1100a763	ADMIN	system.moab	WARN	MWM_UNKNOWN_ADMINCFG_PARAMETER	Unknown ADMINCFG parameter '%s'.	Check the syntax in the configuration file.
0x1100a764	ADMIN	system.moab	WARN	MWM_UNKNOWN_MID_ATTR	Unknown identity attribute '%s'.	Check the MIDCFG lines in the configuration file.
0x1100a765	ADMIN	system.moab	WARN	MWM_UNKNOWN_AM_ATTR	Unknown account manager attribute '%s'.	Check the AMCFG lines in the configuration file.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a766	ADMI-N	sys-tem.-moab	WAR-N	MWM_ UNKNOWN_ ATTRIBUTE_ SPECIFIED	Unknown attribute '%s' specified for %s %s.	An error occurred while parsing the con-figuration for the lis-ted object. The specified attribute is unknown or invalid.
0x110-0a769	ADMI-N	sys-tem.-moab	WAR-N	MWM_ MONGOSERVER_ CONNECTION_ FAILURE	Unable to connect to Mongo server '%s' (%s).	The program will con-tinue to try and con-nect in the background.
0x110-0a76e	ADMI-N	sys-tem.-moab	WAR-N	MWM_EVENT_ QUERY_ODBC	Event querying is only supported with ODBC.	Check the USEDATABASE option.
0x110-0a770	ADMI-N	sys-tem.-moab	WAR-N	MWM_DB_ CONNECT	Cannot connect to DB--falling back to file and memory-based storage (%s).	Verify that the data-base is running.
0x110-0a772	ADMI-N	sys-tem.-moab	WAR-N	MWM_ DATABASE_ STATS	Unable to retrieve statistics from the database.	Verify that the data-base is running.
0x110-0a774	ADMI-N	sys-tem.-moab	WAR-N	MWM_SERVER_ CONNECTION_ FAILED_TRYING_ FALLBACK	The system was unable to connect to the server %s:%s - attempting fallback server %s.	Make sure the server's address is correct and it is run-ning.
0x110-0a775	ADMI-N	sys-tem.-moab	WAR-N	MWM_ PRIMARY_ SERVER_FAILED_ TRYING_ BACKUP	The system was unable to connect to the server %s (%s:%s) - trying backup server (%s:%s).	Make sure the server's address is correct and it is run-ning.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a778	ADMI-N	sys-tem.-moab	WAR-N	MWM_INVALID_REDUCE_CLIENTMAXCON-NECTIONS	Reducing CLIENTMAXCONNEC-TIONS to %s from %s not allowed dur-ing runtime.	Decreasing the value of CLIENTMAXCONNEC-TIONS cannot be done during runtime.
0x110-0a785	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNABLE_TO_START_JOB	Cannot start job %s. (%s)	The job failed to start.
0x110-0a788	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNABLE_TO_ALLOCATE_NODES_FOR_JOB	Cannot allocate nodes for job %s. (%s)	Cannot allocate a node list that matches the require-ments for this job. This may not be ser-ious since multiple passes may occur.
0x110-0a7d2	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNABLE_LOAD_PBS_JOB	Cannot load PBS job '%s'.	Could not load a job discovered from a PBS resource man-ager into Moab.
0x110-0a7e3	ADMI-N	sys-tem.-moab	WAR-N	MWM_TRUNCATING_ATTRIBUTE_FOR_CLASS	Truncating %s for class: %s (rm reports: %s; Moab enforces: %s).	The resource man-ager reports a certain value for a class, but Moab has been instructed to keep it within certain limits. The value will be truncated to keep it within the limits.
0x110-0a7e4	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNEXPECTED_JOB_STATE	Unexpected job state '%s' detected for job %s.	The listed job was found to be in a state that was not expec-ted. This may or may not be an error con-dition.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a821	ADMI-N	sys-tem.-moab	WAR-N	MWM_CLOCK_SKEW_DETECTED	Clock skew detected (%s time for job %s in %s).	A reported time asso-ciated with the job appears to be wrong. This could be because of a lack of synchronization between system clocks on all nodes.
0x110-0a83e	ADMI-N	sys-tem.-moab	WAR-N	MWM_INVALID_WIKI_ATTRIBUTE	Encountered invalid wiki attribute while reading '%s'.	Check the syntax of the attribute.
0x110-0a83f	ADMI-N	sys-tem.-moab	WAR-N	MWM_DUPLICATE_WIKI_ATTRIBUTE	Wiki attribute '%s' is already set.	Check for duplicate instances of the attribute.
0x110-0a840	ADMI-N	sys-tem.-moab	WAR-N	MWM_VM_UNSUPPORTED_WIKI_ATTRIBUTE	Wiki attribute '%s' is unsupported for VM creation.	Remove the attribute.
0x110-0a843	ADMI-N	sys-tem.-moab	WAR-N	MWM_ADD_NODE_FAILURE	Cannot add node '%s' to global node table. Index is already used.	Cannot have two nodes with the same name.
0x110-0a844	ADMI-N	sys-tem.-moab	WAR-N	MWM_HT_ADD_NODE_FAILURE	Cannot add node '%s' to hash table. Index is already used.	Cannot have two nodes with the same name.
0x110-0a846	ADMI-N	sys-tem.-moab	WAR-N	MWM_VM_MIGRATION_FAILURE	Cannot migrate VM '%s'.	The VM might not be eligible for migration.
0x110-0a84c	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNEXPECTED_SUBCOMMAND_RECEIVED	Unexpected sub-command '%s' received.	The communication from a Moab client includes an unknown subcommand.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a84d	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNABLE_TO_REGISTER_JOB_AM	Unable to register job %s with account-ing manager for job %s. reason: '%s' mes-sage:'%s'.	The accounting man-ager was unable to register the listed job for a certain action. An optional reason and/or message may be given to assist in diagnosis.
0x110-0a851	ADMI-N	sys-tem.-moab	WAR-N	MWM_IMPROPER_VM_MIGRATION_DECISION	The migration decision for the VM was not properly set up.	The information indicating the des-tination node is miss-ing.
0x110-0a85b	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_UPDATE_NULL_STARTTIME	Start time is NULL for job update.	Specify a start time that is greater than zero.
0x110-0a85c	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_UPDATE_NULL_DISPATCHTIME	Dispatch time is NULL for job update.	Specify a dispatch time that is greater than zero.
0x110-0a862	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNABLE_TO_REGISTER_RESERVATION_AM	Unable to register reservation %s with accounting manager for %s processors for reservation %s.	The accounting man-ager was unable to register the listed reservation for a cer-tain action.
0x110-0a863	ADMI-N	sys-tem.-moab	WAR-N	MWM_DEPRECATED_PARAMETER_VALUE	Deprecated value '%s' specified for parameter '%s'. %s	The listed value is no longer valid for this parameter. A hint may be provided with the message. Check the most recent doc-umentation for the software version.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x1100a864	ADMIN	system.moab	WARN	MWM_INVALID_FILE_ATTRIBUTES_WARNING	Invalid value '%s' specified for %s (%s).	Checking a file to see whether it exists, is executable, etc, has produced unexpected results.
0x1100a865	ADMIN	system.moab	WARN	MWM_CANNOT_SET_JOB_ATTRIBUTE_VIA_TEMPLATE_WARNING	Cannot set %s %s via template %s.	Failed to set the listed attribute to the listed value for a specified job template.
0x1100a866	ADMIN	system.moab	WARN	MWM_KILL_PROCESS_FAILURE	Unable to kill process %s.	The system tried to kill the given process and failed.
0x1100a86a	ADMIN	system.moab	WARN	MWM_NODELIST_STRING_BUFFER	Insufficient buffer space to convert a node list into a string.	The buffer must be larger to hold all the nodes.
0x1100a86b	ADMIN	system.moab	WARN	MWM_MAX_NODES_EXCEEDED	The maximum number of nodes associated with a reservation has been exceeded.	The number of nodes must be reduced.
0x1100a871	ADMIN	system.moab	WARN	MWM_VC_ALREADY_ADDED	The virtual container '%s' is already an ancestor of VC '%s'.	Cannot create a circular chain, must maintain a hierarchical structure.
0x1100a872	ADMIN	system.moab	WARN	MWM_VC_REMOVAL_FAILURE	The virtual container '%s' cannot be removed.	This is an internal error.
0x1100a874	ADMIN	system.moab	WARN	MWM_RESERVATION_JOB_NOT_FOUND	Unable to find the job for reservation '%s'.	The host job for the reservation is NULL.

Code	Escal- ation level	Topic	Sev- erit- y	Event name	Message tem- plate	Comment
0x110-0a875	ADMI-N	sys-tem.-moab	WAR-N	MWM_SINGLE_USE_RESERVATION_DESTRUCTION	Unable to destroy a single-use reser-vation.	This is an internal error.
0x110-0a878	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNEXPECTED_JOB_SUBMISSION_POLICY	The system encountered an unexpected job sub-mission policy (%s).	The job submission policy did not match a defined policy.
0x110-0a879	ADMI-N	sys-tem.-moab	WAR-N	MWM_SIMULATION_JOB_RECORDS	Unable to simulate workload by creating job records (1000 attempts).	The system may be low on memory.
0x110-0a880	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_TRANSITION_XML_MESSAGE	Unable to add mes-sages to job '%s' transition XML.	The system may be low on memory.
0x110-0a884	ADMI-N	sys-tem.-moab	WAR-N	MWM_PBS_API_STALE	PBS API is stale - re-initializing.	Re-initializing the PBS environment.
0x110-0a885	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNABLE_TO_GET_PBS_QUEUE_INFO	Cannot process PBS queue info for RM %s (node %s) - no data available.	Unable to get any information on the PBS queues. Make sure that there was at least a queue set up in PBS.
0x110-0a886	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNABLE_TO_SET_JOB_ATTRIBUTE	Cannot set job '%s' attribute '%s:%s' to '%s' (rc: %s; '%s').	There was a problem while changing the job attribute and the error status was dis-played in rc.
0x110-0a887	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNABLE_TO_CONNECT_PBS_SCHEDULER	Cannot connect to PBS event/scheduler port %s.	Ensure the PBS scheduler is running and listening on the specified port.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a888	ADMI-N	sys-tem.-moab	WAR-N	MWM_NODE_UNUSABLE_NO_DISK	Idle node %s is unus-able (inadequate disk space in /var).	Ensure that the node has sufficient disk space.
0x110-0a889	ADMI-N	sys-tem.-moab	WAR-N	MWM_NODE_UNUSABLE_BAD_STATE	Node '%s' is unus-able in state 'NONE'.	The node has become unusable because of its state being NONE.
0x110-0a88a	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNABLE_TO_FIND_USERS_GID	Cannot locate OS GID information for user '%s' - ignoring user.	Moab was unable to find the GID of this user. Make sure that this user has a GID.
0x110-0a88b	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNABLE_TO_FIND_USERS_UID	Cannot locate OS information for user '%s' - ignoring user - %s.	Moab was unable to find the user on the system. Make sure that this user exists.
0x110-0a88c	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNABLE_TO_FIND_GID_LIST	Cannot locate OS group list inform-ation for user '%s' - ignoring user.	Moab was unable to find the group list for this user.
0x110-0a88d	ADMI-N	sys-tem.-moab	WAR-N	MWM_TIMEOUT	Command '%s' timed out, or wait failed after %s seconds.	Increasing the TIMEOUT settings in moab.cfg may help.
0x110-0a88e	ADMI-N	sys-tem.-moab	WAR-N	MWM_INSUFFICIENT_POLICIES	Insufficient policies specified; hpoli-cy=%s,spolicy=%s.	Please revise your policies along with their actions.
0x110-0a88f	ADMI-N	sys-tem.-moab	WAR-N	MWM_NO_STDOUT	Request succeeded with no stdout but stderr='%s'.	Typically there will also be stdout when there is stderr. Depending on the request this may be the intended result.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a890	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_CONNECT_WIKI	Cannot connect to Wiki event port %s.	Failure to connect to the Wiki event port.
0x110-0a891	ADMI-N	sys-tem.-moab	WAR-N	MWM_MISSING_COLON_STR	Colon delimiter not located in %s wiki string '%s...' in %s.	Check that the string contains the right format.
0x110-0a892	ADMI-N	sys-tem.-moab	WAR-N	MWM_ADD_DEPENDENCY_FAIL	Failed to add dependecies to job %s's submission.	There was a problem in adding the dependencies to the job.
0x110-0a893	ADMI-N	sys-tem.-moab	WAR-N	MWM_INVALID_PRIORITY_FUNCTION	Invalid priority func-tion '%s' on job '%s'.	The priority function applied to the job was invalid.
0x110-0a894	ADMI-N	sys-tem.-moab	WAR-N	MWM_MISSING_JOB_REQ	Invalid job '%s'; no requirements.	The job was invalid because it was miss-ing the requirements.
0x110-0a895	ADMI-N	sys-tem.-moab	WAR-N	MWM_MISSING_JOB_REQ_AT_INDEX	Invalid job %s; no requirement at index %s.	The job was invalid because an index was missing require-ments.
0x110-0a896	ADMI-N	sys-tem.-moab	WAR-N	MWM_INVALID_WIKI_STR_MISSING_EQUAL	Malformed wiki string '%s' - no '='.	The wiki string was missing an equal sign '='.
0x110-0a897	ADMI-N	sys-tem.-moab	WAR-N	MWM_INVALID_EMPTY_WIKI_STR	Malformed wiki string '%s' - EOF.	The wiki string was empty.
0x110-0a898	ADMI-N	sys-tem.-moab	WAR-N	MWM_INCORRECT_STAGE_LOC	stage-data source loc-ation is being incor-rectly reported via wiki '%s' != '%s'.	The stage data source location was incorrectly reported in wiki.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x110-0a899	ADMIN	system.moab	WARN	MWM_INVALID_VM_OBJECT_ID	VM '%s' is not a valid object, ignoring.	The VM does not have a valid object ID.
0x110-0a89a	ADMIN	system.moab	WARN	MWM_INVALID_JSON_CLUSTER	Could not parse JSON cluster query data from MWS RM (%s): %s.	The JSON construct (s) for the cluster may contain some invalid syntax.
0x110-0a89b	ADMIN	system.moab	WARN	MWM_INVALID_JSON_WORKLOAD	Could not parse JSON workload query data from MWS RM (%s): %s.	The JSON construct (s) for the workload may contain some invalid syntax.
0x110-0a89c	ADMIN	system.moab	WARN	MWM_MISSING_REQ_PROPERTIES	JSON cluster query data from MWS RM (%s) does not contain required properties (%s, %s, %s).	The JSON constructs for the cluster query data are missing the required properties.
0x110-0a89d	ADMIN	system.moab	WARN	MWM_INVALID_JSON_CLUSTER_OBJECT	JSON cluster query data from MWS RM (%s) is not a valid object.	Review the JSON construct for the cluster query data to ensure its syntax is correct.
0x110-0a89e	ADMIN	system.moab	WARN	MWM_INVALID_JSON_WORKLOAD_OBJECT	JSON workload query data from MWS RM (%s) is not a valid object.	Review the JSON construct for the workload query data to ensure its syntax is correct.
0x110-0a89f	ADMIN	system.moab	WARN	MWM_EMPTY_RESPONSE	Empty %s response from RM (%s).	The response from the resource manager query was empty.
0x110-0a8a0	ADMIN	system.moab	WARN	MWM_INVALID_NODE_DATA	Nodes data from MWS RM (%s) is not a valid object.	The response from the Moab Web Services resource manager query was empty.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a8a1	ADMI-N	sys-tem.-moab	WAR-N	MWM_CREATE_RESERVATION_FAIL	Cannot create requested reser-vation (%s).	The request to create the given reservation has failed.
0x110-0a8a2	ADMI-N	sys-tem.-moab	WAR-N	MWM_LOCATE_RSVPROFILE_FAIL	Cannot locate RSVPROFILE '%s'.	Moab failed to find the given RSVPROFILE. Confirm that the file exists.
0x110-0a8a3	ADMI-N	sys-tem.-moab	WAR-N	MWM_LOCATE_RSV_PARENT_FAIL	Cannot locate parent '%s' for reservation '%s'.	Moab failed to locate the parent of the given reservation.
0x110-0a8a4	ADMI-N	sys-tem.-moab	WAR-N	MWM_LOCATE_COMMAND_FAIL	Cannot locate com-mand '%s'.	Moab failed to locate the given command. Confirm that the com-mand exists.
0x110-0a8a5	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNSUPPORTED_SCHED_CMD	Received unexpected sched command '%s'.	Received an unex-pected mschedctl command. Confirm that the used option is supported.
0x110-0a8a6	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNSUPPORTED_EVENT	Unsupported event '%s' from RM '%s'.	The given event is not supported by the given resource man-ager.
0x110-0a8a7	ADMI-N	sys-tem.-moab	WAR-N	MWM_VM_MIGRATE_FAIL	VM %s should migrate from node %s but cannot locate valid destination - %s (policy).	Attempt to migrate the given VM from the given node failed. Please check that the destination is valid.
0x110-0a8a8	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_START_FAIL	Start of system job %s failed; no action specified.	Failed to start a job because there was no action specified.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a8a9	ADMI-N	sys-tem.-moab	WAR-N	MWM_VMTRACKING_JOB_FAIL	VM '%s' reported a system job failure on VMTracking job '%s'.	The given VM repor-ted it failed on the given VMTracking job.
0x110-0a8aa	ADMI-N	sys-tem.-moab	WAR-N	MWM_VMTRACKING_EXCEED_WALTIME	VM '%s' exceeded its allocated walltime. VMTracking job '%s' (pointing to job '%s').	The given VM has exceeded its alloc-ated walltime on the associated VM track-ing job.
0x110-0a8ac	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNKNOWN_POWER_STATE	No RM can report node '%s' power state for system job '%s'.	No resource manager can report the power state for the given nodes on the given job.
0x110-0a8ad	ADMI-N	sys-tem.-moab	WAR-N	MWM_ADD_GLOBAL_NODE_FAIL	Cannot add global node '%s'.	Failed to add the given global node.
0x110-0a8ae	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNKNOWN_CLIENT	Client ID '%s' is unknown.	Moab failed to recog-nize the name/ID of the given client.
0x110-0a8af	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_DEBIT_ACCOUNT	Unable to charge funds for job.	The account manager failed to debit the account for the job.
0x110-0a8b0	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_RESERVE_ACCOUNT	Unable to reserve funds for job (Reason: %s).	The account manager failed to reserve funds on the account for the job.
0x110-0a8b1	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANCEL_LEIN	Unable to cancel lien for instance '%s' (Reason: %s).	The account manager failed to release the lien.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x110-0a8b2	ADMIN	system.moab	WARN	MWM_RESERVATION_RESERVE_ACCOUNT	Unable to reserve funds for reservation (Reason: %s).	The account manager failed to reserve funds on the account for the reservation.
0x110-0a8b3	ADMIN	system.moab	WARN	MWM_JOB_TASK_DISTRIBUTION	The system cannot distribute the tasks allocated for a job.	Check the tasks specified in the job.
0x110-0a8b4	ADMIN	system.moab	WARN	MWM_JOB_DEFAULT_CLASS	Job cannot run with default class '%s'.	Check the limits set on the class.
0x110-0a8b5	ADMIN	system.moab	WARN	MWM_RM_START_JOB	Cannot start job through a resource manager.	The resource manager may not be set to run the job.
0x110-0a8b6	ADMIN	system.moab	WARN	MWM_JOB_DEPENDENCY_UPDATE	Cannot find job '%s' to update dependency '%s' for job '%s'.	The dependency job for the specified job is missing.
0x110-0a8b7	ADMIN	system.moab	WARN	MWM_EXPIRED_CHECKPOINT	The checkpoint has expired.	Items within the checkpoint may no longer be valid.
0x110-0a8b8	ADMIN	system.moab	WARN	MWM_BAD_CHECKPOINT_LINE	The system encountered an incorrectly formed checkpoint line for key '%s'.	All lines must end with a NEWLINE character.
0x110-0a8b9	ADMIN	system.moab	WARN	MWM_CONVERT_XML_FROM_STRING	XML data cannot be obtained from an XML string ('%s').	There was an error converting from a string that should contain XML into internal XML data structures.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a8ba	ADMI-N	sys-tem.-moab	WAR-N	MWM_CONVERT_XML_TO_STRING	An XML string can-not be constructed from XML data.	There was an error converting from internal XML data structures into an XML string rep-resentation.
0x110-0a8bb	ADMI-N	sys-tem.-moab	WAR-N	MWM_SOCKET_OPERATION	Cannot %s message on sd %s within %s second timeout.	There is a com-munication error with sockets.
0x110-0a8bc	ADMI-N	sys-tem.-moab	WAR-N	MWM_COMPLETED_JOB_RECORD	Could not create job record for completed job %s - %s.	The system may be low on memory.
0x110-0a8bd	ADMI-N	sys-tem.-moab	WAR-N	MWM_CREATE_TEMPLATE_JOB_DEPENDENCY	Could not create tem-plate job depend-ency %s - %s.	The system may be low on memory.
0x110-0a8be	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_FIND_SMP_NODE_BY_FEATURE	Could not find SMP node by feature '%s'.	The feature did not match any of the SMP nodes.
0x110-0a8bf	ADMI-N	sys-tem.-moab	WAR-N	MWM_CHECKPOINT_PROCESS_COMPLETED_JOB	Could not process completed job from checkpoint.	Examine the check-point entry for the job.
0x110-0a8c0	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_FIND_SMP_NODE_IN_PARTITION	Could not find SMP node in partition '%s'.	The feature did not match any of the SMP nodes in the par-tition.
0x110-0a8c1	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_REPORTED_BY_RM_NOT_OWNER	Job '%s' is being reported by RM '%s' but is owned by RM '%s'.	The resource man-ager reporting does not own the job.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x110-0a8c2	ADMIN	system.moab	WARN	MWM_PEER_RM_UNKNOWN_LANGUAGE	Peer RM '%s' reported unknown language: '%s'.	The language does not match a known format.
0x110-0a8c3	ADMIN	system.moab	WARN	MWM_PEER_RM_UNKNOWN_SUBLANGUAGE	Peer RM '%s' reported unknown sublanguage: '%s'.	The language does not match a known format.
0x110-0a8c4	ADMIN	system.moab	WARN	MWM_REQ_ATTR_TO_STRING	Required attribute %s '%s' could not be converted to a string.	There is no string conversion routine for that attribute type.
0x110-0a8c5	ADMIN	system.moab	WARN	MWM_JOB_ARCH_VALUE	Job '%s' does not have a valid arch (architecture) value '%s'.	Check the specified value for the architecture.
0x110-0a8c6	ADMIN	system.moab	WARN	MWM_INVALID_HOST_REQ	Job '%s' does not have a valid host requirement '%s'.	Check the specified value for the requirement.
0x110-0a8c7	ADMIN	system.moab	WARN	MWM_JOB_OPSYS_VALUE	Job '%s' does not have a valid operating system value '%s'.	Check the specified value for the operating system.
0x110-0a8c8	ADMIN	system.moab	WARN	MWM_UNSUPPORTED_REQ	Resouce requirement '%s' not supported.	The requirement specified is unsupported.
0x110-0a8c9	ADMIN	system.moab	WARN	MWM_NO_TASKS	Job loaded in active state with no tasks allocated.	Jobs must have at least one task.
0x110-0a8ca	ADMIN	system.moab	WARN	MWM_SINGLE_ITERATION_JOB_COMPLETION	Scheduler cannot handle job completion in a single iteration.	The job must not start and complete while the scheduler is sleeping.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a8cb	ADMI-N	sys-tem.-moab	WAR-N	MWM_MAX_TASKS_EXCEEDED	The number of tasks associated with a job has exceeded the maximum (%s).	The number of tasks must be reduced or the scheduler must be configured to accept more tasks.
0x110-0a8cc	ADMI-N	sys-tem.-moab	WAR-N	MWM_NODE_OUT_OF_RANGE	Job '%s' node index (%s) at task list index (%s) is out of range.	This is an internal limit.
0x110-0a8cd	ADMI-N	sys-tem.-moab	WAR-N	MWM_NODE_NULL	Job '%s' node index (%s) at task list index (%s) is NULL.	This is an internal limit.
0x110-0a8ce	ADMI-N	sys-tem.-moab	WAR-N	MWM_NODESET_CONSTRAINTS	Nodeset constraints prevent use of task for job '%s':%s at %s.	The specified node-set cannot run the task.
0x110-0a8cf	ADMI-N	sys-tem.-moab	WAR-N	MWM_DEFAULT_WALLTIME	Job assigned default walltime limit (%s).	Unlimited or no wall-time limit specified.
0x110-0a8d0	ADMI-N	sys-tem.-moab	WAR-N	MWM_PARTITION_ACCESS	Job cannot access requested partitions (%s).	The partition access list disallows the job.
0x110-0a8d1	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNABLE_TO_ALLOCATE_TASKS_FOR_JOB	Cannot allocate tasks for job at %s.	The system may be low on memory.
0x110-0a8d2	ADMI-N	sys-tem.-moab	WAR-N	MWM_IGNORING_PARTIAL_RANGE	Ignoring partial time range since full range previously loc-ated.	The system will use the full range instead.
0x110-0a8d3	ADMI-N	sys-tem.-moab	WAR-N	MWM_DESTINATION_RM	Cannot locate a valid destination resource manager for job.	The submitted job could not be sent to a resource manager.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x1100a8d4	ADMIN	system.moab	WARN	MWM_JOB_CREDENTIALS	Cannot authenticate the submitted job (Reason: %s).	The user for the job is not a member of a group or account with access.
0x1100a8d5	ADMIN	system.moab	WARN	MWM_SMPNODE_BY_FEATURE	Could not find SMPNode by feature %s (%s).	None of the nodes has the feature specified.
0x1100a8d6	ADMIN	system.moab	WARN	MWM_SMPNODE_BY_PARTITION	Could not find SMPNode in partition %s.	The SMPNode specified is not in the given partition.
0x1100a8d7	ADMIN	system.moab	WARN	MWM_RSV_ATTR_TO_STRING	Reservation '%s' attribute '%s' could not be converted to a string.	There is no string conversion routine for that attribute type.
0x1100a8d8	ADMIN	system.moab	WARN	MWM_AM_STATUS	Account manager sent failure message - %s.	Check status message.
0x1100a8d9	ADMIN	system.moab	WARN	MWM_AM_FAILURE	Native accounting manager call '%s' failed using input XML '%s'.	Check XML syntax.
0x1100a8da	ADMIN	system.moab	WARN	MWM_AM_INSUFFICIENT_FUNDS	Account manager - Insufficient funds '%s'.	Validate that the user has access to account.
0x1100a8db	ADMIN	system.moab	WARN	MWM_MIGRATE_JOB	Unable to migrate job '%s' to RM '%s' (%s).	Check the error message.
0x1100a8dc	ADMIN	system.moab	WARN	MWM_RESERVE_PRIORITY_JOB	Unable to reserve priority job.	Check the error message.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x110-0a8dd	ADMIN	system.moab	WARN	MWM_SYNC_JOB	Job '%s' not synchronized to start with job '%s'.	The two jobs must start at the same time.
0x110-0a8de	ADMIN	system.moab	WARN	MWM_SYNC_JOB_REQUEUE	Job '%s' could not start. Requeuing any synchronized jobs.	The other jobs should be back on the queue.
0x110-0a8df	ADMIN	system.moab	WARN	MWM_UNHANDLED_PLUGIN_EXCEPTION	A node allocation plugin '%s' encountered an unhandled exception '%s'.	Consult the documention for the plugin.
0x110-0a8e0	ADMIN	system.moab	WARN	MWM_PLUGIN_LOADED_FAILURE	Error loading node allocation plugin '%s' for partition '%s' %s.	A NodeAllocation plugin was not loaded because of an error. Default node allocation will be used.
0x110-0a8e2	ADMIN	system.moab	WARN	MWM_UNKNOWN_JOB_DEPENDENCY	Unknown job dependency '%s' on job.	The job is trying to use a dependency that is unknown.
0x110-0a8e3	ADMIN	system.moab	WARN	MWM_UNSUPPORTED_JOB_DEPENDENCY	Unknown job dependency type '%s' on job.	The job is trying to use a dependency type that is unsupported.
0x110-0a8e4	ADMIN	system.moab	WARN	MWM_MISSING_JOB_DEPENDENCY	Cannot find dependency job. MissingDependencyAction is '%s'.	Check for the existence of the job dependency.
0x110-0a8e5	ADMIN	system.moab	WARN	MWM_PARTITION_REP_NODE	Corrupt partition representative node.	Check the representative node for the partition.
0x110-0a8e6	ADMIN	system.moab	WARN	MWM_PARTITION_ATTRIBUTE	Partition attribute '%s' is not configurable.	Consult the documentation to see which attributes can be configured.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a8e7	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNABLE_TO_LOCATE_NODE	Unable to locate spe-cified nodes for job.	Could not find a node in the job's node list.
0x110-0a8e8	ADMI-N	sys-tem.-moab	WAR-N	MWM_NODES_MISSING_FROM_FEASIBLE_LIST	Specified node(s) not found in feasible hostlist.	Could not find a node.
0x110-0a8e9	ADMI-N	sys-tem.-moab	WAR-N	MWM_HOSTLIST_HAS_TOO_FEW_TASKS	A hostlist has too few tasks available for job '%s':'%s' (%s < %s).	More nodes are needed to satisfy the task requirements.
0x110-0a8ea	ADMI-N	sys-tem.-moab	WAR-N	MWM_TASKS_REMAINING	A hostlist was unable to handle all tasks (%s remain).	More nodes are needed to satisfy the task requirements.
0x110-0a8eb	ADMI-N	sys-tem.-moab	WAR-N	MWM_NODE_DOWN	Unable to detect node '%s' for '%s' seconds. Marking it down or removing it.	Make sure the node is up and running.
0x110-0a8ec	ADMI-N	sys-tem.-moab	WAR-N	MWM_NODE_RESET_EMPTY	Unable to reset node. Node list empty.	Must specify a valid node to reset.
0x110-0a8ed	ADMI-N	sys-tem.-moab	WAR-N	MWM_NODE_RESET_URL	Unable to reset node. NODEPOWERURL not specified.	Must specify a valid URL for the node to reset.
0x110-0a8ee	ADMI-N	sys-tem.-moab	WAR-N	MWM_AM_DOWN	The account man-ager is not currently running.	Check the status of the account manager.
0x110-0a8ef	ADMI-N	sys-tem.-moab	WAR-N	MWM_KEYBOARD_ACTIVITY_PREEMPT_JOB	Keyboard activity on node prevented job preemption.	Jobs can be pree-mpted only if the key-board is idle.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a8f0	ADMI-N	sys-tem.-moab	WAR-N	MWM_KEYBOARD_ACTIVITY_SET_NODE_STATE	Keyboard activity on node prevented set-ting the node state to '%s'.	Node states can be changed only if the keyboard is idle.
0x110-0a8f1	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_MISMATCHED_TIMES	Fixing job '%s' with invalid '%s' times (%s - %s).	Check the times for the specified job.
0x110-0a8f2	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_OPSYS	Cannot add oper-ating system '%s' to job.	Check the type of operating system spe-cified.
0x110-0a8f3	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_ARCH	Cannot add archi-tecture '%s' to job.	Check the type of architecture spe-cified.
0x110-0a8f4	ADMI-N	sys-tem.-moab	WAR-N	MWM_LOCATE_AM	Cannot locate the account manager '%s'.	Check the account manager command option syntax.
0x110-0a8f5	ADMI-N	sys-tem.-moab	WAR-N	MWM_LOCATE_RM	Cannot locate the resource manager '%s'.	Check the resource manager command option syntax.
0x110-0a8f6	ADMI-N	sys-tem.-moab	WAR-N	MWM_LOCATE_RMID	Cannot locate the resource manager ID '%s'.	Check the ID com-mand option syntax.
0x110-0a8f7	ADMI-N	sys-tem.-moab	WAR-N	MWM_LOCATE_PARTITION	Cannot locate the partition '%s'.	Check the partition command option syn-tax.
0x110-0a8f8	ADMI-N	sys-tem.-moab	WAR-N	MWM_RM_QUEUE_MODIFY	Command to modify RM queue failed on resource manager %s - '%s'.	Queue may be con-figured to reject modify requests.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a8f9	ADMI-N	sys-tem.-moab	WAR-N	MWM_RM_QUEUE_CREATE	Command to create RM queue failed on resource manager %s - '%s'.	System may be con-figured to reject queue creation requests.
0x110-0a8fa	ADMI-N	sys-tem.-moab	WAR-N	MWM_RM_QUEUE_CREATE_MISSING_ARGS	Command to create RM queue failed - arguments missing.	The user must sup-ply the needed argu-ments to the command.
0x110-0a8fb	ADMI-N	sys-tem.-moab	WAR-N	MWM_STATIC_RM_DESTRUCTION	An attempt was made to destroy a static resource man-ager.	Static resource man-agers cannot be des-troyed.
0x110-0a8fc	ADMI-N	sys-tem.-moab	WAR-N	MWM_ADD_SYSTEM_USER	Unable to create a new user '%s' in the system.	The system may be low on memory.
0x110-0a8fd	ADMI-N	sys-tem.-moab	WAR-N	MWM_ADD_PARTITION	The system was unable to create par-tition '%s'.	The system may be low on memory.
0x110-0a8fe	ADMI-N	sys-tem.-moab	WAR-N	MWM_CORE_LIMIT	System core limit set to %s (complete core files might not be generated).	Expand the system core limit to ensure the complete core dump can be saved.
0x110-0a8ff	ADMI-N	sys-tem.-moab	WAR-N	MWM_KEY_FILE_PERMISSIONS	The .moab.key file exists, but the file permissions prevent access (%s).	Check the ownership permissions on the file.
0x110-0a900	ADMI-N	sys-tem.-moab	WAR-N	MWM_STATS_PERIOD_TYPE	The system could not process stats for period type %s.	'Day' is the only period type currently supported.
0x110-0a901	ADMI-N	sys-tem.-moab	WAR-N	MWM_STATS_BUFFER_SIZE	The system could not process stats for period type %s (buf-fer too small).	The buffer allocated was too small to hold the data.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x110-0a902	ADMIN	system.moab	WARN	MWM_STATS_FILE	The system could not create the stats file '%s'.	Check the path and user permissions on the directory.
0x110-0a903	ADMIN	system.moab	WARN	MWM_JOB_NO_ACCOUNT	No account specified for job '%s'.	Check the job for an account specification.
0x110-0a904	ADMIN	system.moab	WARN	MWM_SET_JOBATTR_FAIL	Cannot set attribute '%s' to value '%s' on jobmatch '%s'.	Failed to set the given attribute to the given value on the given job.
0x110-0a905	ADMIN	system.moab	WARN	MWM_JOBATTR_NOT_SUPPORTED	JobAttr not supported. '%s'.	The given attribute is not a supported job attribute.
0x110-0a906	ADMIN	system.moab	WARN	MWM_INVALID_TRIGGER_DEFINITION	Invalid trigger definition: %s.	The given trigger is invalid. Check that the given trigger has been defined.
0x110-0a907	ADMIN	system.moab	WARN	MWM_ATTRIBUTE_NOT_HANDLED	System attribute '%s' not handled.	Check that the given attribute was spelled correctly.
0x110-0a908	ADMIN	system.moab	WARN	MWM_QOS_IN_PARAM_NOT_FOUND	Cannot locate QOS '%s' for parameter %s.	Make sure that the given QOS exists.
0x110-0a909	ADMIN	system.moab	WARN	MWM_INVALID_PROFILEDURATION_VAL	Invalid PROFILEDURATION specified, modified internally to %s (see documentation).	The entered PROFILEDURATION value is invalid. Moab uses the given value instead.
0x110-0a90a	ADMIN	system.moab	WARN	MWM_NO_DATA_STAGING_PATH	No path in data staging specification '%s' (bad format).	Verify the data staging path is specified.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x1100a90b	ADMIN	system.moab	WARN	MWM_UNSUPPORTED_RM_DATA_STAGING	Cannot stage-out stdout/stderr (unsupported RM type '%s').	Failed to stage-out stdout/stderr because the given resource manager does not support such feature.
0x1100a90c	ADMIN	system.moab	WARN	MWM_RM_DATA_ON_NON_EXISTING_JOB	Storage RM '%s' reporting data operation for non-existant job '%s'.	The given resource manager is reporting data operation on the non-existing job.
0x1100a90d	ADMIN	system.moab	WARN	MWM_INVALID_RM_DATA_STAGE	Data stage for RM '%s' not possible as it has no nodelist.	Check CLUSTERQUERYURL to ensure it at least has a nodelist.
0x1100a90e	ADMIN	system.moab	WARN	MWM_DATA_STAGE_IN_FAIL	Data stage in failed for job '%s' file '%s' (%s).	Failed to complete the data staging in operation for the job on the given file due to error(s).
0x1100a90f	ADMIN	system.moab	WARN	MWM_UNABLE_TO_REMOVE_DATA_STAGE	Cannot remove data staging block for job '%s'.	Failed to remove the data staging block for the given job.
0x1100a910	ADMIN	system.moab	WARN	MWM_DATA_STAGE_OUT_FAIL	Data stage out failed for job '%s' file '%s' (%s).	Failed to complete the data staging out operation for the job on the given file due to error(s).
0x1100a911	ADMIN	system.moab	WARN	MWM_JOB_INVALID_OPSYS	Job '%s' cannot request OS '%s').	The requested operating system is not available for the job.
0x1100a912	ADMIN	system.moab	WARN	MWM_NODE_BUFFER_OVERFLOW	Node buffer is full.	Try increasing the node buffer size.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a913	ADMI-N	sys-tem.-moab	WAR-N	MWM_REMOVE_NODE_WITH_RESERVATION	Unable to remove node '%s' because of reservation ref-erences.	Remove the reser-vations from the node.
0x110-0a914	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_PURGE_RM_INACTIVE	Unable to purge job '%s' because the resouce manager '%s' is inactive.	Check the status of the resource man-ager.
0x110-0a915	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_PURGE_RM_NO_RESOURCES	Unable to purge job '%s' because the resouce manager '%s' is reporting no resources.	Check the status of the resource man-ager.
0x110-0a916	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_NOT_DETECTED	Job '%s' in state '%s' no longer detected (Last Detected %s > PurgeTime %s).	The job may have been purged in the meantime.
0x110-0a917	ADMI-N	sys-tem.-moab	WAR-N	MWM_BACKFILL_DEPTH_REACHED	The backfill depth (BFDEPTH) has been reached so no more jobs will be back-filled this iteration.	Wait for the next iter-ation or increase the depth.
0x110-0a918	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_NO_QUEUETIME	No QueueTime has been specified for job.	Configure the job with a queue time.
0x110-0a91c	ADMI-N	sys-tem.-moab	WAR-N	MWM_CHECKPOINT_CREATE_RSV_FROM_XML	Unable to create a reservation from checkpoint XML.	The system may be low on memory.
0x110-0a91d	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_CACHE_REMOVAL	Failed to remove job %s (ID = %s) from the cache.	The job was missing from the system hash table.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a91e	ADMI-N	sys-tem.-moab	WAR-N	MWM_INVALID_CRED_VALUE	Invalid credential value '%s'.	Check the syntax in the configuration file.
0x110-0a91f	ADMI-N	sys-tem.-moab	WAR-N	MWM_INVALID_CRED_ATTR	Invalid credential attribute '%s'.	Check the syntax in the configuration file.
0x110-0a920	ADMI-N	sys-tem.-moab	WAR-N	MWM_MAX_JOBS_EXCEEDED	The maximum num-ber of jobs has been exceeded.	Increase the value of the MAXJOB setting.
0x110-0a921	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_FAILED_PROCESSING_PBS	A job failed while processing PBS resources.	May not have been able to locate host or vnode.
0x110-0a922	ADMI-N	sys-tem.-moab	WAR-N	MWM_FIND_PEER	Cannot find client peer for job %s (Name: %s).	The resource man-ager cannot be loc-ated.
0x110-0a923	ADMI-N	sys-tem.-moab	WAR-N	MWM_AM_INSUFFICIENT_BALANCE	Insufficient balance in primary account '%s' to run job '%s' (attempting fallback credentials).	Validate that the user has access to account.
0x110-0a924	ADMI-N	sys-tem.-moab	WAR-N	MWM_AM_JOB_SUBMIT_VALIDATION	Job submission val-idation failed for job '%s' -- taking action '%s'.	Validate that the job has access.
0x110-0a925	ADMI-N	sys-tem.-moab	WAR-N	MWM_TOO_MANY_NODE_SETS	The maximum num-ber of node sets has been exceeded.	This is a configurable setting.
0x110-0a926	ADMI-N	sys-tem.-moab	WAR-N	MWM_CLASS_SET_LIST_INVALID	The specified class set list is invalid '%s'.	Check the doc-umentation for valid classes.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a928	ADMI-N	sys-tem.-moab	WAR-N	MWM_RM_START	Cannot start resource manager '%s' (Reason: %s).	The resource man-ager may not be avail-able.
0x110-0a929	ADMI-N	sys-tem.-moab	WAR-N	MWM_STDOUT_FAIL	Request succeeded with no stdout. stderr= '%s'.	The standard out may not have been specified.
0x110-0a92a	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNSUPPORTED_REQ_ATTR	Unsupported req attribute '%s'.	The attribute is not one that can be set.
0x110-0a92b	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNSUPPORTED_GENERAL_ATTR	Unsupported gen-eral attribute '%s'.	The attribute is not one that can be set.
0x110-0a92e	ADMI-N	sys-tem.-moab	WAR-N	MWM_CREATE_ACCOUNT	Unable to create account '%s' on the account manager.	Verify that the account manager is running.
0x110-0a92f	ADMI-N	sys-tem.-moab	WAR-N	MWM_QUERY_ACCOUNT	Unable to query account '%s' on the account manager.	Verify that the account name is cor-rect.
0x110-0a930	ADMI-N	sys-tem.-moab	WAR-N	MWM_ACCOUNT_ADD_USER	Unable to add user '%s' to account '%s' on the account man-ager.	Verify that the account name is cor-rect.
0x110-0a931	ADMI-N	sys-tem.-moab	WAR-N	MWM_ACCOUNT_DEPOSIT	Unable to deposit '%s' credits to account '%s' on the account manager.	Verify that the account name is cor-rect.
0x110-0a932	ADMI-N	sys-tem.-moab	WAR-N	MWM_ALLOCATE_REQ	Unable to allocate requirement '%s' using NAllocPolicy '%s' (%s).	The system may be low on memory.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x110-0a933	ADMIN	system.moab	WARN	MWM_VALID_STAT_DATA	Unable to generate valid statistic data for external query.	The system may be low on memory.
0x110-0a934	ADMIN	system.moab	WARN	MWM_JOB_QOS_REQUEST	Job '%s' cannot request QOS '%s').	The requested QOS is not available for the job.
0x110-0a938	ADMIN	system.moab	WARN	MWM_GRES_OVERFLOW	GRES overflow.	Unable to add another GRES.
0x110-0a939	ADMIN	system.moab	WARN	MWM_RM_NO_RESOURCES	The resouce manager '%s' is reporting no resources.	Check the nodes on the resource manager.
0x110-0a93d	ADMIN	system.moab	WARN	MWM_EMPTY_FILE	File '%s' is empty.	Check the file specified.
0x110-0a93f	ADMIN	system.moab	WARN	MWM_PREEMPT_NONACTIVE_JOB	Cannot preempt non-active job '%s' (state: '%s' estate: '%s').	The job must currently be active to preempt it.
0x110-0a940	ADMIN	system.moab	WARN	MWM_REQUEUE_NONSTARTABLE_JOB	Cannot requeue non-startable job '%s' (canceling instead).	The job could not be requeued.
0x110-0a941	ADMIN	system.moab	WARN	MWM_NODE_RACK_VALUE	Invalid rack value '%s' specified for node %s (must be digit).	Check the value of the rack parameter.
0x110-0a942	ADMIN	system.moab	WARN	MWM_ENCODE_JOB_MESSAGE	Cannot encode job message.	Check the value of the rack parameter.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a943	ADMI-N	sys-tem.-moab	WAR-N	MWM_INVALID_PROFILECOUNT_VAL	Invalid PROFILECOUNT spe-cified, modified internally to %s (see documentation).	The PROFILECOUNT value is invalid. Moab uses the default value instead.
0x110-0a944	ADMI-N	sys-tem.-moab	WAR-N	MWM_FAIRSHARE_FILE	Cannot load fair-share file '%s' for slot %s.	Check for the exist-ence of the fairshare file in the file system.
0x110-0a945	ADMI-N	sys-tem.-moab	WAR-N	MWM_REQ_ATTR_ALREADY_SET	Requirement attrib-ute %s '%s' is already set.	Check the attribute setting in the con-figuration file.
0x110-0a946	ADMI-N	sys-tem.-moab	WAR-N	MWM_ZERO_START_TIME	StartTime set to zero for reservation on job '%s'.	Check the start time for the specified job.
0x110-0a947	ADMI-N	sys-tem.-moab	WAR-N	MWM_EXISTING_RESERVATION	Reservation created for reserved job '%s' (existing reservation '%s' deleted).	Only one reservation can exist at a time for the reserved job.
0x110-0a948	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_PARSE_REQ_LINE	Cannot parse requirement line for job '%s'.	The syntax of the requirement line is incorrect.
0x110-0a949	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNKNOWN_RESOURCE_TYPE	Unknown resource type '%s' for job '%s'.	Check the doc-umentation for valid resource types.
0x110-0a94a	ADMI-N	sys-tem.-moab	WAR-N	MWM_UNKNOWN_TRANSACTION_ATTR	Unknown trans-action attribute '%s'.	Check the doc-umentation for valid transaction attrib-utes.
0x110-0a94c	ADMI-N	sys-tem.-moab	WAR-N	MWM_ID_MANAGER_DOWN	The identity man-ager is down.	Check the status of the identity manager.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a94d	ADMI-N	sys-tem.-moab	WAR-N	MWM_PROCESS_ID_LINE	Unable to process the ID line '%s'.	Check the syntax of the attribute/value pairs.
0x110-0a94e	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_ID_MISSING	Unable to locate the job ID for a job sub-mitted to the resource manager.	Check the job being submitted`.
0x110-0a950	ADMI-N	sys-tem.-moab	WAR-N	MWM_EMPTY_NODELIST	The nodelist is empty for reser-vation '%s'.	Reservations should include a node list.
0x110-0a951	ADMI-N	sys-tem.-moab	WAR-N	MWM_FAIRSHARE_PAL	Fairshare does not allow specified PAL (%s).	The fairshare algorithm is reverting to the original PAL.
0x110-0a952	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_START_TIME	Cannot find earliest start time for job '%s'.	Resources needed to run the job may never be available.
0x110-0a953	ADMI-N	sys-tem.-moab	WAR-N	MWM_NODE_MODIFY	Cannot modify node '%s' Error(%s).	The node could not be modified.
0x110-0a954	ADMI-N	sys-tem.-moab	WAR-N	MWM_TRIGGER_RSV_CREATE	Unable to create a trigger reservation.	Check the reser-vation time, nodes, and account.
0x110-0a955	ADMI-N	sys-tem.-moab	WAR-N	MWM_MISSING_TRIGGER	Trigger '%s' with PID '%s' does not exist--completing!	The process may have already com-pleted.
0x110-0a956	ADMI-N	sys-tem.-moab	WAR-N	MWM_EMPTY_HOSTLIST	The hostlist is empty for reservation.	Reservations should include a hostlist.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a957	ADMI-N	sys-tem.-moab	WAR-N	MWM_RSV_POLICY_VIOLATION	Unable to create requested reser-vation due to a policy violation (%s).	Reservations must conform to existing policies.
0x110-0a958	ADMI-N	sys-tem.-moab	WAR-N	MWM_RSV_CREATE_FAILURE	Unable to create requested reser-vation at time %s (%s).	Resources are unavailable at requested time.
0x110-0a959	ADMI-N	sys-tem.-moab	WAR-N	MWM_RSV_OWNER	Cannot process owner '%s' for stand-ing reservation '%s' (%s).	Consult the error message.
0x110-0a95a	ADMI-N	sys-tem.-moab	WAR-N	MWM_RSV_PARTIAL	Partial reservation %s reserved %s of %s procs in partition '%s' to start in %s at (%s) %s.	Entire reservation could not be filled.
0x110-0a95c	ADMI-N	sys-tem.-moab	WAR-N	MWM_RSV_NEGATIVE_JOBCOUNT	Reservation %s job-count is %s, should not decrement less than 0.	JobCount cannot be negative.
0x110-0a95d	ADMI-N	sys-tem.-moab	WAR-N	MWM_EMPTY_REQ_NODELIST	Req node list empty for job %s:%s in state %s (job nodel-ist copied to req nodelist).	Job should include a req node list.
0x110-0a95e	ADMI-N	sys-tem.-moab	WAR-N	MWM_TASK_ALLOCATION_INFO	Cannot locate task allocation info for job %s:%s in state %s.	Job should include a task list.
0x110-0a95f	ADMI-N	sys-tem.-moab	WAR-N	MWM_INVALID_THREADPOOL_SIZE	Invalid ThreadPoolS-ize '%s' (must be a non-negative integer no larger than %s).	Check the size for a valid value.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x1100a960	ADMIN	system.moab	WARN	MWM_INVALID_QUEUE_TIME	Job '%s' has invalid system queue time (SQ: %s > ST: %s).	Check the job queue time value.
0x1100a961	ADMIN	system.moab	WARN	MWM_NO_WCLIMIT	Job '%s' has no WCLimit specified.	Check the job for the correct value.
0x1100a962	ADMIN	system.moab	WARN	MWM_AM_INVALID_PROTOCOL	Invalid protocol '%s' specified for account manager '%s'.	Communication with the account manager must be over a supported protocol.
0x1100a963	ADMIN	system.moab	WARN	MWM_NO_POWER_INTERFACE	No external power interface - cannot set power state '%s' on node '%s%s%s'.	Cannot set the power state on the node without a power interface.
0x1100a964	ADMIN	system.moab	WARN	MWM_JOB_STARTED_ON_ANOTHER_RM	Job '%s' started externally: (rc: %s; errmsg: '%s'; Tasklist: '%s').	Two or more resource managers are running side-by-side and the job is already running on one of them.
0x1100a965	ADMIN	system.moab	WARN	MWM_COMMAND_FAILED_CHILD_PROCESS	Job submit request failed with child process status code='%s', stderr='%s', stdout='%s', EMsg='%s'.	Review the status code and error message for further information.
0x1100a967	ADMIN	system.moab	WARN	MWM_AM_REGISTER_JOB	Unable to register job creation with account manager for job '%s', reason: '%s'.	Check the status of the account manager.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0a968	ADMI-N	sys-tem.-moab	WAR-N	MWM_AM_DEPRECATED_PARAMETER	Use of the JOBFAILUREACTION parameter is deprec-ated. Use STARTFAILUREACTI-ON instead.	Check the doc-umentation for the new parameter syn-tax.
0x110-0a969	ADMI-N	sys-tem.-moab	WAR-N	MWM_AM_INVALID_ACTION	Invalid action '%s' specified in '%s' for account manager '%s'.	Check the doc-umentation for valid actions for the account manager.
0x110-0a9bf	ADMI-N	sys-tem.-moab	WAR-N	MWM_ERASING_JOB	Erasing job '%s' by address.	The specified job could not be found by name. The entire job table was searched to find the matching job.
0x110-0a9e1	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_INVALID_TASK_LAYOUT	Job '%s' has invalid task layout (TPN:%s * N:%s != T:%s).	The task layout does not compute.
0x110-0a9e2	ADMI-N	sys-tem.-moab	WAR-N	MWM_JOB_ACCESS_QOS	Job '%s' does not have access to QOS '%s' (QAL: %s).	The QoS is not access-ible from the job.
0x110-0a9f8	ADMI-N	sys-tem.-moab	WAR-N	MWM_DUPLICATE_SYSTEMJID	Duplicate SystemJID '%s' [JState: %s] found from RM '%s'.	The SystemJID must be unique.
0x110-0a9fc	ADMI-N	sys-tem.-moab	WAR-N	MWM_CANNOT_PING_RM	Cannot ping RM '%s' because a file was not specified.	A file path to a valid file is needed.
0x110-0aa07	ADMI-N	sys-tem.-moab	WAR-N	MWM_VM_CONTAINER_NODE	Cannot find or add container node '%s' for VM '%s'.	The node could not be found.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x1100e710	INTERNAL	system.moab	WARN	MWM_INVALID_ARG_VALUE	Invalid arguments passed to this function.	One or more arguments passed to this function were not valid. This is an internal error logged for informational purposes.
0x1100e72b	INTERNAL	system.moab	WARN	MWM_SEND_SENT_NO_DATA	No data was sent to the socket when it should have been.	The send() system call reported no data was sent when data should have been sent.
0x1100e72e	INTERNAL	system.moab	WARN	MWM_SOCKET_BLOCKED_UNEXPECTEDLY	Read operations on the socket were blocked when it should have been available.	A socket operation reported that the operation was blocked. Previous information indicated that this operation should have been available.
0x1100e73f	INTERNAL	system.moab	WARN	MWM_MUTEX_LOCK	Cannot lock mutex semaphore using pthread_mutex_lock().	This is an operating system call problem.
0x1100e740	INTERNAL	system.moab	WARN	MWM_MUTEX_UNLOCK	Cannot unlock mutex semaphore using pthread_mutex_unlock().	This is an operating system call problem.
0x1100e771	INTERNAL	system.moab	WARN	MWM_THREAD_DB_INIT	Thread %s attempting to re-initialize database info struct.	Internal error condition.
0x1100e84b	INTERNAL	system.moab	WARN	MWM_CORRUPT_COMMAND_RECEIVED	Corrupt command '%s' received.	The communication packet received from a Moab client command is malformed.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0e919	INTER-NAL	sys-tem.-moab	WAR-N	MWM_CHECKPOINT_NO_XML	The checkpoint data does not contain XML.	Internal error.
0x110-0e91a	INTER-NAL	sys-tem.-moab	WAR-N	MWM_CHECKPOINT_INVALID_XML	The checkpoint data does not contain valid XML (%s).	Internal error.
0x110-0e91b	INTER-NAL	sys-tem.-moab	WAR-N	MWM_CHECKPOINT_UPDATE_RSV_FROM_XML	Unable to update a reservation from checkpoint XML.	Internal error.
0x110-0e927	INTER-NAL	sys-tem.-moab	WAR-N	MWM_JOB_ATTR_TO_STRING	Job attribute '%s' not yet translated to string value.	Internal warning.
0x110-0e92c	INTER-NAL	sys-tem.-moab	WAR-N	MWM_MISSING_STATUS_CODE	The status code was missing from the S3 response.	This is an internal error.
0x110-0e92d	INTER-NAL	sys-tem.-moab	WAR-N	MWM_MISSING_STATUS_VALUE	The status value was missing from the S3 response.	This is an internal error.
0x110-0e93a	INTER-NAL	sys-tem.-moab	WAR-N	MWM_SIMULATION_NO_JOBS	No jobs loaded in simulation.	Internal simulation error.
0x110-0e93b	INTER-NAL	sys-tem.-moab	WAR-N	MWM_SIMULATION_JOB_DETECTED_TRACEBUFFER	Job '%s' previously detected in tracefile (MJobTraceBuffer [%s]/JC: %s; IT: %s).	Internal simulation error.
0x110-0e93c	INTER-NAL	sys-tem.-moab	WAR-N	MWM_SIMULATION_JOB_DETECTED	Job '%s' previously detected in tracefile (Job/JC: %s; IT: %s).	Internal simulation error.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x110-0e93e	INTER-NAL	sys-tem.-moab	WAR-N	MWM_READ_COMMAND_OUTPUT	Cannot read output of command '%s'.	This is an internal communications error.
0x110-0e94b	INTER-NAL	sys-tem.-moab	WAR-N	MWM_THREAD_TIMEOUT	Thread %s killed (%s micro-second time out reached).	This is an internal issue.
0x110-0e94f	INTER-NAL	sys-tem.-moab	WAR-N	MWM_INVALID_XML_RM	Invalid XML data for resource manager '%s'.	Check the XML syn-tax.
0x210-00067	USER	domain.li-fecycle	ERR-OR	MWM_JOB_END_FAILED	Job %s failed. %s	The job finished unsuccessfully.
0x210-000ca	USER	domain.li-fecycle	ERR-OR	MWM_NODE_EVAC_VMS_ERROR	Error evacuating VMs off node %s. %s	There was an error while attempting to evacuate the VMs off the node.
0x210-02882	USER	sys-tem.-moab	ERR-OR	MWM_NODE_MODIFY_FAILURE	Cannot modify node state of '%s' Error (%s).	The node state could not be modified.
0x210-02a1a	USER	sys-tem.-moab	ERR-OR	MWM_DEPRECATED_RM_FEATURE	RM flag SUBMITJOBSASROOT not supported with this version, %s. Must be >= 2.4.8.	The resource man-ager version should be updated to get support for this fea-ture.
0x210-08003	ADMI-N	sys-tem.-moab	ERR-OR	MWM_TESTING_ERROR	Testing with argu-ment1: %s. and argu-ment2: %s.	Internal error for testing diagnostics.
0x210-08263	ADMI-N	domain.li-fecycle	ERR-OR	MWM_VC_SCHEDULE_FAILURE	Failed to schedule virtual container '%s'.	This is an internal error.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a718	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_FORK_ERROR	Cannot fork the pro-cess, errno: %s (%s).	The fork() system call failed. Use the errno and associated message to determ-ine possible causes.
0x210-0a719	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_EXEC_PROGRAM	Cannot exec action '%s', errno: %s (%s).	The exec() system call failed to execute the command. This may be because the command does not exist or the per-missions do not allow it to be run. Use the errno and associated message to determ-ine possible causes.
0x210-0a71b	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_CHOWN_FILE	Failure changing ownwership of file: '%s' to uid:'%s', gid:'%s', errno: %s (%s).	The chown() system call failed. Use the errno and associated message to determ-ine possible causes.
0x210-0a71f	ADMI-N	sys-tem.-moab	ERR-OR	MWM_PIPE_READ_FAILED	Failed to read pipe on command '%s', errno: %s (%s).	The fread() system call failed. Use the errno and associated message to determ-ine possible causes.
0x210-0a720	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_READ_FILE	Cannot read file '%s', errno: %s (%s).	The fread() system call failed. Use the errno and associated message to determ-ine possible causes.
0x210-0a721	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_WRITE_TO_FILE	Failure writing to file, errno: %s (%s).	The write() system call failed. Use the errno and associated message to determ-ine possible causes.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x2100a725	ADMIN	system.moab	ERROR	MWM_CANNOT_GET_HOSTNAME	Cannot get hostname '%s', errno: %s (%s).	The gethostname() system call failed. Use the errno and associated message to determine possible causes.
0x2100a727	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_SOCKET	Failure creating a socket, errno: %s (%s).	The socket() system call failed. Use the errno and associated message to determine possible causes.
0x2100a728	ADMIN	system.moab	ERROR	MWM_CANNOT_CONNECT_TO_HOST	Failure connecting to server '%s' on port %s, errno: %s (%s).	The connect() system call failed. Use the errno and associated message to determine possible causes.
0x2100a730	ADMIN	system.moab	ERROR	MWM_EPOCH_FAIL	Epoch Fail, time: '%s' cannot be converted to an epoch time, errno: %s (%s).	The mktime() system call failed. Use the errno and associated message to determine possible causes.
0x2100a731	ADMIN	system.moab	ERROR	MWM_MEMORY_ALLOCATION_FAILURE_MALLOC	Failure allocating memory (malloc), allocating '%s' bytes, errno: %s (%s).	The malloc() system call failed. Use the errno and associated message to determine possible causes.
0x2100a732	ADMIN	system.moab	ERROR	MWM_MEMORY_ALLOCATION_FAILURE_CALLOC	Failure allocating memory (calloc), allocating '%s' elements of size '%s' bytes, errno: %s (%s).	The calloc() system call failed. Use the errno and associated message to determine possible causes.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x2100a733	ADMIN	system.moab	ERROR	MWM_MEMORY_ALLOCATION_FAILURE_REALLOC	Failure allocating memory (realloc), allocating '%s' bytes, errno: %s (%s).	The realloc() system call failed. Use the errno and associated message to determine possible causes.
0x2100a734	ADMIN	system.moab	ERROR	MWM_CANNOT_DUPLICATE_STRING	Failure duplicating string, errno: %s (%s).	The strdup() system call failed. Use the errno and associated message to determine possible causes.
0x2100a735	ADMIN	system.moab	ERROR	MWM_CANNOT_CHANGE_PROCESS_GROUP	Failure changing process group, errno: %s (%s).	The setpgrp() system call failed. Use the errno and associated message to determine possible causes.
0x2100a736	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_THREAD	Failure creating thread: '%s', errno: %s (%s).	The pthread_create() system call failed. Use the errno and associated message to determine possible causes.
0x2100a737	ADMIN	system.moab	ERROR	MWM_CANNOT_TRUNCATE_FILE	Failure truncating a file '%s', errno: %s (%s).	The truncate() system call failed. Use the errno and associated message to determine possible causes.
0x2100a739	ADMIN	system.moab	ERROR	MWM_PIPE_OPEN_FAILED	Failed to open pipe on command '%s', errno: %s (%s)	The popen() system call failed. Use the errno and associated message to determine possible causes.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a73d	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CHANGE_DIR_FAILURE	OS call to change dir-ectory to '%s' failed errno: %s (%s).	The chdir() system call failed. Use the errno and associated message to determ-ine possible causes.
0x210-0a74a	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_LOCK_MOAB_PID_FILE	Cannot lock the PID file '%s'. Is Moab already running?	Moab tries to ensure that only one instance of itself is running. In the default configuration it will exit if it cannot obtain a lock.
0x210-0a74e	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CONFIG_FILE_NOT_FOUND_ERROR	Cannot locate con-figuration file in any predetermined loc-ation.	Moab cannot find the configuration file. Verify that it is present and installed in a proper location.
0x210-0a757	ADMI-N	sys-tem.-moab	ERR-OR	MWM_MWS_RM_CONFIGURATIO-N	The resource man-ager with Moab Web Services (%s) does not have a base URL, username, and pass-word configured.	Correctly configure the Moab Web Ser-vices resource man-ager.
0x210-0a761	ADMI-N	sys-tem.-moab	ERR-OR	MWM_STRICT_INVALID_CONFIG_LINE	Error processing line #%s: %s - (%s).	Check the line num-ber in the con-figuration file.
0x210-0a767	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NO_MONGOSERVER_SPECIFIED	Failed to initialize connection to Mongo server. (Moab is con-figured to use Mongo, but no MONGOSERVER is specified.)	Cannot connect to the Mongo server since the MONGOSERVER para-meter was unspe-cified. Add MONGOSERVER para-meter to moab.cfg.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x2100a768	ADMIN	system.moab	ERROR	MWM_MONGOSERVER_INITIALIZATION_FAILED	Failed to initialize connection to Mongo server '%s'.	Failed to initialize connection to the configured MONGOSERVER. Check the following: (1) network connection to Mongo server; and (2) check MONGOUSER and MONGOPASSWORD parameters in moab-private.cfg.
0x2100a76a	ADMIN	system.moab	ERROR	MWM_MONGOSERVER_AUTHENTICATION_FAILURE	Failed to authenticate to Mongo server (%s).	Check user credentials.
0x2100a76b	ADMIN	system.moab	ERROR	MWM_MONGOSERVER_WRITE_FAILURE	Unable to write out transition object '%s'.	The BSON information is invalid or missing.
0x2100a76c	ADMIN	system.moab	ERROR	MWM_MONGOSERVER_DOWN	The Mongo server is down.	Check the status of the server.
0x2100a76f	ADMIN	system.moab	ERROR	MWM_DB_CHECKPOINT_OBJECT	Unable to checkpoint object to the database (%s).	Make sure the database is running.
0x2100a773	ADMIN	system.moab	ERROR	MWM_BACKUP_SERVER_CONNECTION_FAILED	The system was unable to connect to the backup server %s (%s:%s).	Make sure the backup server's address is correct.
0x2100a776	ADMIN	system.moab	ERROR	MWM_CONNECTION_REFUSED	Connection to the server was refused (%s).	Primary server refused and no fallback server available.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x210-0a777	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_CONNECT	Cannot send request to %s:%s (%s may not be running).	Unable to connect to the scheduler program.
0x210-0a779	ADMIN	system.moab	ERROR	MWM_CLIENT_MAX_CONNECTIONS_REACHED	Cannot accept connection number %s (transaction number %s) from '%s' (limit reached).	May need to increase the CLIENTMAXCONNECTIONS configuration setting.
0x210-0a77a	ADMIN	system.moab	ERROR	MWM_SERVER_CONNECTION_FAILED	The system was unable to connect to the server %s:%s - %s.	Make sure the server's address is correct and it is running.
0x210-0a77c	ADMIN	system.moab	ERROR	MWM_COMMUNICATION_ERROR	Communication error %s:%s (%s).	General error trying to communicate with the host.
0x210-0a77d	ADMIN	system.moab	ERROR	MWM_CANNOT_PARSE_SERVER_RESPONSE_STATUS	Cannot parse server response (status).	The response sent from the server is malformed.
0x210-0a77e	ADMIN	system.moab	ERROR	MWM_CANNOT_PARSE_SERVER_RESPONSE_DATA	Cannot parse server response (data).	The response sent from the server is malformed.
0x210-0a77f	ADMIN	system.moab	ERROR	MWM_INVALID_FS_TARGET	Invalid type specified for FSTarget.	Fairshare target type is invalid.
0x210-0a780	ADMIN	system.moab	ERROR	MWM_COULD_NOT_ADD_FS_TREE_NODE	Could not add fstree node %s.	Unable to add a node to the fairshare configuration tree.
0x210-0a781	ADMIN	system.moab	ERROR	MWM_CANNOT_ADD_MANAGER_TO_FS_TREE	Could not add manager %s to fstree.	Unable to add a manager to the fairshare tree.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a782	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CRED_MANAGER_OVERFLOW	CredManager over-flow, manager %s not added.	Credential Manager could not add another manager.
0x210-0a783	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CRED_MANAGER_OVERFLOW_CHILD	CredManager over-flow while adding managers to child in fstree.	Fairshare tree con-figuration problem.
0x210-0a786	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_TO_SELECT_TASKS_FOR_JOB	Cannot select tasks for job %s. (%s)	Cannot select a node list that matches the requirements for this job. This may not be serious since multiple passes may occur.
0x210-0a787	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_GET_TASK_ON_RESERVATION	Cannot get tasks on (ERR: %s/no reser-vation/iteration %s).	Cannot select tasks that meet the requirements.
0x210-0a78a	ADMI-N	sys-tem.-moab	ERR-OR	MWM_BEST_VAL_ACHIEVED_BUT_SCHEDULE_EMPTY	BestVal %s achieved but schedule is empty.	Best value has been set, but the schedule is empty.
0x210-0a78b	ADMI-N	sys-tem.-moab	ERR-OR	MWM_JOB_SCHEDULING_FAILURE_NO_RESERVATION	Scheduling failure %s (policy viol-ation/no reser-vation) iteration: %s. (%s)	The job was not scheduled because no reservations are available.
0x210-0a78c	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNSUPPORTED_SERVICE	Service '%s' (%s) not supported.	A request for an unsupported service was sent.
0x210-0a78d	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INVALID_CLASS_HOST_EXPRESSION	Invalid class host expression received (%s) : %s.	Failed to expand the class's host pattern to a list.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x2100a78e	ADMIN	system.moab	ERROR	MWM_TOO_MANY_COALLOCATION_REQUESTS	Too many co-allocation requests (%s > %s).	Too many co-allocation requests were received.
0x2100a78f	ADMIN	system.moab	ERROR	MWM_INVALID_JOBID_COUNTER	Min Job ID '%s' must be less than Max Job ID '%s'.	Invalid job id was encountered.
0x2100a790	ADMIN	system.moab	ERROR	MWM_PARAMETER_NOT_HANDLED	Parameter[%s] '%s' not handled.	The specified parameter was not handled due to an unknown format.
0x2100a791	ADMIN	system.moab	ERROR	MWM_CIRCULAR_JOB_DEPENDENCY	Job cannot be dependent on itself.	The job is trying to use itself as a dependency, which creates a circular dependency and is invalid.
0x2100a792	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_AM	Cannot create AM %s.	Could not create account manager object.
0x2100a793	ADMIN	system.moab	ERROR	MWM_INVALID_FLUSH_INTERVAL	%s for AM %s.	An invalid flush interval has been entered.
0x2100a794	ADMIN	system.moab	ERROR	MWM_FAILED_SERVER_AUTH	Unable to authenticate server.	The server could not be authenticated.
0x2100a795	ADMIN	system.moab	ERROR	MWM_NO_QUOTE	No quote output provided in response.	No quote output provided in response.
0x2100a796	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_PARSE_XML	Unable to parse XML (%s): %s.	Unable to parse XML data.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x2100a797	ADMIN	system.moab	ERROR	MWM_INVALID_QUOTE	Invalid quote amount (%s).	Quote is invalid.
0x2100a798	ADMIN	system.moab	ERROR	MWM_RECURRING_COST	Unable to determine recurring cost.	Unable to determine recurring cost.
0x2100a799	ADMIN	system.moab	ERROR	MWM_AVAILABLE_PORT_NOT_FOUND	Cannot locate an available port for listening.	After trying to bind to a large number of ports, none were found to be available. Check network socket status for saturation.
0x2100a79a	ADMIN	system.moab	ERROR	MWM_CANNOT_RESOLVE_IP_FROM_HOSTNAME	Cannot resolve IP address from hostname '%s', getaddrinfo() rc: %s (%s).	There is a failure matching an IP address to a hostname. Check DNS, /etc/hosts or applicable nameservice.
0x2100a79b	ADMIN	system.moab	ERROR	MWM_UNKNOWN_CHECKPOINT_TYPE	Unexpected checkpoint type, %s.	Unknown checkpoint type while reading from the file.
0x2100a79c	ADMIN	system.moab	ERROR	MWM_CHECKPOINT_FILE_LINE_NOT_HANDLED	Line '%s' not handled in checkPoint file '%s'.	Please contact Adaptive Computing for assistance.
0x2100a79d	ADMIN	system.moab	ERROR	MWM_CANNOT_ADD_DEFAULT_GROUP	Cannot add default group.	Default group cannot be added.
0x2100a79e	ADMIN	system.moab	ERROR	MWM_CANNOT_ADD_GROUP	Cannot add group %s.	Group cannot be added.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a79f	ADMI-N	sys-tem.-moab	ERR-OR	MWM_ACCOUNT_NOT_ACCESSIBLE_BY_JOB	Account '%s' is not accessible by job '%s'.	The job is not author-ized to run under the listed account.
0x210-0a7a0	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_DETERMINE_DEFAULT_ACCOUNT	Unable to determine default account for job '%s', user '%s'.	There is not a default account type for this job.
0x210-0a7a1	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_CREATE_RESERVATION	Cannot create reser-vation for job '%s'.	Failed to create reser-vation for job.
0x210-0a7a2	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INVALID_NODELIST_BAD_TASKCOUNT	Invalid nodelist for job %s:%s (inad-equate taskcount, %s < %s).	Invalid node list due to indequate task count.
0x210-0a7a3	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INVALID_NODELIST_BAD_NODECOUNT	Invalid nodelist for job %s:%s (inad-equate nodecount, %s < %s).	Invalid node list due to indequate node count.
0x210-0a7a4	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INVALID_ALLOCATION_POLICY	Invalid allocation policy (%s).	Invalid allocation policy.
0x210-0a7a5	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NO_MEMORY_FOR_ALLOCPARTITIO-N_VARIABLE	Cannot set ALLOCPARTITION variable for job %s (no memory).	No memory remain-ing to create job vari-able.
0x210-0a7a6	ADMI-N	sys-tem.-moab	ERR-OR	MWM_BASIL_RSVID_NOT_FOUND	Cannot locate BASIL RSVID (job 'ALLOCPARTITION' variable) that was just created.	Cannot locate BASIL reservation id stored in the ALLOCPARTITION variable.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a7a7	ADMI-N	sys-tem.-moab	ERR-OR	MWM_JOB_ADD_CLASS_ATTR	Cannot add class for job %s (Class: %s).	Unable to add a class requirement attrib-ute to a job.
0x210-0a7a8	ADMI-N	sys-tem.-moab	ERR-OR	MWM_JOB_ADD_DRM_ATTR	Cannot set des-tination RM for job %s (RM: %s).	Unable to add a des-tination resource manager attribute to a job.
0x210-0a7a9	ADMI-N	sys-tem.-moab	ERR-OR	MWM_JOB_FLAGS_INVALID_SOURCE	Attempting to set job flags from invalid format.	Job flags must be cre-ated using doc-umented formats.
0x210-0a7aa	ADMI-N	sys-tem.-moab	ERR-OR	MWM_SET_SIZE_ON_NONEXISTENT_REQ	Reqirement must be created before size is set.	Unable to set the size of an unallocated requirement.
0x210-0a7ab	ADMI-N	sys-tem.-moab	ERR-OR	MWM_ADD_GROUP_TO_JOB_FAILURE	Cannot add group for job %s (Group: %s).	Unable to set a group attribute on a job.
0x210-0a7ac	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NULL_JOB_NAME	Cannot add an empty name as an alternate name attribute for job %s.	No value specified. Make sure the altern-ate job name has a value.
0x210-0a7ad	ADMI-N	sys-tem.-moab	ERR-OR	MWM_SPACES_IN_JOB_NAME	Attempted to set a job name (%s) with space(s) for job %s.	A job name with space(s) was spe-cified. Job names can-not contain embedded spaces.
0x210-0a7ae	ADMI-N	sys-tem.-moab	ERR-OR	MWM_ADD_QOS_TO_JOB_FAILURE	Cannot add QOS for job %s (QOS: %s).	Unable to set a QOS attribute on a job.
0x210-0a7af	ADMI-N	sys-tem.-moab	ERR-OR	MWM_ADD_SRM_TO_JOB_FAILURE	Cannot add Submit RM for job %s (RM: %s).	Unable to find the entered name as an available resource manager.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a7b0	ADMI-N	sys-tem.-moab	ERR-OR	MWM_ADD_VARIABLE_TO_JOB_FAILURE	Cannot set variable for job %s (no vari-able name specified).	Only variables with names can be added as a job attribute.
0x210-0a7b1	ADMI-N	sys-tem.-moab	ERR-OR	MWM_ADD_USER_TO_JOB_FAILURE	Cannot add user for job %s (User: %s).	Unable to set a user attribute on a job.
0x210-0a7b2	ADMI-N	sys-tem.-moab	ERR-OR	MWM_ADD_NODE_TO_JOB_FAILURE	Cannot add node for job %s (Node: %s).	Unable to set a node attribute on a job.
0x210-0a7b3	ADMI-N	sys-tem.-moab	ERR-OR	MWM_ADD_ACCOUNT_TO_JOB_FAILURE	Cannot add account for job %s (Name: %s).	Failed to add account to the job.
0x210-0a7b5	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INVALID_TIME_STRING	Invalid format for time specification: '%s'.	A string that describes a time can-not be parsed because the format is wrong, or the values are out of range.
0x210-0a7b6	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_FIND_ARRAY_JOB	Cannot find array job at index %s for job '%s'.	Array job is missing.
0x210-0a7b7	ADMI-N	sys-tem.-moab	ERR-OR	MWM_JOB_BUFFER_FULL	Job buffer is full (ignoring job '%s').	Ignoring job since job buffer is full. Try increasing the value specified for the MAXJOB parameter.
0x210-0a7b8	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_FIND_MASTER_JOB	Cannot find master job (%s) for job '%s'; job array slot limits may not be enforced.	Cannot find the mas-ter job that is asso-ciated with a job array.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x2100a7b9	ADMIN	system.moab	ERROR	MWM_INVALID_ACTION_STRING	The action string (%s) is invalid.	The format of the action string is '<operation type>:<operation ID>:<operation action>' Example: job:145+146+147:cancel where 145,146 and 147 are job IDs.
0x2100a7ba	ADMIN	system.moab	ERROR	MWM_INVALID_OBJECT_TYPE	The object type %s is invalid.	The format of the action string is '<operation type>:<operation ID>:<operation action>' Example: job:145+146+147:cancel where 145,146 and 147 are job IDs.
0x2100a7bb	ADMIN	system.moab	ERROR	MWM_JOB_NOT_FOUND	Unable to locate job %s.	The named job was not located in the system.
0x2100a7bc	ADMIN	system.moab	ERROR	MWM_JOB_IN_BAD_STATE_FOR_COMPLETE	Completed trigger action is specified for job %s but it is in an invalid state.	The job is not a system job and is not allowed to be started by the resource manager.
0x2100a7bd	ADMIN	system.moab	ERROR	MWM_JOB_CANNOT_BE_HELD	Job %s cannot be put into hold state.	The resource manager cannot hold the job, usually because the job is not in a state that can be held.
0x2100a7be	ADMIN	system.moab	ERROR	MWM_CANNOT_SET_TRIGVAR	Cannot set trigger variable on job %s.	The trigger variables on a job cannot be set.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x2100a7bf	ADMIN	system.moab	ERROR	MWM_CANNOT_SET_REQATTR	Cannot set request attribute variable on job %s.	The request attribute variables on a job cannot be set.
0x2100a7c0	ADMIN	system.moab	ERROR	MWM_CANNOT_ADJUST_GRES	Cannot adjust generic resources for job %s.	The generic resources of the job could not be modified.
0x2100a7c1	ADMIN	system.moab	ERROR	MWM_INVALID_GRES_VALUE	Invalid value '%s' for GRes '%s' %s.	The value being set on the generic resource is not valid.
0x2100a7c2	ADMIN	system.moab	ERROR	MWM_CANNOT_MODIFY_ATTRIBUTE	Attribute %s cannot be modified for job %s.	The job's attribute could not be modified.
0x2100a7c3	ADMIN	system.moab	ERROR	MWM_COULD_NOT_SEND_SIGNAL	Signal %s could not be sent to job %s.	The resource manager was unable to send the signal to the job.
0x2100a7c4	ADMIN	system.moab	ERROR	MWM_COULD_NOT_START_JOB	Could not start job %s in %s.	The resource manager was unable to start the job.
0x2100a7c5	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_REQUEUE_JOB	Cannot requeue job %s.	The job could not be requeued.
0x2100a7c6	ADMIN	system.moab	ERROR	MWM_UNHANDLED_ACTION	The action %s was not handled.	The action was undefined in this function.
0x2100a7c7	ADMIN	system.moab	ERROR	MWM_UNRECOGNIZED_ATTRIBUTE	The attribute %s is not recognized.	The attribute is not in the lookup table.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x2100a7c8	ADMIN	system.moab	ERROR	MWM_UNRECOGNIZED_JOB_ACTION	The job action %s is not recognized.	The job action is not in the lookup table.
0x2100a7c9	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_CANCEL_JOB	Job %s could not be canceled.	The job could not be canceled.
0x2100a7ca	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_HOLD_JOB	Job %s could not be held in.	The job was unable to be put into a hold state.
0x2100a7cb	ADMIN	system.moab	ERROR	MWM_INVALID_PBS_SBINDIR	Invalid SBINDIR specified (%s).	Check paths for the directory containing pbs_iff.
0x2100a7cc	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_CONNECT_PBS_SRVR	Cannot connect to PBS server '%s'; rc: %s (pbs_errno=%s, '%s').	Make sure the pbs_server process is running.
0x2100a7cd	ADMIN	system.moab	ERROR	MWM_UNABLE_GET_SRVR_INFO	Cannot get server info: %s.	Make sure that the pbs_server process is running.
0x2100a7ce	ADMIN	system.moab	ERROR	MWM_UNABLE_LOAD_SRVR_INFO	Cannot load PBS server info: %s.	Make sure that the pbs_server process is running.
0x2100a7cf	ADMIN	system.moab	ERROR	MWM_UNABLE_LOAD_PBS_CLUSTER	Cannot load PBS cluster info: %s.	Make sure that the pbs_server process is running.
0x2100a7d0	ADMIN	system.moab	ERROR	MWM_UNABLE_LOAD_PBS_WORKLOAD	Cannot load PBS workload info: %s.	Make sure that the pbs_server process is running.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a7d1	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_LOAD_PBS_QUEUE	Cannot load PBS queue info: %s.	Make sure the path to the queue con-figuration is access-ible by Moab.
0x210-0a7d3	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_PROCESS_NODE_INFO	Cannot process node info.	Make sure the resource manager is running.
0x210-0a7d4	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NODE_BUFFER_FULL	Node buffer is full (ignoring node '%s').	Try increasing the node buffer.
0x210-0a7d5	ADMI-N	sys-tem.-moab	ERR-OR	MWM_JOB_CANNOT_START	Job '%s' cannot be started: (cannot gen-erate Tasklist).	Check the PBS server log to see reason of failure.
0x210-0a7d6	ADMI-N	sys-tem.-moab	ERR-OR	MWM_JOB_CANNOT_START_TASK_EMPTY	Job '%s' cannot be started: (empty Task-list).	Check the PBS server log to see reason of failure.
0x210-0a7d7	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_SET_NODE_COUNT	Cannot set nodecount for job '%s' - %s.	Check the PBS server log to see reason of failure.
0x210-0a7d8	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_SET_WALLTIME	Cannot set walltime for job '%s' - %s.	Check the PBS server log to see reason of failure.
0x210-0a7d9	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_SET_TASKLIST	Cannot set Tasklist for job '%s' - %s.	Check the PBS server log to see reason of failure.
0x210-0a7da	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_TO_START_JOB_RC	Job '%s' cannot be started: (rc: %s; errmsg: '%s'; Task-list: '%s').	Check the PBS server log to see reason of failure.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a7db	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_TO_SIGNAL_JOB	%s' cannot be sig-nalled: %s.	Check the PBS server log to see reason of failure.
0x210-0a7dc	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_TO_SUSPEND_JOB	Job '%s' cannot be suspended: %s.	Check the PBS server log to see reason of failure.
0x210-0a7dd	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_TO_RESUME_JOB	Job '%s' cannot be resumed: %s.	Check the PBS server log to see reason of failure.
0x210-0a7de	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_TO_FIND_RESOURCE	Failed to find/add %s generic resource.	Failure to find/add GPUs/MICs to the global GRES/MIC slots.
0x210-0a7df	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_TO_SET_CREDENTIALS	Cannot authenticate job '%s' (U: %s; G: %s; A: '%s').	Could not set the cre-dentials on the job.
0x210-0a7e0	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_TO_REQUEUE	PBS job '%s' cannot be requeued (rc: %s; '%s').	Check the PBS server log to see reason of failure.
0x210-0a7e1	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_TO_CHECKPOINT	PBS job '%s' cannot be checkpointed (rc: %s; '%s').	Check the PBS server log to see reason of failure.
0x210-0a7e2	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_TO_RELEASE	PBS job '%s' cannot be released from hold (rc: %s; '%s').	Check the PBS server log to see reason of failure.
0x210-0a7e5	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_TO_FIND_ACCOUNT	Cannot find account for job %s (Name: %s).	Make sure the account exists.
0x210-0a7e6	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INVALID_ARGUMENT	Command '%s' args not handled.	An unsupported argument was used.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a7e7	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INVALID_LOGDIR	LogDir '%s' is invalid.	Make sure that the path to the logs dir-ectory exists.
0x210-0a7e8	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INVALID_SPOOLDIR	SpoolDir '%s' is invalid.	Make sure that the path to the spool dir-ectory exists.
0x210-0a7e9	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INVALID_STATDIR	StatDir '%s' is invalid.	Make sure that the path to the stat dir-ectory exists.
0x210-0a7ea	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INVALID_TOOLSDIR	ToolsDir '%s' is invalid.	Make sure that the path to the tools dir-ectory exists.
0x210-0a7eb	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_CREATE_DAT_FILE	Cannot cre-ate/modify dat file: '%s'.	Moab encountered an error creating the dat file.
0x210-0a7ec	ADMI-N	sys-tem.-moab	ERR-OR	MWM_FEATURE_NOT_AVAILABLE_IN_BUILD	The '%s' feature is not available in the build of Moab.	Moab can be con-figured with various features. The listed feature is not avail-able in the binary being run.
0x210-0a7ed	ADMI-N	sys-tem.-moab	ERR-OR	MWM_FEATURE_NOT_AVAILABLE_WITH_LICENSE	The '%s' feature is not enabled with the current Moab license.	Moab can be licensed with various features. The listed feature is not available with the current license. Contact Adaptive Computing for more information.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x2100a7ee	ADMIN	system.moab	ERROR	MWM_RESOURCE_LIMIT_EXCEEDED	The maximum number of '%s' (%s) has been reached.	Moab has certain resources that are limited. This error occurs when you have reached or exceeded those limits. Contact Adaptive Computing for more information.
0x2100a7f1	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_VM_MIGRATION_JOB	Failed to create migration job for VM %s.	The migration job was not created. Check MIGRATETEMPLATE on workflow and its trigger.
0x2100a7f2	ADMIN	system.moab	ERROR	MWM_CANNOT_OPEN_EXTENSION_INTERFACE	Cannot open extension interface socket on port %s.	There was a failure opening the HTTP extension service. This feature will not work until the problem is corrected.
0x2100a7f3	ADMIN	system.moab	ERROR	MWM_JOB_USER_AUTHENTICATION	The system was unable to connect the given user to job %s (User: %s, Group: %s).	Check the credentials of the given user and/or group.
0x2100a7f4	ADMIN	system.moab	ERROR	MWM_JOB_AUTHENTICATION	The system was unable to authenticate the user connected with job %s (User: %s, Group: %s, Account %s) - %s.	Check the credentials of the given user and/or group.
0x2100a7f5	ADMIN	system.moab	ERROR	MWM_SEND_DATA_FAILED	The system was unable to send data to the server %s (%s:%s).	Make sure the server's address is correct and that the server is running.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a7f6	ADMI-N	sys-tem.-moab	ERR-OR	MWM_RECEIVE_DATA_FAILED	The system was unable to receive data from the server %s (%s:%s).	Make sure the server's address is correct and that the server is running.
0x210-0a7f7	ADMI-N	sys-tem.-moab	ERR-OR	MWM_JOB_OVERLAP	Job '%s' overlaps an existing job.	Check the job being created for overlap.
0x210-0a7f8	ADMI-N	sys-tem.-moab	ERR-OR	MWM_JOB_CREATION	The system was unable to create job '%s'	Verify that the job being created is cor-rectly specified.
0x210-0a7f9	ADMI-N	sys-tem.-moab	ERR-OR	MWM_MISSING_STATUS_ELEMENT	The status element was missing from the S3 response.	This is an internal error.
0x210-0a7fa	ADMI-N	sys-tem.-moab	ERR-OR	MWM_VC_WORKFLOW_JOB	Virtual container '%s' was marked as workflow, but could not find job that cre-ated it.	This is an internal error.
0x210-0a7fb	ADMI-N	sys-tem.-moab	ERR-OR	MWM_VC_COMBINE_JOBS	Failed to combine jobs in virtual con-tainer '%s'.	This is an internal error.
0x210-0a7fc	ADMI-N	sys-tem.-moab	ERR-OR	MWM_VC_SCHEDULE_TIME_FAILURE	Failed to schedule virtual container '%s' for requested time.	This is an internal error.
0x210-0a7fd	ADMI-N	sys-tem.-moab	ERR-OR	MWM_VC_RESERVATION_FAILURE	Failed to find a reser-vation for virtual con-tainer '%s'.	This is an internal error.
0x210-0a7fe	ADMI-N	sys-tem.-moab	ERR-OR	MWM_VC_RESERVATION_CREATE_FAILURE	Failed to create a reservation for jobs in virtual container '%s'.	This is an internal error.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a7ff	ADMI-N	sys-tem.-moab	ERR-OR	MWM_VC_RESOURCE_FAILURE	Requested resources are not available at any time for virtual container '%s'.	This is an internal error.
0x210-0a800	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NONEXISTING_JOB_USER	Job template %s requests non-exist-ent user %s.	Make sure the user exists.
0x210-0a801	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NONEXISTING_JOB_GROUP	Job template %s requests non-exist-ent group %s.	Make sure the group exists.
0x210-0a802	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NONEXISTING_JOB_QOS	Job template %s requests non-exist-ent QoS %s.	Make sure the QoS exists.
0x210-0a803	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_CREATE_CLASS	Unable to create class %s for job tem-plate %s.	Make sure the class exists.
0x210-0a804	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NONEXISTING_JOB_ACCOUNT	Job template %s requests non-exist-ent account %s.	Make sure the account exists.
0x210-0a805	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INVALID_WALLTIME_SPECIFIED	Invalid walltime spe-cification '%s.	Make sure the format for walltime is cor-rect.
0x210-0a806	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_PARSE_WIKI_STR	Cannot parse wiki string for job '%s'.	Make sure the format for wiki string is cor-rect.
0x210-0a807	ADMI-N	sys-tem.-moab	ERR-OR	MWM_MISSING_STATS_XML_ELEMENT	%s is not a valid tem-plate job stat child element.	Make sure there is a stats element in the XML.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a808	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NULL_NODE_POINTER	Node pointer is NULL and cannot be used to find SMP node.	Node pointer is NULL and cannot be used to find SMP node by node.
0x210-0a809	ADMI-N	sys-tem.-moab	ERR-OR	MWM_PINDEX_OUT_OF_RANGE	PIndex is less than -1 which is out of range.	PIndex must be greater than or equal to -1 to find a node by partition.
0x210-0a80a	ADMI-N	sys-tem.-moab	ERR-OR	MWM_FEATURE_OUT_OF_RANGE	Feature is less than -1 which is out of range.	Feature must be greater than or equal to -1 to find a node by feature.
0x210-0a80b	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INCORRECT_ARG	Incorrect argument in %s: %s, %s,%s.	Name must point to a valid string, Feature must be greater than or equal to -1, and N must point to a valid node.
0x210-0a80c	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NODE_ALLOCATION_ERROR	Failed to allocate a node named %s.	Call to MUMalloc failed, system is prob-ably low on memory.
0x210-0a80d	ADMI-N	sys-tem.-moab	ERR-OR	MWM_FAILED_TO_APPEND_MSMPNODE	Failed to append smpnode %s to MSMPNodes.	The call to append the node to the array list failed, probably due to a low memory condition.
0x210-0a80e	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NULL_SMPNODE_POINTER	Cannot initialize node because pointer is NULL.	Call to MSMPNodeIni-tialize must have a valid pointer to a valid node.
0x210-0a80f	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NULL_SMPNODE_POINTER_IN_RESET	Cannot reset node because pointer is NULL.	Call to MSMPNodeRe-setStats must have a valid pointer to a valid node.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a810	ADMI-N	sys-tem.-moab	ERR-OR	MWM_RESET_NODE_FAILED	Call to MSMPNodeRe-setStats failed.	Call to MSMPNodeRe-setStats failed. The most likely cause is passing a NULL pointer to SMPNode.
0x210-0a811	ADMI-N	sys-tem.-moab	ERR-OR	MWM_FREE_NODE_FAILED	Call to free MSMPNodes failed.	Call to free MSMPNodes failed, most likely due to cor-rupted memory.
0x210-0a812	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NULL_NODE_IN_UPDATE	Node pointer in %s cannot be NULL.	Node pointer cannot be NULL when trying to update node.
0x210-0a813	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_FIND_NODE	Unable to find SMP node with node %s.	Unable to find SMP node by node.
0x210-0a814	ADMI-N	sys-tem.-moab	ERR-OR	MWM_EMPTY_NODE_LIST	Updating node from list with empty node list.	Updating node from node list must not be called with an empty node list.
0x210-0a815	ADMI-N	sys-tem.-moab	ERR-OR	MWM_BAD_ARG_IN_FEASIBLE_JOB	Incorrect argument to function %s: %s, %s.	A parameter in the function was incor-rect.
0x210-0a816	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_FIND_INDEX_IN_LIST_FOR_FEATURE	Could not find index into NodeSetList for node feature %s.	Could not find index into NodeSetList for node feature.
0x210-0a817	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_CREATE_SR	Could not create standing reservation: %s.	Failed to create the named standing reservation.
0x210-0a818	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNEXPECTED_STATISTICS_TYPE	Unexpected statistics type: %s.	Number is not a member of MMStatTypeEnum enumeration.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a819	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_PROCESS_VM_ATTRIBUTE	Cannot process VM attribute %s for VM %s.	Either AttrName or NodeName is not found in string.
0x210-0a81a	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_FIND_NODE_FOR_VM	Cannot find node %s for VM %s.	The node does not exist or cannot be found.
0x210-0a81b	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_LOAD_JOB	Cannot load job %s (state: %s).	There was an error creating a job in Moab that was repor-ted by the resource manager.
0x210-0a81c	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_CREATE_CHECKPOINT_FILE_ENTRY	Cannot create check-point file entry.	There was an error writing a checkpoint file entry for the asso-ciated objects.
0x210-0a81d	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_CREATE_OBJECT_FROM_CHECKPOINT_FILE	Cannot create object from checkpoint file entry.	There was an error reading a checkpoint file entry for the asso-ciated objects.
0x210-0a81e	ADMI-N	sys-tem.-moab	ERR-OR	MWM_TASKLIST_TOO_LARGE	The tasklist for job '%s' is too large (size = %s, growth = %s).	The system has a fixed maximum size for the task map for each job.
0x210-0a81f	ADMI-N	sys-tem.-moab	ERR-OR	MWM_TASKLIST_MISSING	The tasklist for job '%s' is missing.	The system requires that each job has at least one task assigned.
0x210-0a820	ADMI-N	sys-tem.-moab	ERR-OR	MWM_TASK_DISTRIBUTION_UNKNOWN	The system encountered an unknown type of task distribution (%s).	This is an interal error.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a822	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INCOMPATIBLE_CHARGE_POLICY	Periodic charging dis-abled due to incom-patible charge policy (%s).	The charge policy is undefined.
0x210-0a823	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INCOMPLETE_JOB_TEMPLATE_ACTION	The job template '%s' has an incom-plete action spe-cification.	Job templates must fully specifiy the action to be per-formed.
0x210-0a824	ADMI-N	sys-tem.-moab	ERR-OR	MWM_INCOMPLETE_JOB_TEMPLATE_GENERIC	The job template '%s' has an incom-plete generic system job specification.	Job templates must fully specifiy the gen-eric system job.
0x210-0a825	ADMI-N	sys-tem.-moab	ERR-OR	MWM_DUPLICATE_JOB_TEMPLATE_VMID	The job template '%s' has a job '%s' that requests an existing VMID.	Virtual machine IDs cannot be shared across job templates.
0x210-0a826	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNKNOWN_JOB_TEMPLATE_VMID	The requested VMID '%s' could not be found or already has a tracking job.	Virtual machine IDs can only be assigned to a single job.
0x210-0a827	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_TO_MODIFY_JOB	The job '%s' on account '%s' cannot be modified in the resource manager.	The job previously submitted to the resource manager cannot be modified.
0x210-0a828	ADMI-N	sys-tem.-moab	ERR-OR	MWM_WORKFLOW_VC_FAILURE	The system failed to generate a workflow virtual container for job '%s'.	This is an internal error.
0x210-0a829	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CREATE_JOB_TEMPLATE_FAILURE	The system failed to create job template '%s'.	The job could not be created or one of its attributes could not be set.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x210-0a82a	ADMIN	system.moab	ERROR	MWM_VC_NOT_FOUND	The system could not find the virtual container for job '%s'.	This is an internal error.
0x210-0a82b	ADMIN	system.moab	ERROR	MWM_JOB_MIGRATION_FAILED	The system failed to migrate a remote job (%s).	Make sure the resource manager has not been disabled.
0x210-0a82c	ADMIN	system.moab	ERROR	MWM_JOB_START_XML_FAILURE	The system could not generate the coomand line needed to start job: '%s'.	The proper command line could not be derived from the XML structure.
0x210-0a82d	ADMIN	system.moab	ERROR	MWM_JOB_START_FAILURE_RESPONSE	The system could not start job - Reason: '%s'.	The system was unable to start the job for the specified reason.
0x210-0a82e	ADMIN	system.moab	ERROR	MWM_JOB_CANCEL_FAILURE_RESPONSE	The system could not cancel job - Reason: '%s'.	The system was unable to cancel the job for the specified reason.
0x210-0a82f	ADMIN	system.moab	ERROR	MWM_JOB_SIGNAL_FAILURE_RESPONSE	The system could not signal job - Reason: '%s'.	The system was unable to signal the job for the specified reason.
0x210-0a830	ADMIN	system.moab	ERROR	MWM_JOB_MODIFY_FAILURE_RESPONSE	The system could not modify job - Reason: '%s'.	The system was unable to modify the job for the specified reason.
0x210-0a831	ADMIN	system.moab	ERROR	MWM_JOB_REQUEUE_FAILURE_RESPONSE	The system could not requeue job - Reason: '%s'.	The system was unable to requeue the job for the specified reason.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x2100a832	ADMIN	system.moab	ERROR	MWM_SEND_EVENT_FAILURE	The system could send event '%s' to resource manager '%s' (%s).	The system was unable to send the event.
0x2100a833	ADMIN	system.moab	ERROR	MWM_UNEXPECTED_BACKFILL_POLICY	The system encountered an unexpected backfill policy '%s' (using '%s' instead).	The backfill policy did not match a defined policy.
0x2100a834	ADMIN	system.moab	ERROR	MWM_NODE_LIST_ALLOCATION	The system was unable to allocate a node list for job '%s' in partition '%s'.	The system may be low on memory.
0x2100a835	ADMIN	system.moab	ERROR	MWM_BAD_NODE_IN_NODELIST	The reservation nodelist for job' %s' has an invalid node at index %s.	Check the nodes specified for the reservation.
0x2100a836	ADMIN	system.moab	ERROR	MWM_RESERVATION_SPANS_PARTITIONS	The reservation request for job '%s' spans partitions (node %s partition %s).	Reservations that span partitions must have the COALLOC flag set.
0x2100a837	ADMIN	system.moab	ERROR	MWM_ADJUST_JOB_RESERVATION_FAILURE	The system failed to adjust job '%s' reservation on node %s.	This is an internal error.
0x2100a838	ADMIN	system.moab	ERROR	MWM_OBJECT_TYPE_INVALID	The object type specified (%s) is not valid.	A valid object type must be specified.
0x2100a839	ADMIN	system.moab	ERROR	MWM_MISSING_OBJECT_ID	The object ID is missing.	A valid object ID must be specified.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a83a	ADMI-N	sys-tem.-moab	ERR-OR	MWM_MISSING_ACTION	The action is missing.	A valid action must be specified.
0x210-0a83b	ADMI-N	sys-tem.-moab	ERR-OR	MWM_PIPE_BUFFER_FAILED	The system could not open a bi-directional pipe.	A valid action must be specified.
0x210-0a83c	ADMI-N	sys-tem.-moab	ERR-OR	MWM_STD_OUT_FAILED	Failed to load stdout file '%s'.	Check the file name and path.
0x210-0a83d	ADMI-N	sys-tem.-moab	ERR-OR	MWM_STD_ERR_FAILED	Failed to load stderr file '%s'.	Check the file name and path.
0x210-0a841	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CREATE_NODE_FAILURE	The system was unable to create node '%s'.	The system may be low on memory.
0x210-0a842	ADMI-N	sys-tem.-moab	ERR-OR	MWM_PARTITION_CREATE_FAILURE	The system was unable to create a shared partition for the global node.	The system may be low on memory.
0x210-0a845	ADMI-N	sys-tem.-moab	ERR-OR	MWM_HT_FIND_NODE_FAILURE	Cannot find node '%s' in hash table.	A node by the given name may not have been created.
0x210-0a847	ADMI-N	sys-tem.-moab	ERR-OR	MWM_HT_FIND_VM_FAILURE	Cannot find VM '%s' in hash table.	A VM with the given name may not have been created.
0x210-0a848	ADMI-N	sys-tem.-moab	ERR-OR	MWM_COMMAND_FAILED	Command '%s' failed. StatusCode: %s; Response: '%s'.	Check the command syntax and para-meters.
0x210-0a849	ADMI-N	sys-tem.-moab	ERR-OR	MWM_HASH_TABLE_INITIALIZATION	There was an unex-pected hash table ini-tialization error.	The hash table for jobs to delete never initialized correctly.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a84e	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNABLE_TO_AUTHENTICAT-E_JOB	Unable to authen-ticate job %s when UID or GID is empty (UID=%s, GID=%s).	Either the UID or the GID field is empty.
0x210-0a84f	ADMI-N	sys-tem.-moab	ERR-OR	MWM_MISSING_JOB_TASKCOUNT	Job does not have a taskcount specified.	Each job must have an associated taskcount.
0x210-0a850	ADMI-N	sys-tem.-moab	ERR-OR	MWM_FAILED_EXCLUDE_NODELIST	The system failed to add an exclude nodelist to a sub-mission.	The job exclude host-list could not be con-verted into a string.
0x210-0a852	ADMI-N	sys-tem.-moab	ERR-OR	MWM_CANNOT_SUBMIT_VM_MIGRATION_JOB	Failed to submit migration job for VM %s.	Check MIGRATETEMPLATE on workflow and its trigger.
0x210-0a853	ADMI-N	sys-tem.-moab	ERR-OR	MWM_WEB_SERVICES_WRITE_FAILURE	Error %s encountered while trying to write to web services.	Encountered prob-lem trying to put HTTP data to web server.
0x210-0a854	ADMI-N	sys-tem.-moab	ERR-OR	MWM_WEB_SERVICES_URL_MISSING	Missing URL in call to web services.	Web services must have a valid des-tination URL.
0x210-0a855	ADMI-N	sys-tem.-moab	ERR-OR	MWM_RM_PARTITION_CREATE_FAILURE	The system was unable to create a partition for RM '%s'.	The system may be low on memory.
0x210-0a856	ADMI-N	sys-tem.-moab	ERR-OR	MWM_PARSE_MPP_NODES_FAILURE	The system failed to parse the MPP nodes value '%s'.	Check the MPP names.
0x210-0a857	ADMI-N	sys-tem.-moab	ERR-OR	MWM_FIND_MPP_NODES_FAILURE	The system failed to find node '%s' in the MPP nodes value '%s'.	Check the MPP names.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x210-0a859	ADMIN	system.moab	ERROR	MWM_NODE_SET_TYPE_INVALID	The node set type specified (%s) is not valid.	Check the NODESETLIST option.
0x210-0a85a	ADMIN	system.moab	ERROR	MWM_GRES_ADD_FAILURE	Unable to add the GRESTOJOBATTRMAP '%s'.	The limit has been reached.
0x210-0a85d	ADMIN	system.moab	ERROR	MWM_NOT_MWS_RM	The resource manager is not Moab Web Services.	Make sure the resource manager has Moab Web Services.
0x210-0a85e	ADMIN	system.moab	ERROR	MWM_MWS_RM_CURL_CONNECTION	The system could not intialize a cURL connection to the MWS RM.	The cURL command to connect to the resource manager has failed.
0x210-0a85f	ADMIN	system.moab	ERROR	MWM_MWS_RM_CURL_CONNECTION_EXPANDED	Could not connect to MWS RM (%s) at '%s%s' as '%s', response code: %s; cURL error: %s (%s); MWS response: '%s'.	The connection has failed.
0x210-0a860	ADMIN	system.moab	ERROR	MWM_MWS_RM_JSON_CLUSTER_QUERY_EMPTY	JSON cluster query data from MWS RM (%s) is null or empty.	The query must contain valid JSON data.
0x210-0a861	ADMIN	system.moab	ERROR	MWM_MWS_RM_JSON_WORKLOAD_QUERY_EMPTY	JSON workload query data from MWS RM (%s) is null or empty.	The query must contain valid JSON data.
0x210-0a867	ADMIN	system.moab	ERROR	MWM_JOB_TRANSITION_FAILURE	Unable to transition a job.	The job was missing requirements.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x2100a868	ADMIN	system.moab	ERROR	MWM_SET_JOB_VARIABLE	Unable to set a job pref variable.	The system is probably low on memory.
0x2100a869	ADMIN	system.moab	ERROR	MWM_ARRAY_EXPANSION	Unable to expand the size of an array.	The system is probably low on memory.
0x2100a86c	ADMIN	system.moab	ERROR	MWM_VC_FIND_FAILURE	The system could not find the virtual container '%s'.	Check the name of the VC.
0x2100a86d	ADMIN	system.moab	ERROR	MWM_VC_USER_CREDENTIALS	User '%s' does not have access to virtual container '%s'.	Check the rights granted to the VC.
0x2100a86e	ADMIN	system.moab	ERROR	MWM_VC_BEING_DELETED	Virtual container '%s' is being deleted; cannot add jobs to it.	Only add jobs to VCs that are not being deleted.
0x2100a86f	ADMIN	system.moab	ERROR	MWM_PARTITION_STATUS	Unable to query the status of a partition - %s.	Check to make sure the resource manager is running.
0x2100a870	ADMIN	system.moab	ERROR	MWM_FIND_JOB_TEMPLATE	The system failed to find job template '%s'.	Check the template name for the given job.
0x2100a873	ADMIN	system.moab	ERROR	MWM_PROCESS_EVENT	Unable to process the generic event.	During processing, unable to get a description of the event.
0x2100a876	ADMIN	system.moab	ERROR	MWM_JOB_CAN_NEVER_RUN	Unable to allocate tasks for job at any time.	Job tasks must match available resources.
0x2100a877	ADMIN	system.moab	ERROR	MWM_NODE_NOT_IN_PARTITION	Node is not associated with any partition.	Node must be in a partition.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0a87a	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NODE_COUNT_EXCEEDS_LICENSE	The number of nodes '%s' exceeds the current license limit '%s'.	A different license is needed to use more nodes.
0x210-0a87b	ADMI-N	sys-tem.-moab	ERR-OR	MWM_KILL_FAILURE	OS call to kill process (PID: %s) %s failed).	This is an operating system error.
0x210-0a87c	ADMI-N	sys-tem.-moab	ERR-OR	MWM_MISSING_JOB_REQUIREMENTS	Job does not have any requirements specified.	Each job must have requirements attached.
0x210-0a87d	ADMI-N	sys-tem.-moab	ERR-OR	MWM_JOB_MISSING_DISPATCH_TIME	Job loaded in alloc state '%s' with no dispatch time.	The job must have a dispatch time.
0x210-0a87e	ADMI-N	sys-tem.-moab	ERR-OR	MWM_UNEXPECTED_OBJECT_TYPE	The object type '%s' was not expected in this operation.	Verify that a valid object type is given.
0x210-0a87f	ADMI-N	sys-tem.-moab	ERR-OR	MWM_JOB_TRANSITION_XML	Unable to create XML element from job transition object.	The system may be low on memory.
0x210-0a881	ADMI-N	sys-tem.-moab	ERR-OR	MWM_VM_CREATE_RESERVATION	Cannot create reser-vation for VM '%s'.	Failed to create reser-vation for the given VM.
0x210-0aa08	ADMI-N	sys-tem.-moab	ERR-OR	MWM_VM_FIELD_VALUE	VM '%s' has an invalid '%s%s%s' field value.	The field value for the VM is invalid.
0x210-0aa09	ADMI-N	sys-tem.-moab	ERR-OR	MWM_NODE_FIELD_VALUE	Node '%s' has an invalid '%s%s%s' field value.	The field value for the node is invalid.
0x210-0aa0a	ADMI-N	sys-tem.-moab	ERR-OR	MWM_JOB_FIELD_VALUE	Job '%s' has an invalid '%s%s%s' field value.	The field value for the job is invalid.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x210-0c001	INTER-NAL	sys-tem.-moab	ERR-OR	MWM_NOT_IMPLEMENTED	Function %s has not been implemented yet.	This error is used when we've stubbed out code but do not expect it to be called in production envir-onments. It's not helpful except for internal diagnostics.
0x210-0e729	INTER-NAL	sys-tem.-moab	ERR-OR	MWM_CANNOT_SEND_TO_SOCKET_DETAILED	Cannot send %s of %s bytes to socket descriptor %s - errno: %s (%s).	The send() system call failed. Socket is blocked (select() indicated socket was available--check MTU).
0x210-0e77b	INTER-NAL	sys-tem.-moab	ERR-OR	MWM_CLIENT_COUNT_NEGATIVE	Client count fell below zero on socket %s.	This is an internal error. The number of client connections should always be zero or greater.
0x210-0e784	INTER-NAL	sys-tem.-moab	ERR-OR	MWM_HOSTLIST_MISSING	A hostlist was spe-cified but now it is NULL/EMPTY.	The job claims to have a specified host-list, but at the cur-rent point in processing no list can be found. This is most likely an internal problem.
0x210-0e7b4	INTER-NAL	sys-tem.-moab	ERR-OR	MWM_REQATTR_UNSUPPORTED_OPERATION	Operation (%s) not supported on required attributes (reqattrs).	See documentation for supported oper-ators allows on required attributes (reqattrs).
0x210-0e7ef	INTER-NAL	sys-tem.-moab	ERR-OR	MWM_VM_NOT_LINKED_TO_TRACKING_JOB	VM '%s' not linked to VMTracking job '%s' (linked to job '%s').	A VM must be asso-ciated with a tracking job.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x310-02a2c	USER	sys-tem.-moab	ALER-T	MWM_NO_TASKS_FOUND_ON_JOB	No tasks found for job '%s'.	Check job submission arguments for desired require-ments.
0x310-08385	ADMI-N	sys-tem.-moab	FATA-L	MWM_EXPIRED_LICENSE	%s License has expired.	A license file was found but it has expired. Please con-tact your sales rep-resentative at Adaptive Computing for assistance.
0x310-08386	ADMI-N	sys-tem.-moab	FATA-L	MWM_EVALUATION_EXPIRED	%s evaluation period has expired.	The evaulation period has expired. Please contact your sales representative at Adaptive Com-puting for assistance.
0x310-08387	ADMI-N	sys-tem.-moab	FATA-L	MWM_UNEXPECTED_LICENSE_ERROR	Moab will now exit. Unexpected error while reading license: %s	Moab was unable to verify that the license file was valid. Please contact your sales representative at Adaptive Computing for assistance.
0x310-0a712	ADMI-N	sys-tem.-moab	FATA-L	MWM_UNABLE_TO_ALLOCATE_MEMORY	Unable to allocate memory.	One or more calls to allocate memory failed.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x310-0a714	ADMIN	system.moab	FATAL	MWM_CANNOT_RESTORE_UID	Cannot restore EUID to '%s' for server, errno: %s (%s).	The setuid() system call failed. There was a failure resetting the UID of the process. This may be because the process is running as a different user. Use the errno and associated message to determine possible causes.
0x310-0a715	ADMIN	system.moab	FATAL	MWM_CANNOT_CHANGE_UID	Cannot change UID to user '%s' (UID: %s) errno: %s (%s).	The setuid() system call failed. Use the errno and associated message to determine possible causes.
0x310-0a716	ADMIN	system.moab	FATAL	MWM_CANNOT_RESTORE_GID	Cannot restore GID to '%s' for server, errno: %s (%s).	The setgid() system call failed. There was a failure resetting the GID of the process. This may be because the process is running in a different group. Use the errno and associated message to determine possible causes.
0x310-0a717	ADMIN	system.moab	FATAL	MWM_CANNOT_FORK_INTO_BACKGROUND	Cannot fork the process into the background, errno: %s (%s).	The fork() system call failed. Moab must do this to daemonize unless run with the '-d' flag. This is usually due to low system resources.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x310-0a71c	ADMI-N	sys-tem.-moab	FATA-L	MWM_CANNOT_CHANGE_OWNERSHIP_FILE_FATAL	Cannot change own-ership of %s file to uid:%s gid:%s errno: %s (%s).	The fchown() system call failed. Use the errno and associated message to determ-ine possible causes.
0x310-0a745	ADMI-N	sys-tem.-moab	FATA-L	MWM_CANNOT_GET_SERVER_HOSTNAME	Cannot determine hostname and attrib-ute '%s' of para-meter %s is not specified.	Moab failed to obtain system host name or ip address inform-ation from the oper-ating system.
0x310-0a746	ADMI-N	sys-tem.-moab	FATA-L	MWM_HA_MOAB_NOT_STARTED_ON_CORRECT_HOSTS	The server must be started on host '%s' or on alternate '%s' (currently on '%s').	Moab must be star-ted on either the primary or alternate host for high avail-ability.
0x310-0a747	ADMI-N	sys-tem.-moab	FATA-L	MWM_MOAB_NOT_STARTED_ON_CORRECT_HOST	The server must be started on host '%s' (currently on '%s').	Moab must be star-ted on specified host as identified by the SCHEDCFG para-meter.
0x310-0a749	ADMI-N	sys-tem.-moab	FATA-L	MWM_MOAB_ALREADY_RUNNING	Moab is already run-ning. Cannot open user interface socket on port %s.	Cannot open user interface socket, which is most likely caused by Moab already running.
0x310-0a74b	ADMI-N	sys-tem.-moab	FATA-L	MWM_CANNOT_LOCATE_FULL_PATH	Cannot locate the full path for '%s'.	Check the path to make sure the Moab executable is in it. Restart manually to work around this problem temporarily.
0x310-0a74c	ADMI-N	sys-tem.-moab	FATA-L	MWM_CANNOT_RESTART_SCHEDULER	Exec failed when attempting to restart the scheduler '%s' rc: %s.	Please check per-missions on this executable to correct and restart manually to work around.

Code	Escal-ation level	Topic	Sev-erit-y	Event name	Message tem-plate	Comment
0x310-0a750	ADMI-N	sys-tem.-moab	FATA-L	MWM_CANNOT_CONNECT_TO_DB_WITH_STRICT_CONFIG_CHECK_ON	StrictConfigCheck ON and cannot connect to DB--please check DB engine and con-figuration (%s).	Moab was unable to connect to the data-base and with strict configuration on Moab must exit.
0x310-0a751	ADMI-N	sys-tem.-moab	FATA-L	MWM_USER_NOT_AUTHORIZED_TO_RUN_THIS_PROGRAM	The user '%s' (UID: %s) is not author-ized to run this pro-gram.	The user has insuf-ficient privileges to run the program.
0x310-0a752	ADMI-N	sys-tem.-moab	FATA-L	MWM_PROBLEMS_WITH_KEY_FILE	Problems with key file.	Key file does not exist or ownership of key file is invalid.
0x310-0aa0d	ADMI-N	sys-tem.-moab	FATA-L	MWM_STRICT_CHECK_EXIT	Exiting because of strict configuration check.	Moab is configured to exit if there are any errors in con-figuration files or file/directory layout. One of these errors has occurred.
0x310-0c002	INTER-NAL	sys-tem.-moab	FATA-L	MWM_TESTING_FATAL	Testing with single argument: %s.	Internal error for testing diagnostics.
0x310-0e74f	INTER-NAL	sys-tem.-moab	FATA-L	MWM_CORRUPT_CHECKPOINT_FILE	Unable to read the checkpoint file.	Please contact Adapt-ive Computing for assistance.

MWS Event Dictionary

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x02000100	USER		INFO	Service Create	The service '{0}' was created	This marks when a service was created.
0x02000101	USER		INFO	Service Modify	The service '{0}' was modified	This marks when a service was modified.
0x02000102	USER		INFO	Service Transition	The service '{0}' took the '{1}' transition. It went from the '{2}' to the '{3}' phase.	This marks a service phase transition.
0x02000103	USER		INFO	Service Terminate	The service '{0}' was terminated	Service termination means that the resources are released and no more modifications may be made to the service or policies. However, it still resides in the database and shows as "Terminated".
0x02000104	USER		INFO	Service Delete	The service '{0}' was deleted and is no longer available	Service deletion occurs after termination and means that the service is fully removed from the database and will no longer be displayed in any queries to MWS.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x02000380	USER		INFO	Notification Condition Create (User)	The notification condition '{0}' was created: {1}	This marks when a notification condition was created at the user escalation level.
0x02000500	USER		INFO	Service Template Create	The service template '{0}' was created	This marks when a service template was created.
0x02000501	USER		INFO	Service Template Modify	The service template '{0}' was modified	This marks when a service template was modified.
0x02000502	USER		INFO	Service Template Delete	The service template '{0}' was deleted and is no longer available	This marks when a service template was deleted.
0x02000580	USER		INFO	Service Hook Start	The service hook definition '{0}' on service '{1}' was executed with an ID of '{2}'	This marks the execution of a service hook definition for a given service.
0x02000581	USER		INFO	Service Hook End	The running service hook '{0}' on service '{1}' finished execution with status '{2}': {3}	This marks the end of execution for a running service hook.
0x02000582	USER		INFO	Service Hook Timeout	The running service hook '{0}' on service '{1}' timed out after {2} seconds.	This marks the execution of a service hook definition for a given service.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x02000583	USER		INFO	Service Hook Error	There was an error running service hook definition '{0}' on service '{1}' with an ID of '{2}'	This signifies that internal service hook processing has failed and the service hook could not be run successfully.
0x02004080	POWER_ USER		INFO	Policy Modify	The policy '{0}' was modified	A policy modification occurs when a PUT client request is received by Moab Web Services for the policy resource.
0x02004381	POWER_ USER		INFO	Notification Condition Create (Power User)	The notification condition '{0}' was created: {1}	This marks when a notification condition was created at the power user escalation level.
0x02008200	ADMIN		INFO	Permission Create	The permission '{0}' was created	This marks when a permission was created.
0x02008201	ADMIN		INFO	Permission Delete	The permission '{0}' was deleted	This marks when a permission was deleted.
0x02008280	ADMIN		INFO	Principal Create	The principal '{0}' was created	This marks when a principal was created.
0x02008281	ADMIN		INFO	Principal Modify	The principal '{0}' was modified	This marks when a principal was modified.
0x02008282	ADMIN		INFO	Principal Delete	The principal '{0}' was deleted	This marks when a principal was deleted.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x02008300	ADMIN		INFO	Role Create	The role '{0}' was created	This marks when a role was created.
0x02008301	ADMIN		INFO	Role Modify	The role '{0}' was modified	This marks when a role was modified.
0x02008302	ADMIN		INFO	Role Delete	The role '{0}' was deleted	This marks when a role was deleted.
0x02008382	ADMIN		INFO	Notification Condition Create (Admin)	The notification condition '{0}' was created: {1}	This marks when a notification condition was created at the administrator escalation level.
0x02008400	ADMIN		INFO	Tenant Create	The tenant '{0}' was created	This marks when a tenant was created.
0x02008401	ADMIN		INFO	Tenant Modify	The tenant '{0}' was modified	This marks when a tenant was modified.
0x02008402	ADMIN		INFO	Tenant Delete	The tenant '{0}' was deleted	This marks when a tenant was deleted.
0x22004180	POWER_USER		ERROR	LDAP Connect	Error communicating with the LDAP server: {0}	This occurs when communication could not be established with the configured LDAP server.

Code	Escalation level	Topic	Severity	Event name	Message template	Comment
0x22008105	ADMIN		ERROR	Service Transitions Missed	Service '{0}' ('{1}') is in phase '{2}' but should be in phase '{3}' according to the state reported by MWM. It is probable that one or more phase transitions were missed. Any hooks associated with those transitions were probably not executed.	This happens when the phase of a service as recorded by MWS differs from the phase expected according to the state of the service's jobs as reported by MWM. This means that oneor more phase transitions were missed and thus any hooks on those transitions did not execute. This problem is most likely caused by either the message queue being misconfigured or MWS being down while MWM is running.
0x22008480	ADMIN		ERROR	Health Retrieval Failed	The health summary could not be retrieved successfully: {1}	This means a serious error occurred while attempting to retrieve the contents of the health summary REST resource from MWS.
0x22008481	ADMIN		ERROR	Health Create Failed	The notification condition for the failed health check could not be created: {1}	This marks when a notification condition failed to be created based on a health check.

Appendix D: Adjusting Default Limits

Moab is distributed in a configuration capable of supporting multiple architectures and systems ranging from a few processors to several thousand processors. However, in spite of its flexibility, for performance reasons, it still contains a number of default object limits parameters and static structures defined in header files. These limits constrain such things as the maximum number of jobs, reservations, and nodes that Moab can handle and are set to values that provide a reasonable compromise between capability and memory consumption for most sites. However, many site administrators want to increase some of these settings to extend functionality, or decrease them to save consumed memory. The most common parameters are listed in what follows. Parameters listed in the Moab configuration file (`moab.cfg`) can be modified by restarting Moab. To change parameters listed in `moab.h`, please contact technical support.

CLIENTMAXCONNECTIONS	
Location	`moab.cfg` (dynamic parameter)
Default	128
Max tested	---
Description	Maximum number of connections that can simultaneously connect to Moab.

JOBMAXNODECOUNT	
Location	`moab.cfg` (dynamic parameter)
Default	1024
Max tested	8192
Description	Maximum number of compute nodes that can be allocated to a job. After changing this parameter, Moab must be restarted for changes to take effect. The value cannot exceed that of the **MAXNODE** on page 973 parameter (specified in moab.cfg). If you specify a value higher than the limit set for the **MAXNODE** parameter, the value will match **MAXNODE**. **JOBMAXNODECOUNT** can also be specified within configure using --with-maxjobsize=<NODECOUNT>.

MAXGRES	
Location	`moab.cfg` (dynamic parameter)
Default	512

MAXGRES	
Max tested	---
Description	Total number of distinct generic resources that can be managed.

MAXJOB	
Location	`moab.cfg` (dynamic parameter)
Default	4096
Max tested	500,000
Description	Maximum number of jobs that can be evaluated simultaneously. (Can also be specified within **configure** using **--with-maxjobs=<JOBCOUNT>**.)

MAXRSVPERNODE	
Location	`moab.cfg` (dynamic parameter)
Default	48
Max tested	1024
Description	Maximum number of reservations a node can simultaneously support.

MMAX_ATTR	
Location	`moab.h`
Default	128
Max tested	512
Description	Total number of distinct node attributes (PBS node attributes/LL node features) that can be tracked.

MMAX_CLASS	
Location	`moab.h`
Default	24
Max tested	64
Description	Total number of distinct job classes/queues available.

MMAX_FSDEPTH	
Location	`moab.h`
Default	24
Max tested	32
Description	Number of active fairshare windows.

MAXNODE	
Location	`moab.cfg` (dynamic parameter
Default	5120
Max tested	160000
Description	Maximum number of compute nodes supported.

MMAX_PAR	
Location	`moab.h`
Default	32
Max tested	32
Description	Maximum number of partitions supported.

MMAX_QOS	
Location	`moab.h`
Default	128
Max tested	128
Description	Total number of distinct QoS objects available to jobs.

MMAX_RACK	
Location	`moab.h`
Default	200
Max tested	200
Description	Total number of distinct rack objects available within cluster.

MMAX_RANGE	
Location	`moab.h`
Default	2048
Max tested	2048
Description	Total number of distinct timeframes evaluated. **Note**: This is proportional to the size of the cluster and the number of simultaneously active jobs in the cluster. (Can be specified within `./configure` using `--with-maxrange=<RANGECOUNT>`.) Increasing this value will not increase the size of total memory consumed by Moab but may result in minor slowdowns in the evaluation and optimization of reservations.

MMAX_REQ_PER_JOB	
Location	`moab.h`
Default	5

MMAX_REQ_PER_JOB	
Max tested	64
Description	Total number of unique requirement structures a job can have. Limits the number of `-w` clauses in the mshow -a command. It also limits the number of `-l nodes=X+Y+Z` a normal HPC job can have.

JOBMAXTASKCOUNT	
Location	`moab.cfg` (dynamic parameter
Default	4096
Max tested	250000
Description	Total number of tasks allowed per job.

Moab currently possesses hooks to allow sites to create local algorithms for handling site specific needs in several areas. The `contrib` directory contains a number of sample local algorithms for various purposes. The `MLocal.c` module incorporates the algorithm of interest into the main code. The following scheduling areas are currently handled via the `MLocal.c` hooks.

- Local Job Attributes
- Local Node Allocation Policies
- Local Job Priorities
- Local Fairness Policies

Related topics

- Appendix I: Considerations for Large Clusters

Appendix E: Security

Moab provides role and host based authorization, encryption*, and DES, HMAC, and MD5 based authentication. The following sections describe these features in more detail.

- Authorization
  - Role Based Authorization Security Configuration
- Authentication

Authorization

Role Based Authorization Security Configuration

Moab provides access control mechanisms to limit how the scheduling environment is managed. The primary means of accomplishing this is through limiting the users and hosts that are trusted and have access to privileged commands and data.

With regard to users, Moab breaks access into three distinct levels.

Level 1 Moab Admin (Administrator Access)

Level 1 Moab administrators have global access to information and unlimited control over scheduling operations. By default, they are allowed to control scheduler configuration, policies, jobs, reservations, and all scheduling functions. They are also granted access to all available statistics and state information. Level 1 administrators are specified using the ADMINCFG[1] parameter.

Level 2 Moab Admin (Operator Access)

Level 2 Moab administrators are specified using the ADMINCFG[2] parameter. By default, the users listed under this parameter are allowed to change all job attributes and are granted access to all informational Moab commands.

Level 3 Moab Admin (Help Desk Access)

Level 3 administrators are specified via the ADMINCFG[3] parameter. By default, they are allowed access to all informational Moab commands. They cannot change scheduler or job attributes.

Configuring Role Based Access

Moab allows site specific tuning of exactly which functions are available to each administrator level. Moab also provides two additional administrator levels (**ADMINCFG[4]** and **ADMINCFG[5]**) that may be used for site specific needs.

To configure Moab role based access, use the ADMINCFG parameter.

```
ADMINCFG[1]    USERS=root,john SERVICES=ALL NAME=admin
ADMINCFG[3]    USERS=joe,mary  SERVICES=mdiag,mrsvctl,mcredctl NAME=power
ADMINCFG[5]    USERS=joy,blake SERVICES=NONE NAME=users
...
```

A *NONE* in services will still allow users to run showq and checkjob on their own jobs.

To determine the role of system users and what commands they can run, use the mcredctl -q role user:<USERID> command.

Using the **SERVICES** attribute of the **ADMINCFG** parameter, access to an arbitrary selection of services can be enabled on a per administrator-level basis. Possible services include the following:

Service	Description
changeparam	Change any scheduling policy or parameter (This command is deprecated. Use mschedctl -m instead).
checkjob	View detailed information for any job.
checknode	View detailed information for any node.
mbal	Perform real-time load-balancing of interactive commands.
mcredctl	View and modify credential attributes.
mdiag	Provide diagnostic reports for resources, workload, and scheduling.
mjobctl	Modify, control, and view jobs.
mnodectl	Modify, control, and view nodes.
mrmctl	Modify, control, and view resource managers.
mrsvctl	Modify, control, and view reservations.
mschedctl	Modify, control, and view scheduler behavior.
mshow	View existing configuration and predicted resource availability.
showstats	View all scheduler and credential statistics.
releaseres	Release all reservations (This command is deprecated. Use mrsvctl -r instead).

Service	Description
runjob	Immediately execute any job (see mjobctl -x).
setqos	Set QoS on any job (This command is deprecated. Use mjobctl -m instead).
setres	Create any reservation (This command is deprecated. Use mrsvctl -c instead).
setspri	Set system priority on any job (This command is deprecated. Use mjobctl -p instead).
showconfig	Show all scheduler configuration parameters (This command is deprecated. Use mschedctl -l instead).
showres	Show detailed information for any reservation.
showstate	Show detailed information for all jobs, including their locations, and display job error messages, if any.

Account and Class/Queue Admins

While the **ADMINCFG** parameter allows organizations to provide controlled access to scheduling objects, it does not allow for distribution along organizational boundaries. For example, a site may set up a level 3 administrator to be able to view statistics, diagnose jobs, and modify job priorities; it does not provide a way to differentiate one type of job from another. If a site administrator wanted to allow control based on the queue or account associated with a job, they would best accomplish this using the credential **MANAGERS** attribute.

A credential manager allows a user to be trusted to administer workload and policies for an associated subgroup of jobs. For example, in the configuration below, a number of queue and account managers are configured.

```
CLASSCFG[orion] MANAGERS=johns
CLASSCFG[xray]  MANAGERS=steve2
CLASSCFG[gamma] MANAGERS=steve2,jpw
ACCOUNTCFG[bio] MANAGERS=charles
```

By default, the specified managers can do anything to a job that the actual job owner could do. By default, this would include the ability to view cumulative and per job statistics, see job details, modify job priorities and holds, cancel and preempt jobs, and otherwise adjust policies and constraints within the associated credential.

Authentication (Interface Security)

Moab supports password-challenge, DES, HMAC, and MD5 based authentication. Authentication protocols may be specified on a per interface basis allowing independent realms of trust with per realm secret keys and even per realm authentication protocols.

Mauth Authentication

Mauth is a tool provided with Moab that provides client authentication services. With mauth enabled, each client request is packaged with the client ID, a timestamp, and an encrypted key of the entire request generated using the shared secret key.

This tool is enabled by providing a secret key. A random key is selected when the Moab `./configure` script is run and may be regenerated at any time by rerunning `./configure` and rebuilding Moab. If desired, this random key may be overridden by specifying a new key in the protected `.moab.key` file as in the example below:

> Moab must be shut down before setting a new secret key. Use the `service moab stop` or mschedctl -k commands to shut down Moab.

```
> vi /opt/moab/etc/.moab.key
(insert key)
> cat /opt/moab/etc/.moab.key
XXXXXXXX
# secure file by setting owner read-only permissions
> chmod 400 /opt/moab/etc/.moab.key
# verify file is owned by root and permissions allow only root to read file
> ls -l /opt/moab/etc/.moab.key
-r-------- 1 root root 15 2007-04-05 03:47 /opt/moab/.moab.key
```

> All directories in the path containing `.moab.key` must be owned by the root or primary Moab user It must not be writable by "other" in its permissions.

> If `.moab.key` is used, this protected file will need to be on each host that is authorized to run Moab client commands.

> By default, this file will be owned by the user root and its contents will be read by the mauth tool which provides client authorization services. If desired, the ownership of this file can be changed so long as this file is readable by the Moab server and the mauth tool. This can be accomplished if the Moab primary administrator, the owner of mauth, and the owner of `.moab.key` are the same.

> By default, it is up to the individual cluster administrators to determine whether to use the `.moab.key` file. For sites with source code, the use of `.moab.key` can be mandated by using `./configure --with-keyfile`.

> By default, mauth is located in the install `bin` directory. If an alternate name or alternate file location is desired, this can be specified by setting the **AUTHCMD** attribute of the CLIENTCFG parameter within the `moab.cfg` file as in the following example.

```
CLIENTCFG  AUTHCMD=/opt/sbin/mauth
```

Configuring Peer-Specific Secret Keys

Peer-specific secret keys can be specified using the CLIENTCFG parameter. This key information must be kept secret and consequently can only be specified in the `moab-private.cfg` file. With regard to security, there are two key attributes that can be set. (Other resource managers or clients such as Moab Accounting Manager or a SLURM/Wiki interface can also use the attributes to configure their authentication algorithms. The default, unless otherwise stated, is always *DES*. These attributes are listed in the table below:

AUTH	
Format	one of *ADMIN1*, *ADMIN2*, or *ADMIN3*
Default	---
Description	Specifies the level of control/information available to requests coming from this source/peer.
Example	`CLIENTCFG[RM:clusterB] AUTH=admin1 KEY=14335443`

AUTHTYPE	
Format	one of *DES*, *HMAC*, *HMAC64*, or *MD5*.
Default	*DES*
Description	Specifies the encryption algorithm to use when generating the message checksum.
Example	`CLIENTCFG[AM:mam] AUTHTYPE=HMAC64`

HOST	
Format	*<STRING >*
Default	---
Description	Specifies the hostname of the remote peer. Peer requests coming from this host will be authenticated using the specified mechanism. This parameter is optional.
Example	`CLIENTCFG[RM:clusterA] HOST=orx.pb13.com  KEY=banana6`

KEY	
Format	*<STRING >*
Default	---
Description	Specifies the shared secret key to be used to generate the message checksum.
Example	`CLIENTCFG[RM:clusterA] KEY=banana6`

The **CLIENTCFG** parameter takes a string index indicating which peer service will use the specified attributes. In most cases, this string is simply the defined name of the peer service. However, for the special cases of resource and allocation managers, the peer name should be prepended with the prefix `RM:` or `AM:` respectively, as in `CLIENTCFG[AM:mam]` or `CLIENTCFG[RM:devcluster]`.

> The first character of any secret key can be viewed by trusted administrators using specific diagnostic commands to analyze Moab interfaces. If needed, increase the length of the secret keys to maintain the desired security level.

Munge Authentication

Moab also integrates with MUNGE, an open source authentication service created by Lawrence Livermore National Laboratory (http://home.gna.org/munge/). MUNGE works with Moab to authenticate user credentials being passed between the Moab client and the Moab server or from Moab server to Moab server.

To set up MUNGE in a cluster or grid, download and install MUNGE on every node in the cluster or grid by following the installation steps found at http://home.gna.org/munge/. The MUNGE secret key must reside on each node in the cluster or grid. Before starting the Moab daemon, the MUNGE daemon must be running on all nodes.

To enable Moab to use MUNGE for authentication purposes, specify the MUNGE executable path in the moab.cfg file using **CLIENTCFG** and **AUTHCMD** as in the following example. The MUNGE executable path must reside in each client's `moab.cfg` file as well.

```
CLIENTCFG AUTHCMD=/usr/bin/munge
```

> Moab requires that the MUNGE and UNMUNGE executable names be "munge" and "unmunge" respectively. It also assumes that the UNMUNGE executable resides in the same directory as the MUNGE executable.

Configuring Munge Command Options

Moab also integrates with MUNGE command line options. For example, to set up Moab to use a specific socket that was created when the MUNGE daemon was started, use **CLIENTCFG** and **AUTHCMDOPTIONS** to specify the newly created socket. The **AUTHCMDOPTIONS** attribute, like **AUTHCMD**, must also reside in the client's `moab.cfg` file.

```
CLIENTCFG     AUTHCMD=/usr/bin/munge
CLIENTCFG     AUTHCMDOPTIONS="-S /var/run/munge/munge.socket.2"
```

Server Response Control

If a request is received that is corrupt or cannot be authenticated, Moab will report some limited information to the client indicating the source of the failure, such as "bad key," "malformed header," and so forth. In the case of highly secure environments, or to minimize the impact of sniffing or denial of service attacks, Moab can be configured to simply drop invalid requests. This is accomplished by adding the **DROPBADREQUEST** attribute to the **CLIENTCFG** parameter in the `moab-private.cfg` file as in the following example:

```
CLIENTCFG[DEFAULT] DROPBADREQUEST=TRUE
```

Interface Development Notes

Sample checksum generation algorithm code can be found in the Socket Protocol Description document.

Host Security for Compute Resources

Host level security can vary widely from one site to another with everything from pure on-your-honor based clusters to complete encrypted VLAN based network security and government approved per job scrubbing procedures being used. The following documentation describes some best practices in use throughout the industry.

Minimal Host Security Enforcement

For minimal host security, no additional configuration is required.

Medium Host Security Enforcement

- Login Access
  - PAM — Enable/disable access by modifying `/etc/security/access.conf`.
- Processes
  - Kill all processes associated with job user (dedicated).
  - Kill all processes associated with job session (dedicated/shared). Use `ps -ju` *<USER>* or *ps -js* <SESSID>.
- IPC (Inter-Process Communication)
  - Remove shared memory, semaphores, and message queues (use ipcs/ipcrm).
  - Remove named pipes.

- Network/Global File System Access
    - Explicitly unmount user home and global file systems.
- Local Temporary File Systems
    - Where possible, mount local file systems read-only.
    - Clear `/tmp`, `/scratch` and other publicly available local file systems.
    - Remove user files with shred; shred is a Linux command that first overwrites files completely before removing them, preventing remnant data from surviving on the hard drive.

Strict Host Security Enforcement

- VLAN creation
- Host rebuild
    - U.S Dept. of Energy Disk/File Sanitization (SCRUB)
    - U.S Dept. of Defense Scrubbing Software (DOD-5520)

Moab Access Portal Security Overview

The Moab Access Portal (MAP) security model is composed of several different components. First, users will use a Web browser to log in and interact with the Web server running MAP. This communication can be encrypted using industry standard SSL to protect usernames/passwords and other sensitive information that may be accessed by the user. (Instructions on how to set up SSL connections with popular Web servers and servlet engines are readily available on the Internet. A guide for setting up SSL with Apache is available in the MAP documentation.)

When a user logs in via their Web browser, the JSP interface passes this request to a back-end Java infrastructure that then uses an encrypted SSH connection to authenticate the user's credentials at the cluster/grid head node. After the credentials are authenticated and the SSH connection established, further communication between MAP and the cluster/grid head node occurs over the encrypted SSH connection. These three components provide an end-to-end security solution for Web-based job submission and workload management.

Appendix F: Initial Moab Testing

Moab has been designed with a number of key features that allow testing to occur in a *no risk* environment. These features allow you to safely run Moab in test mode even with another scheduler running whether it be an earlier version of Moab or another scheduler altogether. In test mode, Moab collects real-time job and node information from your resource managers and acts as if it were scheduling live. However, its ability to actually affect jobs (that is, start, modify, cancel, charge, and so forth) is disabled.

Moab offers the following test modes to provide a means for verifying such things as proper configuration and operation:

Scheduler Modes

Central to Moab testing is the **MODE** attribute of the SCHEDCFG parameter. This parameter attribute allows administrators to determine how Moab will run. The possible values for **MODE** are *NORMAL*, *MONITOR*, *INTERACTIVE*, and *SIMULATION*. For example, to request monitor mode operation, include the following in the `moab.cfg` file:

```
SCHEDCFG MODE=MONITOR
```

Normal Mode

If initial evaluation is complete or not required, you can place the scheduler directly into *production* by setting the **MODE** attribute of the **SCHEDCFG** parameter to *NORMAL* and (re)starting the scheduler.

Monitor Mode (or Test Mode)

Monitor mode allows evaluation of new Moab releases, configurations, and policies in a risk-free manner. In monitor mode, the scheduler connects to the resource manager(s) and obtains live resource and workload information. Using the policies specified in the `moab.cfg` file, the monitor-mode Moab behaves identical to a live or normal-mode Moab except the ability to start, cancel, or modify jobs is disabled. In addition, allocation management does not occur in monitor mode. This allows safe diagnosis of the scheduling state and behavior using the various diagnostic client commands. Further, the log output can also be evaluated to see if any unexpected situations have arisen. At any point, the scheduler can be dynamically changed from monitor to normal mode to begin *live* scheduling.

To set up Moab in monitor mode, do the following:

```
> vi moab.cfg
  (change the MODE attribute of the SCHEDCFG parameter from NORMAL to MONITOR)
> moab
```

Remember that Moab running in monitor mode will not interfere with your production scheduler.

Running Multiple Moab Instances Simultaneously

If running multiple instances of Moab, whether in simulation, normal, or monitor mode, make certain that each instance resides in a different home directory to prevent conflicts with configuration, log, and statistics files. Before starting each additional Moab, set the **MOABHOMEDIR** environment variable in the execution environment to point to the local home directory. Also, each instance of Moab should run using a different port to avoid conflicts.

> If running multiple versions of Moab, not just different Moab modes or configurations, set the *$PATH* variable to point to the appropriate Moab binaries.

To *point* Moab client commands (such as showq) to the proper Moab server, use the appropriate command line arguments or set the environment variable **MOABHOMEDIR** in the client execution environment as in the following example:

```
# point moab clients/server to new configuration
> export MOABHOMEDIR=/opt/moab-monitor
# set path to new binaries (optional)
> export PATH=/opt/moab-monitor/bin:/opt/moab-monitor/sbin:$PATH
# start Moab server
> moab
# query Moab server
> showq
```

> moabd is a safe and recommended method of starting Moab if things are not installed in their default locations.

Interactive Mode

Interactive mode allows for evaluation of new versions and configurations in a manner different from monitor mode. Instead of disabling all resource and job control functions, Moab sends the desired change request to the screen and asks for permission to complete it. For example, before starting a job, Moab may post something like the following to the screen:

```
Command:  start job 1139.ncsa.edu on node list test013,test017,test018,test021
Accept:  (y/n) [default: n]?
```

The administrator must specifically accept each command request after verifying that it correctly meets desired site policies. Moab will then execute the specified command. This mode is useful in validating scheduler behavior and can be used until configuration is appropriately tuned and all parties are comfortable with the scheduler's performance. In most cases, sites will want to set the scheduling mode to normal after verifying correct behavior.

Simulation Mode

Simulation mode is of value in performing a *test drive* of the scheduler or when a stable production system exists and an evaluation is desired of how various policies can improve the current performance. See the Simulations documentation for more information.

Appendix G: Integrating Other Resources with Moab

Moab can interface with most popular resource managers, many cluster services, and numerous general protocols. The following links provide additional information.

Compute Resource Managers

- TORQUE - Integration Guide, TORQUE documentation
- SLURM - Integration Guide, http://www.llnl.gov/linux/slurm
- WIKI - WIKI Integration Guide
- Cray XT/TORQUE - Integration Guide (html, pdf), http://www.cray.com

Provisioning Resource Managers

- xCAT - Validating an xCAT Installation for Use with Moab

Hardware Integration

- NUMA - Integration Guide

Compute Resource Managers

Moab-TORQUE Integration Guide

- Overview
- Integration Steps
  - Install TORQUE
  - Install Moab
  - Configure TORQUE
  - TORQUE/Moab Considerations
- Current Limitations
- Troubleshooting

Install TORQUE

- Install TORQUE

> Keep track of the PBS target directory, *$PBSTARGDIR*

Install Moab

- Untar the Moab distribution file.
- Change the directory to the `moab-<version>` directory.
- Run `./configure`.
- Specify the PBS target directory (**$PBSTARGDIR** from step 2.1) when queried by `./configure`.

Moab interfaces to PBS by utilizing a few PBS libraries and include files. If you have a non-standard PBS installation, you may need to modify `Makefile` and change **PBSIP** and **PBSLP** values and references as necessary for your local site configuration.

The `./configure` script automatically sets up Moab so that the user running configure will become the default Primary Moab Administrator (*$MOABADMIN*). This can be changed by modifying the **ADMINCFG** [1] **USERS**=*<USERNAME>* line in the Moab configuration file (`moab.cfg`). The primary administrator is the first user listed in the **USERS** attribute and is the ID under which the Moab daemon runs.

Some Tru64 and IRIX systems have a local `libnet` library that conflicts with PBS's `libnet` library. To resolve this, try setting **PBSLIB** to '*${PBSLIBDIR}/libnet.a -lpbs*' in the Moab `Makefile`.

Moab is 64-bit compatible. If PBS/TORQUE is running in 64-bit mode, Moab likewise needs to be built in this manner to use the PBS scheduling API (i.e., for IRIX compilers, add *-64* to **OSCCFLAGS** and **OSLDFLAGS** variables in the `Makefile`).

When starting both TORQUE and Moab it is best to have a small delay between starting the servers. In general (and especially for very fast or very large systems) this is recommended startup procedure:

- Start TORQUE.
- Start Moab with scheduling paused (moab -P) to give it a chance to load everything in the checkpoint file and to sync with TORQUE.
- Unpause Moab with mschedctl -r.

General Configuration for All Versions of TORQUE

- Make $MOABADMIN a PBS admin.
  - By default, Moab only communicates with the pbs_server daemons and the *$MOABADMIN* should be authorized to talk to this daemon (See suggestions for more information.).
- *(OPTIONAL)* Set default PBS queue, nodecount, and walltime attributes. (See suggestions for more information.)
- *(OPTIONAL - TORQUE Only)* Configure TORQUE to report completed job information by setting the qmgr**keep_completed** parameter:>

```
> qmgr -c 'set server keep_completed = 300'
```

PBS nodes can be configured as *time shared* or *space shared* according to local needs. In almost all cases, space shared nodes provide the desired behavior.

> PBS/TORQUE supports the concept of virtual nodes. Using this feature, Moab can individually schedule processors on SMP nodes. The online TORQUE documentation describes how to set up the `$PBS_HOME/server_priv/nodes` file to enable this capability. (For example, *<NODENAME>* **np**=*<VIRTUAL NODE COUNT>*)

Version-Specific Configuration for TORQUE

Do not start the pbs_sched daemon. This is the default scheduler for TORQUE; Moab provides this service.

> Moab uses PBS's scheduling port to obtain real-time event information from PBS regarding job and node transitions. Leaving the default qmgr setting of `set` **server scheduling**=*True* allows Moab to receive and process this real-time information.

Configure Moab

By default, Moab automatically interfaces with TORQUE/PBS when it is installed. Consequently, in most cases, the following steps are not required:

- Specify PBS as the primary resource manager by setting **RMCFG**[*base*] **TYPE**=*PBS* in the Moab configuration file (`moab.cfg`).

If a non-standard PBS installation/configuration is being used, additional Moab parameters may be required to enable the Moab/PBS interface as in the line **RMCFG**[*base*] **HOST**=*$PBSSERVERHOST* **PORT**=*$PBSSERVERPORT*. See the Resource Manager Overview for more information.

> Moab's user interface port is set using the SCHEDCFG parameter and is used for user-scheduler communication. This port must be different from the PBS scheduler port used for resource manager-scheduler communication.

TORQUE/Moab Considerations

The default meaning of a node for TORQUE and Moab are not the same. By default, a node is a host in TORQUE. The node may have one or more execution slots (procs) allocated to it in the $TORQUE_HOME/server_priv/nodes file. However, the number of nodes recognized by TORQUE is equivalent to the number of node entries in the *$TORQUE_HOME*/`server_priv/nodes` file. A node specification from qsub such as `-l nodes=2:ppn=2` will direct TORQUE to allocate to execution slots on two separate nodes.

Moab is more liberal in its interpretations of a node. To Moab, the qsub request above would be interpreted to mean allocate four tasks with at least two tasks on a node. Where TORQUE would require two nodes for the request, Moab will place all four tasks on the name node (host) if four execution slots are available.

If a cluster has four nodes with eight processors each, TORQUE still sees only four nodes. Moab sees 32 nodes. However, if a user made a qsub request with `-l nodes=10`, TORQUE would reject the request because there are only four nodes available. To enable TORQUE to accommodate Moab's more liberal node interpretation, the server parameter available_resources.nodect can be set as a server parameter in TORQUE. The value of **available_resources.nodect** should equal at least the number of execution slots in the cluster.

For our example, cluster **available_resources.nodect** should be *32*. With this parameter set, the user can now make a request such as `-l nodes=8:ppn=2`. In this example, the user is still limited to a maximum node request of 32.

With **available_resources.nodect** set in TORQUE, Moab can be directed to honor the default TORQUE behavior by setting JOBNODEMATCHPOLICY to *EXACTNODE*.

PBS Features Not Supported by Moab

Moab supports basic scheduling of all PBS node specifications.

Moab Features Not Supported by PBS

PBS does not support the concept of a job QoS or other extended scheduling features by default. This can be handled using the techniques described in the PBS Resource Manager Extensions section. See the Resource Manager Extensions Overview for more information.

Troubleshooting

On TRU64 systems, the PBS `libpbs` library does not properly export a number of symbols required by Moab. This can be worked around by modifying the Moab `Makefile` to link the PBS `rm.o` object file directly into Moab.

TORQUE/PBS Integration Guide - RM Access Control

Server Configuration

Using the PBS qmgr command, add the Moab administrator as both a *manager* and *operator*.

```
> qmgr
Qmgr: set server managers += <MOABADMIN>@*.<YOURDOMAIN>
Qmgr: set server operators += <MOABADMIN>@*.<YOURDOMAIN>
Qmgr: quit
```

For example:

```
> qmgr
Qmgr: set server managers += staff@*.ucsd.edu
Qmgr: set operators += staff@*.ucsd.edu
Qmgr: quit
```

If desired, the Moab administrator can be enabled as a manager and operator only on the host on which Moab is running by replacing "*.<YOURDOMAIN>" with "<MOABSERVERHOSTNAME>".

Mom Configuration (optional)

If direct Moab to pbs_mom communication is required, the mom_priv/config file on each compute node where pbs_mom runs should be set as in the following example:

```
$restricted *.<YOURDOMAIN>
$clienthost <MOABSERVERHOSTNAME>
```

For security purposes, sites may want to run Moab under a non-root user id. If so, and Moab-pbs_mom communication is required, the mom_priv/config files must be world-readable and contain the line '$restricted *.<YOURDOMAIN>'. (i.e., '$restricted *.uconn.edu')

TORQUE/PBS Config - Default Queue Settings

Default Queue

To set the default queue (the queue used by jobs if a queue is not explicitly specified by the user), issue the following:

```
>> qmgr
Qmgr: set system default_queue = <QUEUENAME>
Qmgr: quit
```

Queue Default Node and Walltime Attributes

To set a default of one node and 15 minutes of walltime for a particular queue, issue the following:

```
> qmgr
Qmgr: set queue <QUEUENAME> resources_default.nodect = 1
Qmgr: set queue <QUEUENAME> resources_default.walltime = 00:15:00
Qmgr: quit
```

Default System Wide Node and Walltime Attributes

To set system wide defaults, set the following:

```
> qmgr
Qmgr: set server resources_default.nodect = 1
Qmgr: set server resources_default.walltime = 00:15:00
Qmgr: quit
```

Moab-SLURM Integration Guide

- Overview
- SLURM Configuration Steps
- Moab Configuration Steps
  - Configuration for Standby and Expedite
  - Configuration for the Quadrics Switch

- Authentication
- Queue/Class Support
- Policies
- Moab Queue and RM Emulation
- SLURM High Availability

Overview

Moab can be used as the scheduler for the SLURM resource manager. In this configuration, the SLURM handles the job queue and the compute resources while Moab determines when, where and how jobs should be executed according to current cluster state and site mission objectives.

The documentation below describes how to configure Moab to interface with SLURM.

> For Moab-SLURM integration, Moab 6.0 or higher and SLURM 2.2 or higher are recommended. From the downloads page, the generic version is needed to install SLURM.

SLURM Configuration Steps

To configure SLURM to utilize Moab as the scheduler, the **SchedulerType** parameters must be set in the `slurm.conf` config file located in the SLURM `etc` directory (`/usr/local/etc` by default)

```
# slurm.conf
SchedulerType=sched/wiki2
```

The **SchedulerType** parameter controls the communication protocol used between Moab and SLURM. This interface can be customized using the wiki.conf configuration file located in the same directory and further documented in the SLURM Admin Manual.

Note: To allow sharing of nodes, the SLURM partition should be configured with 'Shared=yes' attribute.

Moab Configuration Steps

By default, Moab is built with WIKI interface support (which is used to interface with SLURM) when running the standard configure and make process.

To configure Moab to use SLURM, the parameter 'RMCFG' should be set to use the **WIKI:SLURM** protocol as in the example below.

```
# moab.cfg

SCHEDCFG[base] MODE=NORMAL
RMCFG[base] TYPE=WIKI:SLURM
...
```

Note: The **RMCFG** index (set to *base* in the example above) can be any value chosen by the site. Also, if SLURM is running on a node other than the one on which Moab is running, then the **SERVER** attribute of the RMCFG parameter should be set.

Note: SLURM possesses a **SchedulerPort** parameter which is used to communicate with the scheduler. Moab will auto-detect this port and communicate with SLURM automatically with no explicit configuration required. Do NOT set Moab's **SCHEDCFG[] PORT** attribute to this value, this port controls Moab client communication and setting it to match the **SchedulerPort** value will cause conflicts. With no changes, the default configuration will work fine.

Note: If the SLURM client commands/executables are not available on the machine running Moab, SLURM partition and other certain configuration information will not be automatically imported from SLURM, thereby requiring a manual setup of this information in Moab. In addition, the SLURM VERSION should be set as an attribute on the RMCFG parameter. If it is not set, the default is version 1.2.0. The following example shows how to set this line if SLURM v1.1.24 is running on a host named Node01 (set using the SERVER attribute).

```
# moab.cfg with SLURM on Host Node01

RMCFG[base] TYPE=WIKI:SLURM SERVER=Node01 VERSION=10124
...
```

Configuration for Standby and Expedite Support

SLURM's 'Standby' and 'Expedite' options are mapped to the Moab QOS feature. By default, when a SLURM interface is detected, Moab will automatically create a 'standby' and an 'expedite' QoS. By default, the 'standby' QoS will be globally accessible to all users and on all nodes and will have a lower than normal priority. Also by default, the 'expedite' QoS will not be accessible by any user, will have no node constraints, and will have a higher than normal priority.

Authorizing Users to Use 'Expedite'

To allow users to request 'expedite' jobs, the user will need to be added to the 'expedite' QoS. This can be accomplished using the MEMBERULIST attribute as in the following example:

```
MEMBERULIST

# allow josh, steve, and user c1443 to submit 'expedite' jobs
QOSCFG[expedite] MEMBERULIST=josh,steve,c1443
...
```

Excluding Nodes for 'Expedite' and 'Standby' Usage

Both 'expedite' and 'standby' jobs can be independently excluded from certain nodes by creating a QoS-based standing reservation.

Specifically, this is accomplished by creating a reservation with a logical-*not* QoS ACL and a hostlist indicating which nodes are to be exempted as in the following example:

```
MEMBERULIST

# block expedite jobs from reserved nodes
SRCFG[expedite-blocker] QOSLIST=!expedite
SRCFG[expedite-blocker] HOSTLIST=c001[3-7],c200
SRCFG[expedite-blocker] PERIOD=INFINITY

# block standby jobs from rack 13
SRCFG[standby-blocker] QOSLIST=!standby
SRCFG[standby-blocker] HOSTLIST=R:r13-[0-13]
SRCFG[standby-blocker] PERIOD=INFINITY
...
```

Quadrics Integration

If managing a cluster with a Quadrics high speed network, significant performance improvement can be obtained by instructing Moab to allocate contiguous collections of nodes. This can be accomplished by setting the NODEALLOCATIONPOLICY parameter to *CONTIGUOUS* as in the example below:

```
# moab.cfg

SCHEDCFG[cluster1]   MODE=NORMAL SERVER=head.cluster1.org
RMCFG[slurm]         TYPE=wiki:slurm
NODEALLOCATIONPOLICY CONTIGUOUS
...
```

Setting Up Authentication

By default, Moab will not require server authentication. However, if SLURM's `wiki.conf` file (default location is `/usr/local/etc`) contains the **AuthKey** parameter or a secret key is specified via SLURM's configure using the `--with-key` option, Moab must be configured to honor this setting. Moab configuration is specified by setting the resource manager **AUTHTYPE** attribute to *CHECKSUM* and the **KEY** value in the moab-private.cfg file to the secret key as in the example below.

```
# /usr/local/etc/wiki.conf

AuthKey=4322953
...
```

```
# moab.cfg

RMCFG[slurm]         TYPE=wiki:slurm AUTHTYPE=CHECKSUM
...
```

```
# moab-private.cfg

CLIENTCFG[RM:slurm]  KEY=4322953
...
```

Note: For the *CHECKSUM* authorization method, the key value specified in the `moab-private.cfg` file must be a decimal, octal, or hexadecimal value, it cannot be an arbitrary non-numeric string.

Queue/Class Support

While SLURM supports the concept of classes and queues, Moab provides a flexible alternative queue interface system. In most cases, sites can create and manage queues by defining partitions within SLURM. Internally, these SLURM partitions are mapped to Moab classes which can then be managed and configured using Moab's CLASSCFG parameter and mdiag -c command.

Policies

By default, SLURM systems only allow tasks from a single job to utilize the resources of a compute node. Consequently, when a SLURM interface is detected, Moab will automatically set the NODEACCESSPOLICY parameter to *SINGLEJOB*. To allow node sharing, the SLURM partition attribute '**Shared**' should be set to *FORCE* in the `slurm.conf` as in the example below:

```
# slurm.conf

PartitionName=batch Nodes=node[1-64] Default=YES MaxTime=INFINITE State=UP
Shared=FORCE
```

Moab Queue and RM Emulation

With a SLURM system, jobs can be submitted either to SLURM or to Moab. If submitted to SLURM, the standard SLURM job submission language must be used. If jobs are submitted to Moab using the msub command, then either LSF*, PBS, or Loadleveler* job submission syntax can be used. These jobs will be translated by Moab and migrated to SLURM using its native job language.

SLURM High Availability

If SLURM high availability mode is enabled, Moab will automatically detect the presence of the SLURM BackupController and utilize it if the primary fails. To verify SLURM is properly configured, issue the SLURM command 'scontrol show config | grep Backup'. To verify Moab properly detects this information, run 'mdiag -R -v | grep FallBack'.

Note: To use SLURM high availability, the SLURM parameter **StateSaveLocation** must point to a shared directory which is readable and writable by both the primary and backup hosts. See the `slurm.conf` man page for additional information.

Related topics

- SLURM Admin Manual
- SLURM's Moab Integration Guide
- Additional SLURM Documentation
- Wiki Overview

Installation Notes for Moab and TORQUE for Cray

Overview

Moab and TORQUE can be used to manage the batch system for Cray. This document describes how to configure Moab and TORQUE to bring Moab's unmatched scheduling capabilities to the Cray.

New to TORQUE 4.1, TORQUE now handles all communication with ALPS, specifically the pbs_mom. Previously, communication with ALPS was handled by a combination of Moab, scripts and TORQUE. In the new model, Moab treats TORQUE as a regular TORQUE cluster without any special configuration. TORQUE now uses an extra MOM called the alps_reporter MOM to communicate with ALPS regarding configured and available resources. From the information reported by the alps_reporter mom, TORQUE creates a virtual node for each Cray compute node. Previously, TORQUE only reported the login nodes.

Note: For clarity this document assumes that your SDB node is mounting a persistent `/var` filesystem from the bootnode. If you have chosen not to use persistent `/var` filesystems please be aware that the instructions below would have to be modified for your situation.

Upgrade Notes

When upgrading to TORQUE 4.1.0 and using the new Cray model as described in this document, there should be no running jobs. Jobs may be queued but not running.

Installing TORQUE on a Cray

These instructions are written for a partitioned system, with separate SDB and boot nodes. A combined SDB/boot node configuration is not supported.

For non-partitioned systems, change *sdb-p1* to *sdb* in these instructions.

Before beginning, note the SDB and login nodes' IDs as you will need them throughout the install process.

```
crayadm@smw> ssh root@boot-p1
boot# grep sdb /etc/hosts
10.128.0.32   nid00031       c0-0c0s0n3     sdb001  sdb002
10.131.255.253 sdb sdb-p1 syslog syslog-p1 ufs ufs-p1

boot# grep login /etc/hosts
10.128.0.3    nid00002       c0-0c0s1n0     login   login-p1       login1  castor-p1
```

In this example, and throughout this page, the login node has NID 2 and the SDB has NID 31.

1. Copy Moab/TORQUE software to SMW and boot node

```
workstation> scp -p /cray/css/release/cray/build/batch/moab-
torque/torque.5.0.1.tar.gz crayadm@smw:/home/crayadm/<yourusername>
workstation> scp -p /cray/css/release/cray/build/batch/moab-torque/moab-8.0.1-
SUSE11-linux-x86_64-torque.tar.gz crayadm@smw:/home/crayadm/<yourusername>

crayadm@smw> cd /home/crayadm//<yourusername>
crayadm@smw> scp -p torque-5.0.1.tar.gz root@boot-p1:/rr/current/software
crayadm@smw> scp -p moab-8.0.1-SUSE11-linux-x86_64-torque.tar.gz root@boot-
p1:/rr/current/software
```

2. Install TORQUE. SSH to the boot node and unpack the TORQUE tarball within xtopview.

```
crayadm@smw> ssh root@boot-p1
boot# xtopview -m "Installing TORQUE"
default/:/# cd /software/
default/:/# tar -zxvf torque-5.0.1.tar.gz

(or, if installing on an esMS)
esms# cd /path/to/software/
esms# tar -zxvf torque-5.0.1.tar.gz
```

3. Configure, build, and install TORQUE within xtopview.

```
default/:/# cd torque-5.0.1
default/:/# ./configure --prefix=/opt/torque/5.0.1 --with-server-
home=/var/spool/torque --with-default-server=sdb-p1 --enable-syslog --disable-gcc-
warnings --with-debug --with-modulefiles=/opt/modulefiles --with-job-create
CFLAGS="-DCRAY_MOAB_PASSTHRU"

(or, if installing on an esMS)
esms# cd /path/to/software/torque-5.0.1
esms# ./configure --prefix=/opt/torque/5.0.1 --with-server-home=/var/spool/torque -
-with-default-server=this-esms --enable-syslog --disable-gcc-warnings --with-debug
--with-modulefiles=/cm/local/modulefiles CFLAGS="-DCRAY_MOAB_PASSTHRU"
```

> *The server name in the example is sdb-p1. Change this to sdb on a non-partitioned system.*

4. Make and install TORQUE.

```
 default/:/# make
 default/:/# make packages
 default/:/# make install
 default/:/# ln -sf /opt/torque/5.0.1 /opt/torque/default # The previous default
symlink might need to be deleted first if it exists
 default/:/# exit
```

5. Copy the TORQUE server directory to the Moab server host.

```
 boot# cd /rr/current/var/spool
 boot# cp -pr torque /snv/31/var/spool
 boot# cp -pr torque /snv/2/var/spool
```

> *In this example, the SDB node has NID 31, and the login node NID 2.*

6. Set up TORQUE on the SDB node.

```
boot# ssh sdb-p1
sdb# export PATH=/opt/torque/default/sbin:/opt/torque/default/bin:$PATH
sdb# cd /software/torque-5.0.1
sdb# ./torque.setup root
root
pbs_server port is: 15001
trqauthd daemonized - port 15005
trqauthd successfully started
initializing TORQUE (admin: root@boot)

You have selected to start pbs_server in create mode.
If the server database exists it will be overwritten.
do you wish to continue y/(n)?

# Type y

sdb# qmgr
Qmgr: set server keep_completed = 60 # Number of seconds to keep completed jobs in
qstat
unset queue batch resources_default.nodes
set server acl_host_enable = true
set server acl_hosts += nid00002
set server acl_hosts += castor-p1 # Where castor-p1 is the hostname of the login
node
set server acl_hosts += sdb-p1
set server submit_hosts += login
set server submit_hosts += login-p1 # Only needed on partitioned systems
set server submit_hosts += castor-p1 # Where castor-p1 is the hostname of the login
node
set server submit_hosts += nid00002 # Where nid00002 is the NID of the login node
set server tcp_timeout = 30
set server query_other_jobs = True
set server disable_server_id_check = True
set queue batch resources_default.mppnppn=16  # On Cascade systems with aprun -j1
set as default
set server cray_enabled = True
set server resources_default.partition = castor # Where "castor" is the same as the
RMCFG[clustername] in moab.cfg
exit
```

7. Get the number of nodes available on the system.

```
sdb# echo Node count is $(($(apstat -v | grep XT | awk "{print \$3}")))
```

In the *nodes* / *nodect* commands, put in your system's number of nodes.

```
sdb# qmgr
set server resources_available.nodes = 20
set server resources_available.nodect = 20
set queue batch resources_available.nodes = 20
set queue batch resources_available.nodect = 20
exit
```

8. Create TORQUE `nodes` file.

```
sdb# vi /var/spool/torque/server_priv/nodes

   castor-p1 alps_login np=1000 # Where "castor-p1" is the hostname of your login
node
   sdb-p1 alps_reporter

 sdb# exit
```

*The **np** attribute is the number of processes that can be running at once. This number should be set appropriately high, depending on the number of nodes on the system.*

9. Install the `torque_server` init.d script on the SDB node.

There is a known issue with some of the init scripts included with TORQUE. If the included init script doesn't work for you, use the process below to create a workaround `torque_server` init.d script for your system.

```
boot# xtopview -n 31 -m "torque_server init.d"
node/31:/ # touch /etc/init.d/torque_server
node/31:/ # xtspec -n 31 /etc/init.d/torque_server
node/31:/ # chmod a+x /etc/init.d/torque_server
node/31:/ # vi /etc/init.d/torque_server

#!/bin/sh
#
# pbs_server    This script will start and stop the PBS Server
#
### BEGIN INIT INFO
# Provides:       pbs_server
# Required-Start: $local_fs network
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:  2 3 5
# Default-Stop:
# Description: Torque is a versatile batch system for SMPs and clusters
### END INIT INFO

PBS_DAEMON=/opt/torque/default/sbin/pbs_server
PBS_HOME=/var/spool/torque
PIDFILE=$PBS_HOME/server_priv/server.lock
export PBS_DAEMON PBS_HOME PIDFILE

# Source the library functions
. /etc/rc.status
rc_reset

[ -f /etc/sysconfig/pbs_server ] && . /etc/sysconfig/pbs_server
[ -x $PBS_DAEMON ] || exit


# How were we called
case "$1" in
      start)
              echo -n "Starting TORQUE Server: "
              ulimit -c unlimited
              if [ -r $PBS_HOME/server_priv/serverdb ]
              then
                      startproc $PBS_DAEMON $SERVER_ARGS
              else
                      startproc $PBS_DAEMON -t create $DAEMON_ARGS
              fi
              rc_status -v
              ;;
      stop)
              echo -n "Shutting down TORQUE Server: "
              killproc -p $PIDFILE $PBS_DAEMON
              rc_status -v
              ;;
      status)
              echo -n "Checking TORQUE Server: "
              checkproc -p $PIDFILE pbs_server
              rc_status -v
              ;;
      restart)
              $0 stop
```

```
                $0 start
                rc_status
                ;;
        try-restart)
                $0 status >/dev/null && $0 restart
                rc_status
                ;;
        reload|force-reload)
                echo -n "Reloading TORQUE Server: "
                killproc -p $PIDFILE pbs_server -HUP
                rc_status -v
                ;;
        *)
                echo "Usage: torque_server {start|stop|status|try-
restart|restart|force-reload|reload}"
                exit 1
esac
rc_exit
```

10. Install the `torque_mom` init.d script on the SDB (or other node that runs the alps_reporter service for TORQUE) and login nodes.

> There is a known issue with some of the init scripts included with TORQUE. If the included init script doesn't work for you, use the process below to create a workaround `torque_mom` init.d script for your system.

Example 3-177: SDB node

```
boot# xtopview -n 31 -m "torque_mom init.d"
node/31:/ #  touch /etc/init.d/torque_mom
node/31:/ #  xtspec -n 31 /etc/init.d/torque_mom
node/31:/ #  chmod +x /etc/init.d/torque_mom
node/31:/ #  vi /etc/init.d/torque_mom

#!/bin/sh
#
# pbs_mom       This script will start and stop the PBS Mom
#
### BEGIN INIT INFO
# Provides:       pbs_mom
# Required-Start: $local_fs
# Should-Start: pbs_server pbs_sched
# Required-Stop:
# Should-Stop:
# Default-Start:  2 3 5
# Default-Stop:
# Description: Torque is a versatile batch system for SMPs and clusters
### END INIT INFO

PBS_DAEMON=/opt/torque/default/sbin/pbs_mom
PBS_HOME=/var/spool/torque
PIDFILE=$PBS_HOME/mom_priv/mom.lock
export PBS_DAEMON PBS_HOME PIDFILE

ulimit -n 32768
# Source the library functions
. /etc/rc.status
rc_reset

[ -f /etc/sysconfig/pbs_mom ] && . /etc/sysconfig/pbs_mom
[ -x $PBS_DAEMON ] || exit

args=""
if [ -z "$PREVLEVEL" ];then
 # being run manually, don't disturb jobs
 args="-p"
fi

# How were we called
case "$1" in
      start)
              echo -n "Starting TORQUE Mom: "
              #ulimit -c unlimited
              /sbin/startproc $PBS_DAEMON $args $DAEMON_ARGS
              rc_status -v
              ;;
      purge)
              [ -f /var/lock/subsys/pbs_mom ] && $0 stop
              echo -n "Starting TORQUE Mom with purge: "
              startproc $PBS_DAEMON -r $DAEMON_ARGS
              rc_status -v
              ;;
      stop)
              echo -n "Shutting down TORQUE Mom: "
              /sbin/killproc -p $PIDFILE $PBS_DAEMON
              rc_status -v
              ;;
```

```
        status)
                echo -n "Checking TORQUE Mom: "
                checkproc -p $PIDFILE $PBS_DAEMON
                rc_status -v
                ;;
        restart)
                $0 stop
                sleep 1
                $0 start -p
                rc_status
                ;;
        try-restart)
                $0 status >/dev/null && $0 restart
                rc_status
                ;;
        reload|force-reload)
                echo -n "Re-reading TORQUE Mom config file: "
                killproc -p $PIDFILE -HUP pbs_mom
                rc_status -v
                ;;
        *)
                echo "Usage: torque_mom {start|stop|status|try-restart|restart|force-
reload|reload|purge}"
                exit 1
esac
```

Example 3-178: Login nodes

```
boot# xtopview -c login -m "torque_mom init.d"
class/login:/ # touch /etc/init.d/torque_mom
class/login:/ # xtspec -c login /etc/init.d/torque_mom
class/login:/ # chmod +x /etc/init.d/torque_mom
class/login:/ # vi /etc/init.d/torque_mom

# Use the same script as the SDB node above
```

11. Create the MOM configuration file. This must be done on every login node and also the alps_reporter node (typically the SDB) specified in the TORQUE `server_priv/nodes` file.

   a. First, determine if the ALPS path needs to be configured in the MOM configuration file.

   ```
   login# which apbasil
   /usr/bin/apbasil
   # No configuration change needed

   login# which apbasil
   /opt/cray/alps/5.0.2-2.0500.7827.1.1.ari/bin/apbasil
   # MOM configuration change is needed. This path needs to be declared in the mom_
   priv/config file.
   ```

   b. Create and populate the MOM configuration file on the nodes.

```
login and sdb# vi /var/spool/torque/mom_priv/config

$usecp *:/ufs /ufs
$usecp *:/home /home
$usecp *:/home/users /home/users
$usecp *:/scratch /scratch
$usecp *:/lus /lus
$usecp *:/extlus /extlus
$login_node true # For login node
$reporter_mom true # For SDB node
$apbasil_protocol 1.2
$prologalarm 120
$apbasil_path /opt/cray/alps/default/bin/apbasil # Only if needed. Use the path
discovered above.
```

c. If needed, you can add CPR information to the MOM configuration on the login nodes .

```
$checkpoint_run_exe /opt/cray/blcr/default/bin/cr_run
$checkpoint_script /opt/cray/cprbatchutils/default/libexec/checkpoint.torque
$restart_script /opt/cray/cprbatchutils/default/libexec/restart.torque
$remote_checkpoint_dirs /lus/scratch/BLCR_checkpoint_dir
```

12. Create the `torque.cfg` file (useful if having issues with LDAP users submitting jobs) on the SDB node.

```
sdb# vi /var/spool/torque/torque.cfg

   QSUBSENDUID true
   VALIDATEPATH FALSE
```

13. Install the `trqauthd` init.d script on the SDB and login nodes.

```
boot# xtopview -n <SDB or login> -m "trqauthd"
node/<SDB or login>:/ # cp /software/torque-5.0.1/contrib/init.d/suse.trqauthd
/etc/init.d/trqauthd
node/<SDB or login>:/ # chmod +x /etc/init.d/trqauthd
node/<SDB or login>:/ # vi /etc/init.d/trqauthd

   PBS_DAEMON=/opt/torque/default/sbin/trqauthd
```

14. Start the trqauth daemon on the SDB and login nodes.

```
<SDB or login># /etc/init.d/trqauthd start
```

Enabling node features for Cray compute nodes

Node features can be set for Cray compute nodes. To add node features to a Cray compute node, use the `cray_compute` keyword on designated nodes in the `nodes` file.:

```
# node_id cray_compute feature_name
2 cray_compute bigmem
```

Configuring TORQUE for ALPS 1.3

To configure TORQUE for ALPS 1.3, configure the `apbasil_protocol` parameter in `mom_priv/config` and set the `nppcu` server parameter. The `nppcu` parameter has three options that determine whether to use Hyper-Threading:

Table 3-6: nppcu values

Value	Description
0	Allow ALPS to choose
1	Hyper-Threading disabled (default)
2	Hyper-Threading enabled

When nppcu is set to *0* or *2*, pbs_nodes reports twice as many cores.

apbasil_protocol:

```
$apbasil_protocol 1.3
$loglevel 3
```

nppcu:

```
qmgr -c 'set server nppcu=1'
```

Installing Moab Workload Manager

1. Unpack the Moab tarball within xtopview.

```
boot# xtopview -m "Installing Moab"
default/:/# cd /software/
default/:/# tar -zxvf moab-8.0.1-linux-x86_64-torque-xt4.tar.gz
default/:/# cd moab-8.0.1
```

2. Configure, build, and install Moab within xtopview.

```
default/:/# ./configure --prefix=/opt/moab/8.0.1 --with-homedir=/var/spool/moab --
with-torque=/opt/torque/default --with-modulefiles=/opt/modulefiles --with-xt4

 (or, if installing on an esMS)
default/:/# ./configure --prefix=/opt/moab/8.0.1 --with-homedir=/var/spool/moab --
with-torque=/opt/torque/default --with-modulefiles=/cm/local/modulefiles --with-xt4

default/:/# make install
default/:/# ln -sf /opt/moab/8.0.1 /opt/moab/default # The previous default symlink
may need to be deleted first if it exists
default/:/# exit
```

3. Configure the moab.cfg file.

```
boot# cd /rr/current/var/spool/moab/etc
boot# vi moab.cfg

   Change the value of SCHEDCFG[Moab] to SERVER=sdb-p1:42559 # Leave the port
number as whatever default is present
   Change ADMINCFG[1] USERS=root to USERS=root,crayadm  # Where "crayadm" is the
administrative user
   If applicable, change TOOLSDIR from /opt/moab/8.0.1/tools to
/opt/moab/default/tools

   Change RMCFG[boot]    TYPE=PBS to
          RMCFG[castor]  TYPE=TORQUE # "castor" can be any logical name for the
partition

   For Moab version 6.x, use TYPE=NATIVE:XT4

   Example:
       RMCFG[tuna]  TYPE=TORQUE SUBMITCMD=/opt/torque/default/bin/qsub
FLAGS=asyncstart


   Add:
   RMPOLLINTERVAL        00:00:10
   DEFERTIME             00:05:00
   JOBNODEMATCHPOLICY    EXACTNODE
   NODECFG[DEFAULT]      OS=linux ARCH=XT
   NODEACCESSPOLICY      SINGLEJOB
   JOBMIGRATEPOLICY      IMMEDIATE
   NODEALLOCATIONPOLICY  PRIORITY
   NODECFG[DEFAULT]      PRIORITYF='PRIORITY'
   NODECFG[castor-p1]    Partition=login # Use a logical name such as "login" to
keep the MOM nodes in a separate
                                         # partition from the compute nodes
                                         # "castor-p1" in this case is the hostname
of the login node
   CLIENTCFG[DEFAULT]    DEFAULTSUBMITPARTITION=castor # Where "castor" is the name
of the partition (see RMCFG[castor] above)
   JOBMAXTASKCOUNT        <total number of processors>


   # Comment out USEDATABASE   INTERNAL

   # If using a re-purposed compute node as the alps_reporter MOM, add:
   NODECFG[nid00060]    Partition=login # Where nid00060 is the nid of the RCN

   # If necessary to ignore nodes (such as 24 core nodes on a primarily 32 core
system), add:
   IGNORENODES   57,58 # Where "57" and "58" are node hostnames
```

4. Configure the `config.xt4.pl` file.

```
boot# vi config.xt4.pl

$basilProtocol = "1.2";

# Uncomment the %loginReplaceTable line, and update the hostnames:
%loginReplaceTable = (nid00002 => "login-p1"); # Where "login-p1" is the exact
hostname of the login node, for example, "castor-p3"

# Uncomment:
$topologyOrdering = 1;

# If on a system that has undergone ALPS standardization (See the apbasil notes
above), replace the $torquePath line with:
my $torquePath = "/opt/torque/default/bin:/usr/bin:/opt/cray/alps/default/bin";
```

5. Copy Moab to the SDB node.

```
boot# cd /rr/current/var/spool/
boot# cp -pr moab /snv/31/var/spool/

For Moab version 6.x only:
boot# mv /snv/31/var/spool/moab/etc/moab.cfg /snv/31/var/spool/moab/

For all versions:
boot# mkdir -p /snv/2/var/spool/moab/etc /snv/2/var/spool/moab/log
boot# cp moab/etc/moab.cfg /snv/2/var/spool/moab/etc/
boot# cp moab/etc/config.xt4.pl /snv/2/var/spool/moab/etc/
```

6. Install the `moab` init.d script

> There is a known issue with some of the init scripts included with Moab. If the included init script doesn't work for you, use the process below to create a workaround `moab` init.d script for your system.

```
boot# xtopview -n 31 -m "Moab init.d"
node/31:/ # touch /etc/init.d/moab
node/31:/ # xtspec -n 31 /etc/init.d/moab
node/31:/ # chmod a+x /etc/init.d/moab
node/31:/ # vi /etc/init.d/moab
#!/bin/bash
#
# Starts the Moab daemon
#
# chkconfig: 345 96 6
# description: Moab Workload Manager
# processname: moab
#
### BEGIN INIT INFO
# Provides: Moab
# Required-Start: $local_fs $syslog $network $named
# Required-Stop: $local_fs $syslog $network $named
# Default-Start: 3 5
# Default-Stop: 0 1 2 6
# Short-Description: Moab daemon management
# Description: Start Moab Workload Manager
### END INIT INFO
#
# 1. This file should be installed as /etc/init.d/moab
#
# 2. Start Moab with:
#
#      /etc/init.d/moab start
#
# Source function library.
[ -f /etc/rc.status ] || exit 0
. /etc/rc.status


export MOABHOMEDIR=/var/spool/moab
export MOABPARCLEANUP=Full
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/torque/default/lib


prog=moab
path=/opt/moab/default/sbin
RETVAL=0

function start()
{
      echo -n "Starting $prog: "
      ulimit -s unlimited    # Increase stack size to unlimited
      ulimit -c unlimited  # Uncomment to preserve core files

      export MOABNOALLOCMASTER=1
      sleep 2
      startproc $path/moab

      RETVAL=$?
      echo
      [ $RETVAL -eq 0 ] && touch /var/lock/subsys/moab
      return $RETVAL
}
```

```
function stop()
{
      echo -n "Shutting down $prog: "
      killproc moab
      RETVAL=$?
      echo
      [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/moab
      return $RETVAL
}

function restart()
{
      stop
      sleep 2
      start
}

function condrestart()
{
      if [ -f /var/lock/subsys/capi ] ; then
              restart
      fi
}

function reload()
{
      echo -n $"Reloading $prog: "
      schedctl -R
      RETVAL=$?
      echo
      return $RETVAL
}

case "$1" in
  start)
              start
              rc_status -v
              ;;
  status)
              status moab
              RETVAL=$?
              rc_status -v
              ;;
  stop)
              stop
              rc_status -v
              ;;
  restart)
              restart
              rc_status -v
              ;;
  condrestart)
              condrestart
              ;;
  reload)
              reload
              rc_status -v
              ;;
  *)
```

```
      echo "Usage: $prog {start|stop|restart|reload|status|condrestart}"
      exit 1
esac

exit $RETVAL
```

7. For versions of Moab earlier than 7.1.3 and TORQUE 4.1.4, root must be allowed to submit jobs.

```
boot# ssh sdb
sdb# module load moab torque
sdb# qmgr
 qmgr> set server acl_roots = root
 qmgr> exit
sdb# /etc/init.d/torque_server restart

sdb# vi /var/spool/moab/etc/moab.cfg # or /var/spool/moab/moab.cfg for Moab version
6

 Add
 ALLOWROOTJOBS TRUE

sdb# /etc/init.d/moab restart
   or
sdb# mschedctl -R
```

8. For versions of Moab newer than 7.1.3, the root user can submit jobs as another user.

```
sdb# qmgr
 qmgr> set server managers += root@*
 qmgr> exit

sdb# ssh login

login# qsub -I -l mppwidth=1 -P <otherusername>
```

9. Copy your Moab license file to the SDB node in the `/var/spool/moab/` directory.

Provisioning Resource Managers

Validating an xCAT Installation for Use with Moab

Introduction to Validating xCAT Configuration

This document describes a series of steps to validate xCAT configuration prior to configuring Moab to manage hardware via xCAT. It is assumed the reader is familiar with xCAT and the xCAT configuration on the target site. This document does not provide xCAT configuration documentation or troubleshooting information; please refer to the xCAT documentation for such information.

Verifying Node List

Verify that all nodes that Moab will manage are known to xCAT with the xCAT nodels command. Ensure that all expected (and no unexpected) nodes are listed. You may find it useful to create new group names to identify Moab-managed nodes.

```
[root@h0 moab]# nodels hyper,compute
h1
h2
h3
h4
h5
h7
kvmm1
kvmm10
kvmm2
kvmm3
kvmm4
kvmm5
kvmm6
kvmm7
kvmm8
[root@h0 moab]#
```

Reporting Node Status

Verify that all nodes report their status correctly using the xCAT nodestat command. Ensure that all nodes show the correct status (`sshd`, `installing`, `noping`, and so forth); there should not be any timeouts or error messages.

```
[root@h0 moab]# nodestat hyper,compute |sort
h1: pbs,sshd
h2: pbs,sshd
h3: pbs,sshd
h4: pbs,sshd
h5: pbs,sshd
h7: noping
kvmm10: noping
kvmm1: pbs,sshd
kvmm2: pbs,sshd
kvmm3: pbs,sshd
kvmm4: pbs,sshd
kvmm5: pbs,sshd
kvmm6: pbs,sshd
kvmm7: pbs,sshd
kvmm8: noping
kvmm9: noping
[root@h0 moab]#
```

Verifying Hardware Management Configuration

Verify that all nodes that Moab will manage have hardware management interfaces correctly configured using the xCAT nodels and rpower commands. After each of the rpower commands, verify the requested state was achieved with rpower stat.

```
[root@h0 moab]# nodels h1,kvmm1 nodehm.mgt nodehm.power
h1: nodehm.power: ilo
h1: nodehm.mgt: ilo
kvmm1: nodehm.power: kvm
kvmm1: nodehm.mgt: kvm
[root@h0 moab]# rpower h1,kvmm1 off
h1: off
kvmm1: off
[root@h0 moab]# rpower h1,kvmm1 stat
h1: off
kvmm1: off
[root@h0 moab]# rpower h1,kvmm1 boot
h1: on reset
kvmm1: on reset
[root@h0 moab]# rpower h1,kvmm1 stat
h1: on
kvmm1: on
[root@h0 moab]#
```

Verifying Provisioning Images

Verify that all operating system images that Moab uses are configured correctly in xCAT. For stateful images, test that all combinations of operating system, architecture, and profile install correctly.

```
[root@h0 moab]# rinstall -o centos5.3 -a x86_64 -p hyper h1
h1: install centos3.2-x86_64-hyper
h1: on reset
[root@n100 ~]# sleep 15 && nodestat n05
n05: ping install centos5.3-x86_64-hyper
[root@h0 moab]#
```

For stateless images, test that nodes are able to network boot the images.

```
[root@h0 moab]# nodech h5 nodetype.os=centos5.3 nodetype.arch=x86_64
nodetype.profile=hyper
[root@h0 moab]# nodeset h5 netboot
h5: netboot centos5.3-x86_64-hyper
[root@h0 moab]# rpower h5 boot
h5: on reset
[root@h0 moab]# sleep 60 && nodestat h5
h5: pbs, sshd
[root@h0 moab]#
```

Verifying VM Migration

If you use VM migration, verify that xCAT can successfully perform migrations using the rmigrate command.

```
[root@h0 moab]# rmigrate kvmm7 h1
kvmm7: migrated to h1
[root@h0 moab]# ssh h1 virsh list
Id Name State
----------------------------------
33 kvmm1 running
34 kvmm2 running
35 kvmm7 running
```

Related topics

- Native Resource Manager Overview
- Resource Provisioning

Hardware Integration

- **Moab-NUMA Integration Guide** on page 1233

Moab-NUMA Integration Guide

Scheduling a shared-memory NUMA type system (not the same as a modern SMP-based individual compute node, which cannot share memory between compute nodes) requires some special configuration. Additionally, Moab can use NODESETs to guarantee feasibility of large memory jobs and to enforce node allocation based on the system's interconnect network topology.

Configuration

To integrate Moab and NUMA

1. Configure Moab to schedule large memory jobs. Because Moab creates a partition for each resource manager by default, you must configure the cluster controlled by the resource manager to be a shared-memory system to support jobs spanning multiple nodes/blades. To do so, use the PARCFG parameter.

```
RMCFG[sys-uv]  TYPE=TORQUE
PARCFG[sys-uv] FLAGS=SharedMem
```

Cluster `sys-uv` is now configured as a shared-memory system to Moab.

2. Configure **NODESET**s as shown below.

```
NODESETISOPTIONAL FALSE
NODESETATTRIBUTE FEATURE
NODESETPOLICY ONEOF
NODESETPRIORITYTYPE FIRSTFIT
```

*The **NODESET** parameters tell Moab that performing node allocation using node sets is required, that the node set name is a feature name assigned to compute nodes, that a job must fit within the available nodes of one node set, and that Moab should use the first node set that contains sufficient available nodes to satisfy the job's request.*

3. To configure Moab to perform topology-aware node allocation using node sets, you must create a node set definition for each set of nodes that has the same number of maximum network "hops" from any node to every other node within the node set. For an example, see the following sample scenario:

Use case

The SGI UV 1000 has a two-socket blade with a physical organization of 16 blades within a blade chassis (SGI term is Intra-Rack Unit or IRU), two blade chassis (IRUs) within a rack, and up to four racks within a single UV system. The UV 1000 interconnect network has a topology that requires zero hops between the two sockets on the same physical blade, one hop between an even-odd blade pair (e.g. blades 0 and 1, 2 and 3, etc.), two hops between all even-numbered or all odd-numbered blades within an IRU, three hops maximum between all blades within an IRU, four hops maximum between all even-numbered blades or all odd-numbered blades within a UV system, and five hops maximum between all blades within a UV system.

a. Define topology-aware node definitions to parallel the compute nodes reachable within a specific hop count. For the UV 1000, this means the sockets of each blade will belong to six separate node set definitions; i.e., one each for 0, 1, 2, 3, 4, and 5 hops).

b. Define multiple node sets for different nodes reachable in a specific hop count based on the context of where they are in the network topology; that is, you must create a separate and distinct node set definition for each pair of blades reachable with one hop, for each IRU for its nodes reachable in three hops, etc.

c. Moab node sets are usually defined as compute node features; that is, each node set defined to Moab should appear as a "feature" name on one or more compute nodes. Which node set/feature names appear on each compute node depends on where the compute node is in the interconnect network topology.

   Since the SGI UV operating system identifies each blade socket as a separate NUMA node, each NUMA node within a UV system is traditionally an individual compute node to Moab (although TORQUE has the ability to redefine a compute node definition by grouping OS NUMA nodes, which some UV installations do to define a blade as a compute node).

   For the sake of illustration, this example assumes each OS NUMA node, which is a UV blade socket, is also a compute node in Moab. This means each compute node (blade socket) will have six feature names assigned, where each feature name must reflect both the compute node's location in the network topology and the hop count the name represents. A feature name is constructed by using the same root name for a hop count and a number for the topology location at the hop-count level.

   For example, the root feature name "blade" represents the zero-hop count and the numbers "0", "1,", etc, represent the consecutively numbered blades throughout the entire UV system, which yields feature names of "blade0" for the first blade in the system, "blade1" for the second blade, etc, to "blade127" for the last blade in a fully populated 4-rack UV system. To illustrate further, the root feature name "iru" represents the 3-hops count and the numbers "0" through "7" represent the eight IRUs within a full 4-rack UV system.

d. For each compute node, configure the correct feature name for each of the hop counts possible and its location within the topology at the hop-count level (e.g., blade (0 hops), blade pair (1 hop), odd- or even-numbered nodes within an IRU (2 hops), IRU (3 hops), odd- or even-numbered nodes within the UV (4 hops), and UV system (5 hops)). The following example illustrates the feature names assigned to the compute nodes for an SGI UV 1000 system using the following root feature names.

- blade (0 hops)
- pair (1 hop)
- eiru (2 hops for even-numbered blades within an IRU)
- oiru (2 hops for odd-numbered blades within an IRU)
- iru (3 hops)
- esys (4 hops for even-numbered blades within a UV system)
- osys (4 hops for odd-numbered blades within a UV system)
- sys (5 hops)

Note that nodes 0 and 1 are not given any feature names. This is because the operating system instance for the UV system runs on the first blade and in order to not adversely affect OS performance, no jobs should run on the same compute resources as the operating system; hence, these nodes have no node set feature names and therefore will never be chosen to run jobs. In addition, some of the first feature names at a specific hop count-level are omitted (such as pair0) since it makes no sense to define them when the first blade is a substantial part of the nodes making up a node set.

The node name of a UV system has the same name as the UV system's host name plus the NUMA node's relative socket number.

```
/var/spool/torque/server_priv/nodes:
sys-uv0
sys-uv1
sys-uv2   blade1          oiru0 iru0 osys sys
sys-uv3   blade1          oiru0 iru0 osys sys
sys-uv4   blade2   pair1  eiru0 iru0 esys sys
sys-uv5   blade2   pair1  eiru0 iru0 esys sys
sys-uv6   blade3   pair1  oiru0 iru0 osys sys
sys-uv7   blade3   pair1  oiru0 iru0 osys sys
sys-uv8   blade4   pair2  eiru0 iru0 esys sys
sys-uv9   blade4   pair2  eiru0 iru0 esys sys
sys-uv10  blade5   pair2  oiru0 iru0 osys sys
sys-uv11  blade5   pair2  oiru0 iru0 osys sys
sys-uv12  blade6   pair3  eiru0 iru0 esys sys
sys-uv13  blade6   pair3  eiru0 iru0 esys sys
sys-uv14  blade7   pair3  oiru0 iru0 osys sys
sys-uv15  blade7   pair3  oiru0 iru0 osys sys
sys-uv16  blade8   pair4  eiru0 iru0 esys sys
sys-uv17  blade8   pair4  eiru0 iru0 esys sys
sys-uv18  blade9   pair4  oiru0 iru0 osys sys
sys-uv19  blade9   pair4  oiru0 iru0 osys sys
sys-uv20  blade10  pair5  eiru0 iru0 esys sys
sys-uv21  blade10  pair5  eiru0 iru0 esys sys
sys-uv22  blade11  pair5  oiru0 iru0 osys sys
sys-uv23  blade11  pair5  oiru0 iru0 osys sys
sys-uv24  blade12  pair6  eiru0 iru0 esys sys
sys-uv25  blade12  pair6  eiru0 iru0 esys sys
sys-uv26  blade13  pair6  oiru0 iru0 osys sys
sys-uv27  blade13  pair6  oiru0 iru0 osys sys
sys-uv28  blade14  pair7  eiru0 iru0 esys sys
sys-uv29  blade14  pair7  eiru0 iru0 esys sys
```

```
sys-uv30  blade15  pair7  oiru0 iru0 osys sys
sys-uv31  blade15  pair7  oiru0 iru0 osys sys
sys-uv32  blade16  pair8  eiru1 iru1 esys sys
sys-uv33  blade16  pair8  eiru1 iru1 esys sys
sys-uv34  blade17  pair9  oiru1 iru1 osys sys
sys-uv35  blade17  pair9  oiru1 iru1 osys sys
...
sys-uv62  blade31  pair15 oiru1 iru1 osys sys
sys-uv63  blade31  pair15 oiru1 iru1 osys sys
sys-uv64  blade32  pair16 eiru2 iru2 esys sys
sys-uv65  blade32  pair16 eiru2 iru2 esys sys
...
sys-uv126 blade63  pair31 oiru3 iru3 osys sys
sys-uv127 blade63  pair31 oiru3 iru3 osys sys
sys-uv128 blade64  pair32 eiru4 iru4 esys sys
sys-uv129 blade64  pair32 eiru4 iru4 esys sys
...
sys-uv190 blade95  pair47 oiru5 iru5 osys sys
sys-uv191 blade95  pair47 oiru5 iru5 osys sys
sys-uv192 blade96  pair48 eiru6 iru6 esys sys
sys-uv193 blade96  pair48 eiru6 iru6 esys sys
...
sys-uv252 blade126 pair63 eiru7 iru7 esys sys
sys-uv253 blade126 pair63 eiru7 iru7 esys sys
sys-uv254 blade127 pair63 oiru7 iru7 osys sys
sys-uv255 blade127 pair63 oiru7 iru7 osys sys
```

4. Define the order in which Moab should check node sets for available nodes. Since the NODESETPRIORITYTYPE has a value of *FIRSTFIT*, the node sets must be ordered from smallest to largest so Moab will always choose the node set with the fewest nodes required to satisfy the job's request. This means listing all blades, blade pairs, even and odd IRUs, IRUs, even and odd system, and system, respectively.

```
moab.cfg:
NODESETLIST
blade1,blade2,blade3,…,blade127,pair1,pair2,pair3,…,pair63,eiru0,oiru0,eiru1,oiru1,
…,eiru7,oiru7,iru0,iru1,…,iru7,esys,osys,sys
```

5. Configure Moab to use the *PRIORITY* **NODEALLOCATIONPOLICY**. This allocation policy causes Moab to allocate enough nodes to fulfill a job's processor and memory requirement.

```
NODEALLOCATIONPOLICY PRIORITY
```

6. Set **NODEACCESSPOLICY** to *SINGLEJOB* to ensure that Moab will schedule large memory requests correctly and efficiently. This is necessary even when a job uses only the memory of a NUMA node.

```
NODEACCESSPOLICY SINGLEJOB
```

The policy *SINGLEJOB* tells Moab not to allow jobs to share NUMA resources (cores and memory), which for a shared-memory system is very important for fast job execution. For example, if Moab scheduled a job to use the cores of a NUMA node where memory is used by another job, both jobs would execute slowly (up to 10 times more slowly).

Job Submission

Jobs can request processors and memory using the `-l nodes=<number of cpus>` and `-l mem=<amount of memory>` syntaxes. You should not have **JOBNODEMATCHPOLICY** *EXACTNODE* configured on a NUMA system. You must use the `sharedmem` job flag on submission to force the job to run only on a sharedmem partition or cluster and to indicate that the job can span multiple nodes. For example:

```
qsub -l nodes=3,mem=640sgb,flags=sharedmem
```

Appendix H: Interfacing with Moab (APIs)

Moab provides numerous interfaces allowing it to monitor and manage most services and resources. It also possesses flexible interfaces to allow it to interact with peer services and applications as both a broker and an information service. This appendix is designed to provide a general overview and links to more detailed interface documentation.

- H.1 Moab Query and Control APIs
    - Allow external portals and services to obtain information about compute resources, workload, and usage statistics.
- H.2 Resource Management Interfaces
    - Allow Moab to monitor, schedule, and control services and resources.
- H.3 Identity and Credential Management Interfaces
    - Allow monitoring and active management of user configuration, credentials, policies, and usage information.
- H.4 Accounting and Event Interfaces
    - Allow import/export of accounting and event information to external entities.
- H.5 Job Submission and Management Interface
    - Query resource availability, submit, modify, and manage jobs, and query the status of active and completed jobs.
- H.6 Grid Services API
    - Provide and use information, data, job, and resource management services in a distributed environment.

Moab interfaces to systems providing various services and using various protocols. This appendix is designed to assist users who want to enable Moab in new environments using one of the existing interfaces. It does not cover the steps required to create a new interface.

H.1 Query and Control APIs

The Moab Cluster and Grid Suites provides a (Moab) workload manager server that supports a broad array of client services. These services can be directly accessed via Moab client commands.

H.1.1 CLI (Command Line Interface) XML API

All Moab client commands can report results in XML format to allow the information to be easily integrated into peer services, portals, databases, and other applications. To request that a client command report its output in XML, specify the `--format=xml` flag as in the following example:

```
> showq --format=xml
<Data>
<Object>queue</Object>
<cluster LocalActiveNodes="1" LocalAllocProcs="1" LocalIdleNodes="0"
LocalIdleProcs="3" LocalUpNodes="1"
  LocalUpProcs="4" RemoteActiveNodes="0" RemoteAllocProcs="0" RemoteIdleNodes="0"
RemoteIdleProcs="0"
  RemoteUpNodes="0" RemoteUpProcs="0" time="1128451812"></cluster>
<queue count="1" option="active">
<job AWDuration="11672" EEDuration="1128451812" Group="[DEFAULT]" JobID="Moab.2"
MasterHost="cw2" PAL="2"
  QOS="bug3" ReqAWDuration="54000" ReqNodes="1" ReqProcs="1" RsvStartTime="1128451812"
RunPriority="0"
  StartPriority="1" StartTime="1128451812" StatPSDed="11886.580000"
StatPSUtl="11886.580000" State="Running"
  SubmissionTime="1128451812" SuspendDuration="0" User="smith"></job>
</queue>
<queue count="1" option="eligible">
<job EEDuration="1128451812" Group="jacksond" JobID="customer.35" QOS="bug"
ReqAWDuration="3600"
  ReqProcs="1" StartPriority="1" StartTime="0" State="Idle"
SubmissionTime="1128451812" SuspendDuration="0"
  User="johnson"></job>
<queue><queue count="0" option="blocked"></queue>
</Data>
```

Common Query/Control Services

- jobs
  - query status - mdiag -j (XML details)
  - submit - msub (XML format)
  - cancel - mjobctl -c
- nodes
  - query status - mdiag -n (XML details)
  - create resource reservation - mrsvctl -c
  - destroy resource reservation - mrsvctl -r

H.2 Resource Management Interfaces

Moab can monitor, schedule, and control services and resources using multiple protocols. These protocols include the following:

- LDAP
- script/flat file
- Resource Manager Specific Interfaces - LSF, SGE, TORQUE, PBSPro, Loadleveler, and so forth

Using the resource manager interfaces, Moab can do the following:

- monitor resources (compute host, network, storage, and software license based resources)
  - load configuration, architecture, and feature information
  - load state, utilization, and workload information
  - load policy and ownership information
- manage resources
  - dynamically reconfigure and reprovision resource hardware (processors, memory, etc.)
  - dynamically reconfigure and reprovision resource software (operating system, application software, filesystem mounts, etc.)
  - dynamically reconfigure and reprovision resource security (VPN's, VLAN's, host security, etc.)
- monitor workload (batch jobs, interactive jobs, persistent services, dynamic services, distributed services)
  - load state, resource requirement, and required environment information
  - load user, group, and credential information
  - load utilization, resource allocation, and policy information
- manage workload
  - migrate jobs from one resource to another (intra-cluster and inter-cluster)
  - modify jobs for translation and optimization purposes
  - suspend, resume, checkpoint, restart, and cancel jobs
- query cluster policies and configuration

H.3 Identity and Credential Management Interfaces

Moab's identity and credential management interfaces allow Moab to exchange credential and user configuration, access, policy, and usage information.

- Identity Manager
- Allocation Manager
- Moab Workload Manager for Grids

H.4 Accounting Interfaces

Moab accounting interfaces allow Moab to export local utilization statistics, events, and accounting information to site specific scripts.

- Accounting Interface

H.6 Job Submission and Management Interface

Moab provides interfaces to enable the following services:

- Resource Availability Query
    - Determine quantity, state, and configuration of configured resources (idle, busy, and down nodes)
    - Determine quantity and configuration of all available resources (idle nodes)
    - Determine resources available subject now and in the future for potential job
    - Determine best target cluster destination for potential job
    - Determine largest/longest job which could start immediately
    - Determine estimated start time for potential job
    - Determine earliest guaranteed start time for potential job
- Reserve Resources
    - Reserve specific resources for desired time frame
- Submit Job (XML format)
    - Submit job to specific cluster
    - Submit job to global job queue
- Manage Job
    - Hold job
    - Adjust job priority
    - Modify job executable, args, data requirements, job dependencies, duration, hostcount, or other attributes
    - Suspend/resume job
    - Checkpoint/requeue job
    - Cancel job
    - Migrate job
    - Adjust job quality of service (QoS)
- Query Job
    - Determine job state, utilization, or output results for idle, active, or completed job
    - Determine estimated start time
    - Determine guaranteed start time

H.7 Grid Interfaces

Moab provides interfaces to allow interaction with various grid brokers and services. These interfaces allow Moab to provide services as well as utilize services.

Services Utilized

- Information Services (import and utilize information service data in making scheduling decisions)
- Job Migration
- Data Migration
- Credential Mapping
- Security and Delegation

See Moab Workload Manager for Grids for more information on utilized services.

Services Provided

- Information Services (provide resource, workload, and credential information)
- Job Migration
- Data Migration
- Credential Mapping

See Moab Workload Manager Grid Basics for more information on provided services.

Appendix I: Considerations for Large Clusters

There are several key considerations in getting a batch system to scale.

I.1 Resource Manager Scaling

Proper Resource Manager Configuration

- TORQUE
  - General Scaling Overview

- OpenPBS/PBSPro
  - Manage Direct Node Communication with NODEPOLLFREQUENCY

I.2 Handling Large Numbers of Jobs

Set a minimum RMPOLLINTERVAL

With event driven resource managers like TORQUE, each time a job is submitted the resource manager notifies the scheduler. In an attempt to minimize response time, the scheduler starts a new scheduling cycle to determine if the newly submitted job can run. In systems with large numbers of jobs submitted at once, this might not result in the desired behavior for two reasons. First, by scheduling at every job submission Moab schedules newly submitted jobs onto available resources in a first come, first served basis rather than evaluating the entire group of new jobs at once and optimizing the placement accordingly. Second, by launching a scheduling iteration for every job submitted, Moab places a heavy load on the resource manager. For example, if a user were to submit 1000 new jobs simultaneously, for each job submitted, the resource manager contacts the scheduler, the scheduler starts a new iteration, and in this iteration, the scheduler contacts the resource manager requesting updated information on all jobs and resources available.

Setting a minimum **RMPOLLINTERVAL** causes the scheduler to not process jobs as quickly as they are submitted, but rather to wait a minimum amount of time to allow more jobs be submitted and to process these new jobs in groups.

```
RMPOLLINTERVAL 30,60
```

If the system is busy, schedule every 30 seconds. If it is not busy, schedule every 60 seconds.

Reduce command processing time

If your system's scheduling cycle regularly takes longer than the CLIENTTIMEOUT value, you can configure Moab to fork a copy of itself that will respond to certain information-only client commands (checkjob, showbf, , and showstart). This enables you to run intense diagnostic commands while Moab is in the middle of its scheduling process.

In both the client and server configuration, when you set **UIMANAGEMENTPOLICY** *FORK* on a submithost, Moab forks a copy of itself that will listen for client commands on a separate port, which you must configure with **CLIENTUIPORT**. This forked process responds to checkjob, showbf, showres, and showstart until the main scheduling cycle has finished. After that, it is killed and the normal process resumes responding to client commands. Moab prints a disclaimer at the top of each command that was populated by the forked process stating that the information may be a few seconds stale.

Example 3-179: Sample configuration

```
UIMANAGEMENTPOLICY      FORK
CLIENTUIPORT            41560
```

Moab forks a copy of itself on port 41560, where it will watch for checkjob, showbf, showres, and showstart commands until the main scheduling process completes.

Example 3-180: Sample command output

```
$ checkjob 34

--------------------------------------------------------------------
NOTE: The following information has been cached by the remote server
and may be slightly out of date.
--------------------------------------------------------------------

job 34

State: Idle
Creds: user:wightman group:company class:batch
WallTime: 00:00:00 of 00:01:00
SubmitTime: Thu May 22 14:17:06
(Time Queued Total: 00:00:18 Eligible: 00:00:18)

TemplateSets: DEFAULT
Total Requested Tasks: 1

Req[0] TaskCount: 1 Partition: ALL

SystemID: scale
SystemJID: 34

IWD: $HOME/test/scale
SubmitDir: $HOME/test/scale
Executable: sleep 60
```

Limited Job Checkpointing

Use the **LIMITEDJOBCP** on page 966 parameter. By default, Moab will checkpoint information about every job it reads from its resource managers. When a cluster routinely runs more than 15000 jobs, they may see some speed-ups by limiting which jobs are checkpointed. When **LIMITEDJOBCP** is set to *TRUE*, Moab will only checkpoint jobs that have a hold, a system priority, jobs that have had their QoS modified, and a few other limited attributes. Some minimal statistical information is lost for jobs that are not checkpointed.

Minimize Job Processing Time

Use the **ENABLEHIGHTHROUGHPUT** on page 929 parameter. By default, Moab processes all job attributes, filters, remap classes, job arrays, and other information when a job is submitted. This requires full access to the Moab configuration and significantly increases the processing time Moab needs when jobs are submitted. By setting **ENABLEHIGHTHROUGHPUT** to *TRUE*, Moab stores the job information in an internal queue and returns the job ID immediately. The internal queue is processed when Moab begins its next scheduling iteration. This enables Moab to process hundreds of jobs per second rather than 20-30 per second. Because the jobs are processed in a separate queue after the job has been returned, it is recommended that MAILPROGRAM be configured. Moab will send an email to the user if a job is rejected.

Because the job is not fully processed, some attributes may change after the job has been submitted. For example, when a job class is remapped, the new class is not reflected until Moab begins its next scheduling iteration. Additionally, job arrays are not instantiated until Moab begins its next scheduling cycle.

If **ENABLEHIGHTHROUGHPUT** on page 929 is *TRUE*, you must set **NODEALLOCATIONPOLICY** on page 978 to *FIRSTAVAILABLE*.

Load all Non-Completed Jobs at Startup

Use the **LOADALLJOBCP** on page 967 parameter. By default, Moab loads non-complete jobs for active resource managers only. By setting **LOADALLJOBCP** to *TRUE*, Moab will load all non-complete jobs from all checkpoint files at startup, regardless of whether their corresponding resource manager is active.

Reducing Job Start Time

Use the ASYNCSTART parameter. By default, Moab will launch one job at a time and verify that each job successfully started before launching a subsequent job. For organizations with large numbers of very short jobs (less than 2 minutes in duration), the delay associated with confirming successful job start can lead to productivity losses. If tens or hundreds of jobs must be started per minute, and especially if the workload is composed primarily of serial jobs, then the resource manager **ASYNCSTART** flag may be set. When set, Moab will launch jobs optimistically and confirm success or failure of the job start on the subsequent scheduling iteration. Also consider adding the *ASYNCDELETE* flag if users frequently cancel jobs.

Reducing Job Reservation Creation Time

Use the **RMCFG** on page 1014 **JOBRSVRECREATE** on page 598 attribute. By default, Moab destroys and re-creates job reservations each time a resource manager updates any aspect of a job. Historically, this stems from the fact that certain resource managers would inadvertently or intentionally migrate job tasks from originally requested nodes to other nodes. To maintain synchronization, Moab would re-create reservations each iteration thus incorporating these changes. On most modern resource managers, these changes never occur, but the effort required to handle this case grows with the size of the cluster and the size of the queue. Consequently, on very large systems with thousands of nodes and thousands of jobs, a noticeable delay is present. By setting **JOBRSVRECREATE** to *FALSE* on resource managers that do not exhibit this behavior, significant time savings per iteration can be obtained.

Optimizing Backfill Time

Use the OPTIMIZEDBACKFILL flag. Speeds up backfill when a system reservation is in use.

Constraining Moab Logging - LOGLEVEL

Use the **LOGLEVEL** on page 969 parameter. When running on large systems, setting **LOGLEVEL** to 0 or 1 is normal and recommended. Only increase **LOGLEVEL** above 0 or 1 if you have been instructed to do so by Moab support.

Preemption

When preemption is enabled Moab can take considerably more time scheduling jobs for every scheduling iteration. Preemption increases the number of options available to Moab and therefore takes more time for Moab to optimally place jobs. If you are running a large cluster or have more than the usual amount of jobs (>10000), consider disabling preemption. If disabling preemption is not possible, consider limiting its scope to only a small subset of jobs (as both preemptors and preemptees).

Handling Transient Resource Manager Failures

Use the RMCFG MAXITERATIONFAILURECOUNT on page 600 attribute.

Constrain the number of jobs preempted per iteration

Use the JOBMAXPREEMPTPERITERATION parameter.

> For very large job count systems, configuration options controlling the maximum supported limits may need to be adjusted including the maximum number of reservations and the maximum number of supported evaluation ranges.

Scheduler settings

If using Moab, there are a number of parameters which can be set on the scheduler which may improve TORQUE performance. In an environment containing a large number of short-running jobs, the **JOBAGGREGATIONTIME** parameter can be set to reduce the number of workload and resource queries performed by the scheduler when an event based interface is enabled. Setting **JOBAGGREGATIONTIME** instructs the scheduler to ignore events coming from the resource manager and to scheduling at regular intervals, rather than around resource manager events. If the pbs_server daemon is heavily loaded and PBS API timeout errors (i.e. "Premature end of message") are reported within the scheduler, the **TIMEOUT** attribute of the **RMCFG** parameter may be set with a value of between 30 and 90 seconds.

I.3 Handling Large Numbers of Nodes

For very large clusters (>= 10,000 processors) default scheduling behavior may not scale as desired. To address this, the following parameters should be considered:

Parameter	Recommended Settings
RMPOLLINTERVAL	In large node environments with large and long jobs, scheduling overhead can be minimized by increasing **RMPOLLINTERVAL** above its default setting. If an event-driven resource management interface is available, values of two minutes or higher may be used. Scheduling overhead can be determined by looking at the scheduling load reported by mdiag -S.
LIMITEDNODECP	Startup/shutdown time can be minimized by disabling full node state checkpointing that includes some statistics covering node availability.
SCHEDCFG FLAGS=" FASTRSVSTARTUP on page 1369	When you have reservations on a large number of nodes, it can take Moab a long time to recreate them on startup. Setting the *FASTRSVSTARTUP* scheduler flag greatly reduces startup time.

* For clusters where the number of nodes or processors exceeds 50,000, the maximum stack size for the shell in which Moab is started may need to be increased (as Moab may crash if the stack size is too small). On most Unix/Linux based systems, the command ulimit -s unlimited may be used to increase the stack size limit before starting Moab. This may be placed in your Moab startup script.

See Appendix D for further information on default and supported object limits.

Avoid adding large numbers of NODECFG lines in the `moab.cfg` or `moab.d/*.cfg` files to keep the Moab boot time low.

For example, adding a configuration line to define features for each node in a large cluster (such as `NODECFG[x] Features+=green,purple`) can greatly increase the Moab boot time. If Moab processes 15 node configuration lines per second for a 50,000-node system, it could add approximately 55 minutes of node configuration processing to the Moab boot time.

In this case, it is better to define the node features in the resource manager configuration.

I.4 Handling Large Jobs

For large jobs, additional parameters beyond those specified for large node systems may be required. These include settings for the maximum number of tasks per job, and the maximum number of nodes per job.

I.5 Handling Large SMP Systems

For large-way SMP systems (> 512 processors/node) Moab defaults may need adjustment.

Parameter	Recommended Settings
MAXRSVPERNODE	By default, Moab does not expect more than 48 jobs per node to be running or have future reservations. Increasing this parameter to a value larger than the expected maximum number of jobs per node is advised.

I.6 Server Sizing

See Hardware and Software Requirements for recommendations.

Related topics

- Appendix D: Adjusting Default Limits

Appendix J: Configuring Moab as a Service

Scripts that follow can be used to start up Moab services automatically upon a reboot. To enable a service script, copy the script to `/etc/rc.d/init.d/S97moab`, edit the file to make needed localization changes (adjust binary paths, execution user, etc), and add links to the `rc3.d`and `rc5.d` directories as in the example that follows:

```
> cp mwm.service /etc/rc.d/init.d/S97moab
> vi /etc/rc.d/init.d/S97moab
    (make needed localizations)
> ln -s /etc/rc.d/init.d/S97moab /etc/rc.d/rc3.d
> ln -s /etc/rc.d/init.d/S97moab /etc/rc.d/rc5.d
```

J.1 Moab Workload Manager Service Scripts

- Moab Workload Manager Script
- Moab Workload Manager + TORQUE Script

J.2 Moab Grid Scheduler Service Script

- sample script

Appendix K: Migrating from 3.2

Overview

This guide is intended to help facilitate migrating from Maui to Moab. If you do not have Moab yet, you can download a free evaluation version. At a high level, migrating from Maui 3.2 to Moab involves minimal effort. In fact, Moab fully supports all Maui parameters and commands. Migration can consist of nothing more than renaming `maui.cfg` to `moab.cfg` and launching Moab using the moab command. With this migration, the biggest single issue is becoming aware of all the new facilities and capabilities available within Moab. Beyond this, migration consists of a few minor issues that may require attention such as some statistics and priorities.

Another approach of migrating from Maui to Moab is to configure Moab in Monitor mode and run it beside Maui. Maui will continue to perform the scheduling and control workload. Moab will simply monitor the cluster environment using the policies configured in `moab.cfg`. Moab will not have the ability to affect workload, providing a safe and risk-free environment to evaluate Moab without affecting your production environment. You can also have Moab capture resource and workload trace files and allow Moab to simulate what it would have done if it controlled workload. When you feel comfortable with and want to run Moab live on your cluster, all you need to do is change the mode to NORMAL, stop Maui, and restart Moab. Current jobs will remain running and Moab will take over control of scheduling.

As with any migration, we suggest that you back up important files such as the following: `maui.cfg`, `maui.log` and `maui.ck`.

View the Flash demo of migrating from Maui to Moab.

Migrating from Maui to Moab

1. Install Moab Workload Manager. (Installation Instructions)
2. Copy your `maui.cfg` file to the `MOABHOMEDIR/etc` (`/opt/moab/etc`) and rename it `moab.cfg`.
3. Stop Maui.
4. Start Moab.

5. If Applicable: Re-apply those configurations found in the Statistics and Checkpointing section that need adjustment after migration as well as any parameters in `moab.cfg` that point to a Maui file like `maui.log`.

Running Maui and Moab Side-By-Side

1. Install Moab Workload Manager on your cluster. (Installation steps will differ slightly from a typical installation.)
   a. Run ./configure.
   b. Run make.
   c. You will need to set your *MOABHOMEDIR* environment variable to the location where you built Moab by typing `export MOABHOMDIR=[make directory]`.
2. To have Moab use all the same policies as Maui, copy `maui.cfg` to the MOAB`HOMEDIR/etc` and rename it `moab.cfg`.
   - You can also start your `moab.cfg` file from scratch. Just use the `moab.cfg` already in the `MOABHOMEDIR/etc`.
3. Make sure that the port in `moab.cfg` is different than the port used in `maui.cfg`.
4. In the `moab.cfg` file, add the parameter, `SERVERMODE=MONITOR`.
   - If you used the `moab.cfg` from scratch, on the **SCHEDCFG** line add `MODE=MONITOR`.
5. You will need to either put the Moab commands in your environment path (located in `MOABHOMEDIR/bin`) or run the commands from their location if you still want to use the Maui commands in your environment path.
6. Run Moab Workload Manager using the moab command located in `MOABHOMEDIR/bin`.

Other Notes

The following are minor differences between Maui and Moab and changes you may need to make:

File Naming

Moab uses slightly different naming than Maui. The following table displays these changes:

File	Maui	Moab
executable	maui	moab
logs	`maui.log`	`moab.log`
configuration file	`maui.cfg`	`moab.cfg`

Statistics and Checkpointing

Moab supports Maui version 3.2 or higher workload traces (statistics) allowing it to process historical statistics based on these traces as well as generate simulations based on them. No changes are required to use these statistics. See the Simulation Configuration documentation for more information on trace files. You can also view a flash demonstration of the simulation mode.

Moab does not support the Maui 3.2 checkpointing format. Because of this, state information checkpointed under Maui will not be available at the time of the migration. The loss of this information will have the following impact:

- Admin reservations, if any, will need to be re-created.
- Processed credential and scheduler statistics (displayed by showstats) will be lost.
- Admin job system priority configured by the setspri command and QoS assignments configured by the setqos command, if any, will be lost.

Verify Configuration File Compatibility

The command mdiag -C will perform diagnostics on your new configuration file and may prove helpful in identifying any issues.

Environment Variables

Scheduler environment variables are supported under Moab with obvious naming changes. Sample environment variables follow:

Maui	Moab
MAUIHOMEDIR	MOABHOMEDIR
MAUIDEBUG	MOABDEBUG
MAUICRASHVARIBALE	MOABCRASHVARIABLE
MAUIENABLELOGBUFFERING	MOABENABLELOGBUFFERING
MAUIRECOVERYACTION	MOABRECOVERYACTION
MAUI-COMMANDS-PATH	MOAB-COMMANDS-PATH
MAUIENABLELOGBUFFERING	MOABENABLELOGBUFFERING

Appendix R: Node Allocation Plug-in Developer Kit

R.1 Overview

Each time Moab schedules a job, it must choose the nodes on which the job will run. Moab uses the Node Allocation policy to select the available nodes to be used. Because there are so many different systems and cluster topologies, you now have the ability to create and use a node allocation plugin for allocating nodes based on your cluster's interconnect topology.

The plugin policy allows you to write your own algorithm to choose which nodes will be used. This algorithm is contained in a shared library that Moab loads at run time.

To obtain the Plug-in Developer Kit (PDK) with the header file and example code, contact your sales representative.

R.1.1 Writing the plugin

A plugin is a shared library that has specific functions and variables that will be called directly from Moab. The plugin conforms to a C language API. The API is specified through an include file: `moab-plugin.h`. This file must be included in the plugin code. The include file provides function definitions, structures and variables that will be used when communicating with Moab.

When you write the plugin, you need to ensure that the plugin code is robust. If the plugin crashes, Moab will crash. You will need to handle your own memory appropriately. If the plugin has memory leaks, Moab will have similar issues. If you want to maintain logs, the plugin will need to be responsible for its own logging.

R.1.1.1 API and Data Structures

The Application Programmer Interface (API) for the Moab Node Allocation Plugin consists of three data items and three entry points that must be supplied to Moab by the plugin.

Plugin Supplied Data	Description
const char *PLUGIN_NAME = "Node Allocation plugin 1.1";	This character pointer is used by Moab when logging information regarding the operation of the plugin.
const char *PLUGIN_TYPE = PLUGIN_TYPE_ NAME_ NODEALLOCATION;	This character pointer is used by Moab to verify the type of plugin. The value of this data is supplied by the `moab-plugin.h` source file. The plugin must set this as shown so that Moab does not attempt to use a plugin incorrectly. Moab uses this to determine whether the plugin API type is correct and to allow Moab to correctly communicate with the plugin.
const char *PLUGIN_VERSION = PLUGIN_API_ VERSION;	This character pointer is used by Moab to verify the API version number. The value of this data is supplied by the `moab-plugin.h` source file. The plugin must set this as shown so that the correct version of the `moab-plugin.h` is supplied to Moab. Moab uses this to determine whether the API version is correct and to allow Moab to correctly communicate with the plugin.

Load Time API	Description
initialize()	`int initialize(const char *name, void **data_handle)` The plugin must supply an initialize() entry point. This entry point is called for each use instance of the plugin. For example, if the plugin is used on two different partitions, the initialize() entry point will be called once for each partition. • Name — The name is the unique identifier which is used to distinguish multiple instances of the plugin and for logging. When configured globally, the name “ALL” will be given. • Data handle — The data_handle points to a location where the plugin should store a pointer to any internal data needed by the plugin between calls to the API. The actual format and structure of the data is up to the plugin. Moab will supply this pointer back to the plugin each time a plugin entry point is called. This data can provide context for the plugin usage instance.
Return codes	The initialize() entry point should return one of two return statuses as defined in `moab-plugin.h`: `#define PLUGIN_RC_SUCCESS 0` `#define PLUGIN_RC_FAILURE 1`

Load Time API	Description
Gathering node info	The initialize() entry point must gather any information about system nodes, their topology, interconnection, and configuration that it needs to make correct node allocations. Since Moab does not know what information the plugin may need, the plugin must gather this information itself.
Memory considerations	The plugin may allocate memory for temporary or persistent data as needed, but *must* de-allocate or return the memory when finished. Not returning memory can result in memory leaks and unstable operation on the part of Moab.
Multiple access	A given loaded plugin can be used by more than one partition. This means that the plugin must maintain its internal data in such a way that calls to the plugin for the separate partitions do not conflict. It is recommended that internal data be allocated and a pointer to the data be kept in the data_handle described above as opposed to using global or static variables. Any global or static data will be shared between possible multiple instances of the plugin.

<table>
<tr><th>Runtime API</th><th>Description</th></tr>
<tr><td>node_allocate()</td><td>
<pre>int node_allocate (
  void   *data_handle,
  const char   *job_name,
  int   container_count,
  nalloc_container_t   container[])</pre>
The plugin must provide a node_allocate() entry point. This entry point is called each time Moab needs to determine where (on what nodes) a job will eventually run. Note that this entry point can be called many times before the job is actually scheduled to run.
Data structures — Moab uses C data structures to pass information and lists of nodes to the plugin and receive them back from the plugin. See <code>moab-plugin.h</code> for the definitions of these structures and for information on how they relate to one another.
</td></tr>
</table>

<table>
<tr><th>Runtime API</th><th>Description</th></tr>
<tr><td>Operations</td><td>
A node allocation request consists of one or more requirements. Each of these requirements is provided within a “container” structure. The container has information regarding the requirement to be met, the count and list of all nodes that are available to meet the requirement and a place to return the list of nodes that the plugin has chosen to use for the job.
<table>
<tr><th>Command</th><th>Moab Job Task Count</th><th>Job Node Count</th><th>Job Tasks Per Node</th><th>Node CFG Procs</th><th>Node AVL Procs</th><th>Plugin Node Mapped TC</th><th>requirement ->taskcount</th><th>return_node_count</th></tr>
<tr><td colspan="9">Non-ExactNode</td></tr>
<tr><td>-l nodes=12</td><td>12</td><td>0</td><td>0</td><td>8</td><td>8</td><td>8</td><td>12</td><td>2</td></tr>
<tr><td>-l nodes=12:ppn=2</td><td>24</td><td>0</td><td>2</td><td>8</td><td>8</td><td>8</td><td>24</td><td>3</td></tr>
<tr><td colspan="9">ExactNode</td></tr>
<tr><td>-l nodes=4</td><td>4</td><td>4</td><td>0</td><td>8</td><td>8</td><td>1</td><td>4</td><td>4</td></tr>
<tr><td>-l nodes=4:ppn=2</td><td>8</td><td>4</td><td>2</td><td>8</td><td>8</td><td>2</td><td>8</td><td>4</td></tr>
<tr><td>-l nodes=12</td><td>12</td><td>0</td><td>0</td><td>8</td><td>6</td><td>6</td><td>12</td><td>2</td></tr>
</table>
The duty of the plugin is to use the information that it has previously gathered (during the initialization) to select from the available nodes those that will best fulfill the requirements.
The basic algorithm is to consume all the taskcount and memory on each node until the consumed task count is greater than or equal to the container's task_count and memory requirements.
A job's taskcount is calculated differently based on the JOBNODEMATCHPOLICY parameter. By default, it isn't defined and <code>-l nodes=#</code> actually requests the number of tasks without respect to the number of nodes. In this case, the plugin should consume all the tasks of each chosen node until the taskcount is greater and/or equal to the container's taskcount requirement. The plugin is for node allocation and not task placement.
</td></tr>
</table>

Runtime API	Description
	When the JOBNODEMATCHPOLICY EXACTNODE is configured, then `-l nodes=#` means the job wants # of nodes with 1 task per node. In this case, the nodes passed to the plugin will have a taskcount that is mapped down to what the job can only use on that node. Each node's taskcount should be consumed on each node until the summed amount is equal to the container's requirement taskcount requirement. The following table shows how commands are interpreted by Moab and translated to the plugin and what is expected of the plugin.
Errors and return codes	The plugin may internally log any errors encountered and must return a success or error status as defined in `moab-plugin.h`: `#define    PLUGIN_RC_SUCCESS   0` `#define    PLUGIN_RC_FAILURE   1`
Multiple access safe	The node_allocate() entry point must support multiple access as described above.

Unload Time API	Description
finish()	`void finish(void *data_handle)` The plugin must supply a finish() entry point. This entry point is called when Moab is preparing to disable and/or unload an instance of the plugin.
Memory/resource cleanup	The plugin must de-allocate and free up any resources acquired either during the initialize() entry point or during any calls to the node_allocate() entry point. When the last entry point returns, there should be no allocated memory or other resources still in use by the plugin instance.
Multiple access safe	The finish() entry point must support multiple access as described above.

R.1.2 Moab configuration

The actual loading of a plugin is accomplished by specifying the plugin in the Moab configuration file, `moab.cfg`.

R.1.2.1 Moab.cfg

We recommend that you store all Moab plugins in the `$MOABHOMEDIR/lib` directory (e.g., `/opt/moab/lib`) as shared libraries (`*.so`). The name of the actual plugin shared library file is up to the plugin developer, which means you must give the correct name in the moab.cfg file to form the absolute plugin filename.

If a plug-in's specified shared library filename starts with a forward slash (/), it is an absolute file path name and Moab simply uses it without alteration. For example, if a plugin's specified shared library filename is `/opt/moab/plugins/plugin.so`, Moab will use it as the absolute plugin file path name.

If a plugin's specified shared library filename does not start with a forward slash (/), it is a plugin name and Moab forms the plugin's absolute path name by concatenating the Moab home directory, "/lib/lib", the specified plugin name, and ".so" to obtain the absolute path name. For example, if the *$MOABHOMEDIR* environment variable contains `/opt/moab` and the plugin name is `plugin`, Moab will create `/opt/moab/lib/libplugin.so` and use it as the absolute plugin file path name.

R.1.2.2 Syntax rules

In order for Moab to use a plugin for the Node Allocation policy, instead of a built-in Moab policy, you must configure the policy in the moab.cfg file with the value "PLUGIN:" followed by the plugin's shared library file name. The examples below assume the environment variable *$MOABHOMEDIR* has a value of `/opt/moab`. Note the use of relative and absolute plugin shared library file path names in the parameter value and how they affect Moab's construction of the full path name.

Partition	Plug-in Name	moab.cfg Parameter	Moab-derived Full Path Name
global	`plugin.so`	NODEALLOCATIONPOLICY *PLUGIN:plugin.so*	`/opt/moab/lib/libplugin.so`
global	`/usr/local/plugins/plugin.so`	NODEALLOCATIONPOLICY *PLUGIN:/usr/local/plugins/plugin.so*	`/usr/local/plugins/plugin.so`
abc	`plugin.so`	PARCFG[*abc*] NODEALLOCATIONPOLICY =*PLUGIN:plugin.so*	`/opt/moab/lib/libplugin.so`
xyz	`/usr/local/plugins/plugin.so`	PARCFG[*xyz*] NODEALLOCATIONPOLICY= *PLUGIN:/usr/local/plugins/plugin.so*	`/usr/local/plugins/plugin.so`

R.1.2.3 Troubleshooting

There are several commands that can be used to confirm that the Plugin Node Allocation Policy was loaded properly.

mschedctl -l

mschedctl -l is used to print out Moab's in memory configurations. If the plugin policy, with its full path, doesn't show for the configured partition then Moab failed to load the partition. Note that when the **NODEALLOCATIONPOLICY** is configured globally, it is configured on the "ALL" partition.

```
$ mschedctl -l -v|grep ^NODEALLOCATIONPOLICY
NODEALLOCATIONPOLICY[ALL] PLUGIN:/opt/moab/lib/libfirstavailable.so
NODEALLOCATIONPOLICY[a] PLUGIN:/opt/moab/lib/liblastavailable.so
NODEALLOCATIONPOLICY[b] CONTIGUOUS
NODEALLOCATIONPOLICY[c] PLUGIN:/opt/moab/lib/libfirstavailable.so
NODEALLOCATIONPOLICY[d] [NONE]
```

mdiag -C

mdiag -C is used to validate the moab.cfg configuration. With a plugin node allocation policy, Moab will validate that it can successfully load the plugin and that all of the required symbols are present.

```
$ mdiag -C
...
INFO: line #35 is valid: 'NODEALLOCATIONPOLICY PLUGIN:firstavailable'
INFO: line #36 is valid: 'PARCFG[a]NODEALLOCATIONPOLICY=PLUGIN:lastavailable'
INFO: line #37 is valid: 'PARCFG[b]NODEALLOCATIONPOLICY=CONTIGUOUS'
INFO: line #38 is valid: 'PARCFG[d]NODEALLOCATIONPOLICY=PLUGIN:firstavailable'
```

Appendix S: Scalable Systems Software Specification

Scalable Systems Software Job Object Specification

SSS Job Object Specification
Draft Release Version 3.1.0
26 April 2011

Scott Jackson, PNNLStringDavid Jackson, Ames Lab
Brett Bode, Ames Lab

Status of This Memo

This document describes the job object to be used by Scalable Systems Software compliant components. It is envisioned for this specification to be used in conjunction with the SSSRMAP protocol with the job object passed in the Data field of Requests and Responses. Queries can be issued to a job-cognizant component in the form of modified XPATH expressions to the Get field to extract specific information from the job object as described in the SSSRMAP protocol.

Abstract

This document describes the syntax and structure of the SSS job object. A job model is described that is flexible enough to support the specification of very simple jobs as well as jobs with elaborate and complex specification requirements in a way that avoids complex structures and syntax when it is not needed. The basic assumption is that a solitary job specification should be usable for all phases of the job lifecycle and can be used at submission, queuing, staging, reservations, quotations, execution, charging, accounting, etc. This job specification provides support for multi-step jobs, as well as jobs with disparate task descriptions. It accounts for operational requirements in a grid or meta-scheduled environment where the job is executed by multiple hosts in different administrative domains that support different resource management systems.

Table of Contents

1.0 Introduction

This specification proposes a standard XML representation for a job object for use by the various components in the SSS Resource Management System. This object will be used in multiple contexts and by multiple components. It is anticipated that this object will be passed via the Data Element of SSSRMAP Requests and Responses.

1.1 Goals

There are several goals motivating the design of this representation.

The representation needs to be inherently flexible. We recognize we will not be able to exhaustively include the ever-changing job properties and capabilities that constantly arise.

The representation should use the same job object at all stages of that job's lifecycle. This object will be used at job submission, queuing, scheduling, charging and accounting, hence it may need to distinguish between requested and delivered properties.

The design must account for the properties and structure required to function in a meta or grid environment. It needs to include the capability to support local mapping of properties, global namespaces, etc.

The equivalent of multi-step jobs must be supported. Each step (job) can have multiple logical task descriptions.

Many potential users of the specification will not be prepared to implement the complex portions or fine-granularity that others need. There needs to be a way to allow the more complicated structure to be added as needed while leaving more straightforward cases simple.

There needs to be guidance for how to understand a given job object when higher order features are not supported by an implementation, and which parts are required, recommended and optional for implementers to implement.

It needs to support composite resources.

It should include the ability to specify preferences or fuzzy requirements.

1.2 Non-Goals

Namespace considerations and naming conventions for most property values are outside of the scope of this document.

1.3 Examples

Example 3-181: Very Simple Example

This example shows a simple job object that captures the requirements of a simple job.

```
<Job>
  <Id>PBS.1234.0</Id>
  <State>Idle</State>
  <User>scottmo</User>
  <Executable>/bin/hostname</Executable>
  <Processors>16</Processors>
  <Duration>3600</Duration>
</Job>
```

Example 3-182: Moderate Example

This example shows a moderately complex job object that uses features such as required versus delivered properties.

```
<Job>
  <Id>PBS.1234.0</Id>
  <Name>Heavy Water</Name>
  <Project>nwchemdev</Project>
  <User>peterk</User>
  <Application>NWChem</Application>
  <Executable>/usr/local/nwchem/bin/nwchem</Executable>
  <Arguments>-input basis.in</Arguments>
  <InitialWorkingDirectory>/home/peterk</InitialWorkingDirectory>
  <Machine>Colony</Machine>
  <QualityOfService>BottomFeeder</QualityOfService>
  <Queue>batch_normal</Queue>
  <State>Completed</State>
  <StartTime>1051557713</StartTime>
  <EndTime>1051558868</EndTime>
  <Charge>25410</Charge>
  <Requested>
    <Processors op=”GE”>12</Processors>
    <Memory op=”GE” units=”GB”>2</Memory>
    <Duration>3600</Duration>
  </Requested>
  <Delivered>
    <Processors>16</Processors>
    <Memory metric=”Average” units=”GB”>1.89</Memory>
    <Duration>1155</Duration>
  </Delivered>
  <Environment>
    <Variable name=”PATH”>/usr/bin:/home/peterk</Variable>
  </Environment>
</Job>
```

Example 3-183: Elaborate Example

This example uses a job group to encapsulate a multi-step job. It shows this protocol’s ability to characterize complex job processing capabilities. A component that processes this message is free to retain only that part of the information that it requires. Superfluous information can be ignored by the component or filtered out (by XSLT for example).

```
<JobGroup>
  <Id>workflow1</Id>
  <State>Active</State>
  <Name>ShuttleTakeoff</Name>
  <JobDefaults>
    <StagedTime>1051557859</StagedTime>
    <SubmitHost>asteroid.lbl.gov</SubmitHost>
    <SubmitTime>1051556734</SubmitTime>
    <Project>GrandChallenge18</Project>
    <GlobalUser>C=US,O=LBNL,CN=Keith Jackson</GlobalUser>
    <User>keith</User>
    <Environment>
      <Variable name="LD_LIBRARY_PATH">/usr/lib</Variable>
      <Variable name="PATH">/usr/bin:~/bin:</Variable>
    <Environment>
  </JobDefaults>
  <Job>
    <Id>fr15n05.1234.0</Id>
    <Name>Launch Vector Initialization</Name>
    <Executable>/usr/local/gridphys/bin/lvcalc</Executable>
    <Queue>batch</Queue>
    <State>Completed</State>
    <Machine>SMP2.emsl.pnl.gov</Machine>
    <StartTime>1051557713</StartTime>
    <EndTime>1051558868</EndTime>
    <Quote>http://www.pnl.gov/SMP2#654321</Quote>
    <Charge units="USD">12.75</Charge>
    <Requested>
      <Duration>3600</Duration>
      <Processors>2</Processors>
      <Memory>1024</Memory>
    </Requested>
    <Delivered>
      <Duration>1155</Duration>
      <Processors consumptionRate="0.78">2</Processors>
      <Memory metric="Max">975</Memory>
    </Delivered>
    <TaskGroup>
      <TaskCount>2</TaskCount>
      <TaskDistribution type="TasksPerNode">1</TaskDistribution>
      <Task>
        <Node>node1</Node>
        <Process>99353</Process>
      </Task>
      <Task>
        <Node>node12</Node>
        <Process>80209</Process>
      </Task>
    </TaskGroup>
  </Job>
  <Job>
    <Id>fr15n05.1234.1</Id>
    <Name>3-Phase Ascension</Name>
    <Queue>batch_normal</Queue>
    <State>Idle</State>
    <Machine>Colony.emsl.pnl.gov</Machine>
    <Priority>1032847</Priority>
    <Hold>System</Hold>
    <StatusMessage>Insufficient funds to start job</StatusMessage>
    <Requested>
```

```
      <Duration>43200</Duration>
    </Requested>
    <TaskGroup>
      <TaskCount>1</TaskCount>
      <Name>Master</Name>
      <Executable>/usr/local/bin/stage-coordinator</Executable>
      <Memory>2048<Memory>
      <Resource name="License" type="ESSL2">1</Resource>
      <Feature>Jumbo-Frame</Feature>
    </TaskGroup>
    <TaskGroup>
      <Name>Slave</Name>
      <TaskDistribution type="Rule">RoundRobin</TaskDistribution>
      <Executable>/usr/local/bin/stage-slave</Executable>
      <NodeCount>4</NodeCount>
      <Requested>
        <Processors group="-1">12</Processors>
        <Processors conj="Or" group="1">16</Processors>
        <Memory>512</Memory>
        <Node aggregation="Pattern">fr15n.*</Node>
      </Requested>
    </TaskGroup>
  </Job>
</JobGroup>
```

2.0 Conventions Used in This Document

2.1 Keywords

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC2119.

2.2 Table Column Interpretations

The columns of the property tables in this document have the following meanings:

Element Name	Name of the XML element (xsd;element) see [DATATYPES]
Type	Data type defined by xsd (XML Schema Definition)as: • String — xsd:string (a finite length sequence of printable characters) • Integer — xsd:integer (a signed finite length sequence of decimal digits) • Float — xsd:float (single-precision 32-bit floating point) • Boolean — xsd:boolean (consists of the literals “true” or “false”) • DateTime — xsd:int (a 32-bit unsigned long in GMT seconds since the EPOCH) • Duration — xsd:int (a 32-bit unsigned long measured in seconds)
Description	Brief description of the meaning of the property

Element Name	Name of the XML element (xsd;element) see [DATATYPES]
Appearance	An indication of whether the given property must appear in the parent element. It assumes the following meanings: • MUST — This property is REQUIRED when the parent is specified • SHOULD — This property is RECOMMENDED when the parent is specified. • MAY — This property is OPTIONAL when the parent is specified.
Compliance	An indication of the relative importance of supporting the given property. • MUST — A compliant implementation MUST support this property. • SHOULD — A compliant implementation SHOULD support this property. • MAY — A compliant implementation MAY support this property.
Categories	Some properties may be categorized into one of several categories. Letters in this column indicate that the given property can be classified in the following property categories. • R — This property can be encompassed in a Requested element. • D — This property can be encompassed in a Delivered element.

2.3 Element Syntax Cardinality

Selected elements in the element syntax sections use regular expression wildcards with the following meanings:

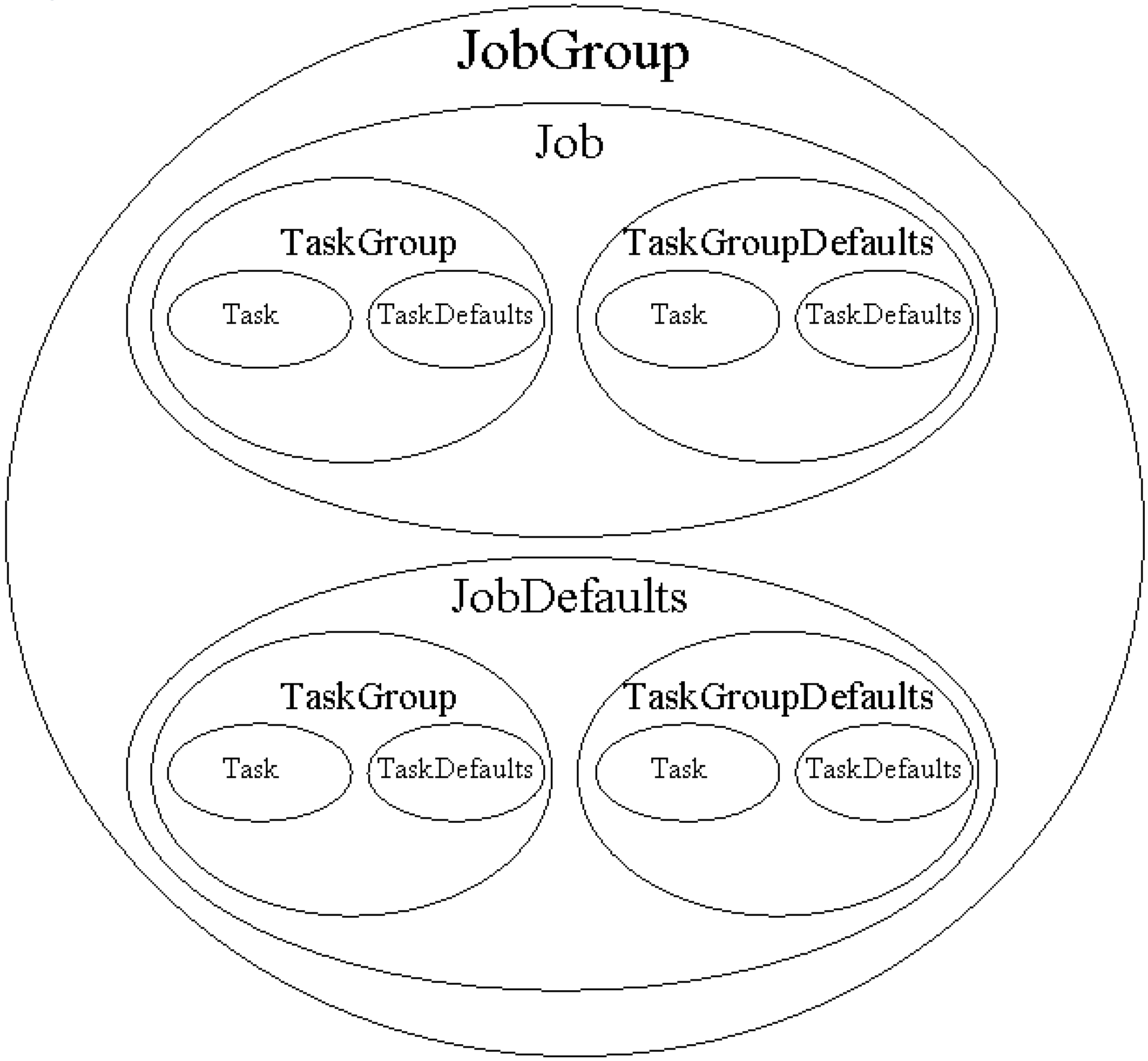
Wildcard	Description
*	Zero or more occurrences
+	One or more occurrences
?	Zero or one occurrences

The absence of one of these symbols implies exactly one occurrence.

3.0 The Job Model

The primary object within the job model is a job. A job can be thought of as a single schedulable entity and will be the object normally seen in job queues.

Image 3-16: JobGroup contains Job and JobDefaults, which contain TaskGroup and TaskGroupDefaults



Jobs with dependencies on other jobs may be submitted in a job group. Jobs within a job group form a DAG (directed acyclic graph) where the nodes are jobs and the edges represent dependencies on the status of previous jobs. A job group will consist of at least one job. A job group can optionally specify job defaults which are a set of job properties to be assumed by all jobs within the job group unless overridden within the job.

A job may consist of multiple tasks, which are the finest grained work unit and represent an endpoint for executing a given process instance. For example, a job that requests 3 nodes and 4 processors will have 4 tasks, two on one node and one on each of two nodes. Tasks may be grouped into task groups, which are logical aggregations of tasks and their common properties. Submit filters, prologs, epilogs, notification scripts, etc. run once only for each job. Whereas task groups function as logical descriptions of tasks and their properties, they also describe the number of such tasks and the nodes that they run on. As an example, a master task group (consisting of a single task) might ask for a node with a MATLAB license, 2GB of memory and an Internet connected network adapter while a slave task group (consisting of 12 tasks) could be targeted for nodes with more CPU bandwidth -- all within the same job and utilizing a common MPI ring. Tasks (and hence taskgroups) can have different executables or environments, specify different consumable resources or node properties. A job, therefore, may specify one or more task group. A job that does not specify an explicit task group is considered as having a single implicit task group. A job can optionally specify task group defaults which are a set of task group properties to be assumed by all task groups within the job unless overridden within a task group.

A task group may specify one or more tasks. A task group that does not specify an explicit task is considered as having a single implicit task. A task group can optionally specify task defaults which are a set of task properties to be assumed by all tasks within the task group unless overridden within a task.

4.0 JobGroup Element

A JobGroup is an optional element that aggregates one or more interdependent jobs. Some resource managers support the submission of job groups (multi-step jobs) and queries on the status of an entire job group.

- A compliant implementation MAY support this element.
- A JobGroup MUST specify one or more JobGroup Properties.
- A JobGroup MUST contain one or more Jobs.
- A JobGroup MAY contain zero or more JobsDefaults.

The following illustrates this element's syntax:

```
C<JobGroup>
  <!-- JobGroup Properties -->+
  <Job/>+
  <JobDefaults/>?
</JobGroup>
```

4.1 JobGroup Properties

JobGroup Properties are properties that apply to the job group as a whole. These include the job group id, jobs and job defaults, and other simple optional job properties.

Simple JobGroup Properties

Simple (unstructured) job group properties are enumerated in the table below.

Table 3-7: Simple JobGroup Properties

Element Name	Type	Description	Appearance	Compliance
CreationTime	DateTime	Date and time that the job group was instantiated	MAY	MAY

Element Name	Type	Description	Appearance	Compliance
Description	String	Description of the job group	MAY	MAY
Id	String	Job group identifier	MUST	MUST
Name	String	Name of the job group	MAY	SHOULD
State	String	State of the job group as a whole. Valid states may include NotQueued, Unstarted, Active, and Completed.	MAY	SHOULD

Job

A job group MUST contain one or more jobs.

See the next section for element details.

JobDefaults

A job group MAY contain zero or one job defaults.

See the next section for element details.

4.2 JobGroup Reference

When a simple reference to a predefined job group is needed in an encapsulating element, a JobGroup element is used with the text content being the job group id:

```
<JobGroup> workflow1</JobGroup>
```

5.0 Job and JobDefaults Element

The Job and JobDefaults elements are of the same structure. A Job element encapsulates a job and may be expressed as a standalone object. A JobDefaults element may only appear within a JobGroup and represents the defaults to be taken by all jobs within the job group. Job properties in Job elements override any properties found in a sibling JobDefaults element.

- A compliant implementation MUST support the Job element.
- A compliant implementation MAY support the JobDefaults element only if it supports the JobGroup element.
- A job MUST specify one or more Job Properties.
- One or more TaskGroup elements MAY appear at this level.
- Zero or one TaskGroupDefaults elements MAY appear at this level.

The following illustrates this element’s syntax:

```
<Job>
  <!-- Job Properties -->+
  <TaskGroup/>*
  <TaskGroupDefaults/>?
</Job>
```

5.1 Job Properties

Job Properties apply to a particular job or as default properties to all jobs. They include the job id, job credentials, task groups, task group defaults, and other simple optional properties.

Simple Job Properties

Simple (unstructured) job properties are enumerated in the table below.

Table 3-8: Simple Job Properties

Element Name	Type	Description	Appearance	Compliance	Categories
Application	String	Type of application such as Gaussian or Nwchem	MAY	MAY	

Element Name	Type	Description	Appearance	Compliance	Categories
Architecture	String	Type architecture for the nodes on which this job must run	MAY	MAY	RD
Arguments	String	The arguments for the executable	MAY	SHOULD	
Charge	Float	The amount charged for the job	MAY	SHOULD	
Checkpointable	Boolean	Can this job be checkpointed?	MAY	MAY	
CpuDuration	Duration	Number of cpu seconds used by the job	MAY	SHOULD	
DeadlineTime	DateTime	Date and time that a job must end by	MAY	MAY	
EligibleTime	DateTime	Date and time that a job must start after	MAY	MAY	
EndTime	DateTime	Date and time that a job ended (independent of success or failure)	MAY	MUST	
Executable	String	Executable. This may be an absolute or relative path or a URI.*	MAY	MUST	

Element Name	Type	Description	Appearance	Compliance	Categories
ExitCode	Integer	Exit code for the job	MAY	SHOULD	
GlobalJob	String	Globally unique job identifier (possibly in the form of a URI)	MAY	SHOULD	
Hold	String	Hold(s) on the job. There may be multiple instances of this element if there is more than one ld on the job	MAY	SHOULD	
InitialWorking-Directory	String	Initial working directory	MAY	SHOULD	
Interactive	Boolean	Is this an interactive job?	MAY	SHOULD	
Id	String	A local job identifier assigned to the job by the local resource manager	MUST	MUST	
Name	String	Name of the job	MAY	SHOULD	
State	String	State of the job. Valid states may include Idle, Hold, Running, Suspended,or Completed	MAY	MUST	

Element Name	Type	Description	Appearance	Compliance	Categories
Type	String	Type of job. Meaning of this extension property is context specific.	MAY	MAY	
Machine	String	Name of the system or cluster that runs the job	MAY	MUST	RD
Network	String	Type of network adapter required by the job	MAY	MAY	RD
NodeCount	Integer	Number of nodes used by the job	MAY	MUST	RD
OperatingSystem	String	Operating System required by the job	MAY	MAY	RD
Partition	String	Name of the partition in which the job should run	MAY	MAY	RD
Priority	Integer	Current queue priority (or rank)for the job	MAY	SHOULD	
QualityOfService	String	Name of the Quality of Service (QoS)	MAY	SHOULD	RD
Queue	String	Name of the Queue (or class) that the job runs in	MAY	SHOULD	RD

Element Name	Type	Description	Appearance	Compliance	Categories
Quote	String	Identifier for a guaranteed charge rate quote obtained by the job	MAY	MAY	
Reservation	String	Identifier for a reservation used by the job	MAY	MAY	RD
ReservationTime	DateTime	Date and time that a reservation was placed for the job	MAY	MAY	
ResourceManagerType	String	Type of resource manager required to run this job	MAY	MAY	RD
Restartable	Boolean	Can this job be restarted?	MAY	MAY	
Shell	String	Specified the shell necessary to interpret the job script	MAY	MAY	
StagedTime	DateTime	Date and time that a job was staged to the local resource management system	MAY	MAY	
StartCount	Integer	Number of times the scheduler tried to start the job	MAY	MAY	

Element Name	Type	Description	Appearance	Compliance	Categories
StartTime	DateTime	Date and time that the job started	MAY	MUST	
StatusMessage	String	Natural language message that can be used to provide detail on why a job failed, isn't running, etc.	MAY	SHOULD	
SubmitTIme	DateTime	Date and time that a job was submitted	MAY	SHOULD	
SubmitHost	String	FQDN of host where the job was submitted from	MAY	SHOULD	
Suspendable	Boolean	CAn this job be suspended?	MAY	MAY	
SuspendDuration	Integer	Number of seconds the job was in the Suspended state	MAY	MAY	
TimeCategory	String	This allows the specification of shifts like PrimeTime for charging purposes	MAY	MAY	
Duration	Duration	Number of seconds in the Running state	SHOULD	MUST	RD

* The Executable may be a script or a binary executable. If it is already on the target system it may be referenced by an absolute or relative pathname (relative to InitialWorkingDirectory). If it is passed with the job in a File object (see SSSRMAP), it can be referenced by an absolute or relative URI. An absolute

URI would specify a URL where the file can be downloaded (like with wget). A relative URI is specified by preceding an identifier by a pound sign, as in

```
<Executable>#Script</Executable>
```

It will be found in a File object included along with the Job object with the Script as an identifier, as in

```
<File id="Script">echo hello world</File>
```

Feature Element

The Feature element connotes an arbitrary named feature of a node.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or one times within a given set of Job Properties.
- This element is of type String.
- This element MAY have an `aggregation` attribute of type String that provides a way to indicate multiple values with a single expression. A compliant implementation MAY support the `aggregation` attribute if the Feature element is supported. Possible values for this attribute include:
  - List — a comma-separated list of features
  - Pattern — a regular expression (perl5) matching desired features
- If an `aggregation` attribute is specified with the value of List, this element MAY also have a `delimiter` attribute of type String that indicates what delimiter is used to separate list elements. The default list delimiter is a comma.
- This element MAY be categorized as a requested or delivered property by being encompassed by the appropriate element.

The following is an example of a feature element:

```
<Feature aggregation="List">feature1,feature2</Feature>
```

OutputFile Element

The `OutputFile` element specifies the name of the file to which the output stream (stdout) from the job will be written.

- This element's character content is the name of the file. If this element is omitted or it is empty, then an appropriate output file is auto-determined by the queuing system.
- This element MAY have a `redirectList` attribute which is a comma-separated list of output redirection attributes of type String. A compliant implementation SHOULD support this attribute if `OutputFile` is supported. Possible values for this attribute include:
  - Append — opens the output file for append
  - Close — closes and discards the output stream
  - Flush — output is written to output file as it is generated

- Keep — leave the output file on the execution host
- Merge — merges the output stream into the error stream

Note that when using the `redirectList` attributes, the cumulative effect of the `ErrorFile` and `OutputFile` directives may be order dependent.

The following is an example of an `OutputFile` element:

```
<OutputFile redirectList="Append">~/myjob.out</OutputFile>
```

ErrorFile Element

The `ErrorFile` element specifies the name of the file to which the error stream (stderr) from the job will be written.

- This element's character content is the name of the file. If this element is omitted or it is empty, then an appropriate error file is auto-determined by the queuing system.
- This element MAY have a `redirectList` attribute which is a comma-separated list of error redirection attributes of type String. A compliant implementation SHOULD support this attribute if `ErrorFile` is supported. Possible values for this attribute include:
  - Close — closes and discards the error stream
  - Append — opens the error file for append
  - Flush — output is written to output file as it is generated
  - Keep — leave the output file on the execution host
  - Merge — merges the error stream into the output stream

Note that when using the `redirectList` attributes, the cumulative effect of the `ErrorFile` and `OutputFile` directives may be order dependent.

The following is an example of an `ErrorFile` element:

```
<ErrorFile redirectList="Merge"></ErrorFile>
```

InputFile Element

The `InputFile` element specifies the name of the file from which the input stream (stdin) for the job will be read.

- This element's character content is the name of the file. If this element is omitted or it is empty, then an appropriate input file is auto-determined by the queuing system.
- This element MAY have a `redirectList` attribute which is a comma-separated list of input attributes of type String. A compliant implementation SHOULD support this attribute if `InputFile` is supported. Possible values for this attribute include:
    - Close — closes and discards the input stream

The following is an example of an `InputFile` element:

```
<InputFile redirectList="Close"></InputFile>
```

NotificationList Element

The `NotificationList` element specifies the job-related events or conditions for which a notification will be sent.

- This element's character content is a comma-separated list of events or conditions for which a notification should be sent. Possible values for the elements of this list include:
    - JobStart — send a notification when the job starts
    - JobEnd — send a notification when the job ends
    - All — send notifications for all notifiable events
    - None — do not send notifications for any events
- This element MAY have a `uri` attribute of type String which indicates where the notification is to be sent. A compliant implementation MAY support this attribute if `NotificationList` is supported. The `uri` is in the format: `[scheme://]authority` with the scheme being smtp and the authority being an email address by default.

The following is an example of a `NotificationList` element:

```
<NotificationList uri="smith@business.com">JobStart,JobEnd</NotificationList>
```

ResourceLimitElement

The `ResourceLimit` element represents a resource limit with its name and value.

- This element MUST have a `name` attribute of type String. A compliant implementation MUST support the name attribute if ResourceLimit is supported.
- This element MAY have a `type` attribute of type String that may have the values `Hard` or `Soft`. If the limit is enforced by the operating system, a hard limit is one that cannot be increased once

it is set while a soft limit may be increased up to the value of the hard limit. If the type attribute is omitted, both the soft and hard limits are set.

- This element's character content is the resource limit's value.

Some typical names include:

Name	Description
CoreFileSize	Maximum core file size
CpuTime	CPU time in seconds
DataSegSize	Maximum data size
FileSize	Maximum file size
MaxMemorySize	Maximum resident set size
MaxProcesses	Maximum number of processes
MaxSwap	Virtual memory limit
MaxMemLock	Maximum locked-in-memory address space
MaxProcessors	Maximum processors
MaxMemory	Maximum memory
MaxDisk	Maximum disk space
MaxNetwork	Maximum network bandwidth
MaxFileIO	Maximum file i/o
OpenFiles	Maximum number of open files
Stacksize	Maximum stack size

The following is an example of a `ResourceLimit` element:

```
<ResourceLimit name="CPUTime">1000000</ResourceLimit>
```

Credentials

Credentials are a special group of job properties that characterize an authenticated token or id. They can be categorized in both requested and delivered forms.

Credential job properties are enumerated in the table below.

Table 3-9: Credential Job Properties

Element Name	Type	Description	Appearance	Compliance	Categories
Project	String	Name of the Project or Charge Account	MAY	SHOULD	RD
GlobalUser	String	Globally unique user identifier. This may be an X.509 DN for example	MAY	SHOULD	RD
Group	String	Name of the local group id	MAY	MAY	RD
User	String	Name of the local user id for the job	MAY	MUST	RD

Environment Element

The Environment element encapsulates environment variables.

- This element MAY have an export attribute of type Boolean that which if set to `True` indicates that all environment variables in the context of the job submission process should be exported in the job's execution environment.
- A compliant implementation SHOULD support this element.
- An Environment element MAY appear zero or one times within a given set of Job (or TaskGroup) Properties.
- An Environment element MAY contain one or more Variable elements.

The following illustrates this element's syntax:

```
<Environment>
  <Variable/>+
</Environment>
```

Variable Element

The Variable element represents an environment variable with its name and value.

This element MUST have a `name` attribute of type String. A compliant implementation MUST support the name attribute if Variable is supported. This element's character content is the environment variable's value.

The following is an example of a Variable element:

```
<Variable name="PATH">/usr/bin:/home/sssdemo</Variable>
```

Node Element

The Node element represents a node.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or one times within a given set of Job Properties.
- This element is of type String.
- This element MAY have an `aggregation` attribute of type String that provides a way to indicate multiple values with a single expression. A compliant implementation MAY support the `aggregation` attribute if the Feature element is supported. Possible values for this attribute include:
    - List - a comma-separated list of features
    - Pattern - a regular expression (perl5) matching desired features
    - Range - a range of nodes of the form: <prefix>[5-23,77]
- If an `aggregation` attribute is specified with the value of List, this element MAY also have a `delimiter` attribute of type String that indicates what delimiter is used to separate list elements. The default list delimiter is a comma.
- This element MAY have a `count` attribute of type Integer that indicates the instance count of the specified node(s).
- This element MAY be categorized as a requested or delivered property by being encompassed by the appropriate element.

The following is an example of a Node element:

```
<Node aggregation="Pattern">node[1-5]</Node>
```

TaskDistribution Element

The `TaskDistribution` element describes how tasks are to be mapped to nodes. This mapping may be expressed as a rule name, a task per node ratio or an arbitrary geometry.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or one times in a given set of Job (or TaskGroup) Properties.
- This element is of type String.
- This element MAY have a `type` attribute of type String that provides a hint as to the type of mapping guidance provided. It may have values including `Rule`, `TasksPerNode`, `ProcessorsPerTask` or `Geometry`. A compliant implementation MAY support the `type` attribute if the `TaskDistribution` element is supported.
- It is possible to use `Processors`, `NodeCount` and `TaskCount` elements to specify a set of mutually contradictory task parameters. When this occurs, components are responsible for resolving conflicting requirements.

The following are three examples of a `TaskDistribution` element:

```
<TaskDistribution type="TasksPerNode">2</TaskDistribution>
<TaskDistribution type="Rule">RoundRobin</TaskDistribution>
<TaskDistribution type="Geometry">{1,4}{2}{3,5}</TaskDistribution>
```

Dependency Element

The Dependency element allows a job's execution to depend on the status of other jobs. In a job group (multi-step job), some jobs may delay execution until the failure or success of other jobs creating in general a Directed Acyclic Graph relationship between the jobs. This element's content is of type String and represents the job that the current job is dependent upon. Since a job may have two or more dependencies, this element may appear more than once in a given job scope. A compliant implementation SHOULD support this element if job groups are supported.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or more times in a given set of Job (or TaskGroup) Properties.
- This element is of type String and contains the JobId that the current job is dependent upon.
- This element MAY have a `condition` attribute of type String that indicates the basis for determining when the current job executes in relation to the specified job. A compliant implementation MUST support this attribute if this element is supported.
  Possible values for this attribute include:
  - OnSuccess this job should run after the referenced job only if it completes successfully (this is the default if the `type` attribute is omitted)

- OnFailure this job should run after the referenced job only if it fails
- OnExit this job should run after the referenced job exits

- If the `condition` attribute is equal to OnExit, this element MAY have a `code` attribute of type Integer that indicates the exit code that will trigger this job to run. If the code attribute is omitted, then the current job should run after the referenced job for any exit status.
- This element MAY have a `designator` attribute of type String that indicates that indicates the property of the job that identifies it as the dependent job. A compliant implementation MAY support this attribute if this element is supported.
  Possible values for this attribute include:
  - JobId the job this job is dependent upon is specified by JobId (this is the default if the designator attribute is omitted)
  - JobName the job(s) this job is dependent upon are specified by JobName

The following is an example of a Dependency element:

```
<Dependency condition="OnSuccess" designator="JobId">PBS.1234.0</Dependency>
```

Consumable Resources

Consumable Resources are a special group of properties that can have additional attributes and can be used in multiple contexts. In general a consumable resource is a resource that can be consumed in a measurable quantity.

- A consumable resource MAY have a `context` attribute of type String that indicates the sense in which the resource is used. A compliant implementation MAY support this attribute. Possible values for this attribute include:
  - Configured — run this task only on nodes having the specified configured resources
  - Available — run this task only on nodes having the specified available resources. (this is the default if the `context` attribute is omitted)
  - Used — the task used the indicated resources (this is analogous to being including in a Delivered block)
  - Dedicated — the indicated amount of the resource should be dedicated to the task
- A consumable resource MAY have a `units` attribute that is of type String that specifies the units by which it is being measured. If this attribute is omitted, a default unit is implied. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a `metric` attribute that is of type String that specifies the type of measurement being described. For example, the measurement may be a Total, an Average, a Min or a Max. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a `duration` attribute of type Duration that indicates the amount of time for which that resource was used. This need only be specified if the resource was

used for a different amount of time than the duration for the job. A compliant implementation MAY support this attribute if the element is supported.

- A consumable resource MAY have a `consumptionRate` attribute of type Float that indicates the average percentage that a resource was used over its duration. For example, an overbooked SMP running 100 jobs across 32 processors may wish to scale the usage and charge by the average fraction of processor usage actually delivered. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a `dynamic` attribute of type Boolean that indicates whether the resource allocated for this job should be allowed to grow or shrink dynamically. For example, if processors is specified with dynamic equal to True, the job may be dynamically allocated more processors as they become available. The growth bounds can be indicated via the `op` attribute which is inherited when a consumable resource element is encapsulated within a *Requested* element. A compliant implementation MAY support this attribute if the element is supported.

A list of simple consumable resources is listed in the table below.

Table 3-10: Simple Consumable Resources

Element Name	Type	Description	Appearance	Compliance	Categories
Disk	Float	Amount of disk	MAY	SHOULD	RD
Memory	Float	Amount of memory	MAY	SHOULD	RD
Network	Float	Amount of network	MAY	MAY	RD
Processors	Integer	Number of processors	MAY	MUST	RD
Swap	Float	Amount of virtual memory	MAY	MAY	RD

The following are two examples for specifying a consumable resource:

```
<Memory metric="Max" units="GB">483</Memory>
<Processors duration="1234" consumptionRate="0.63">4</Processors>
```

Resource Element

In addition to the consumable resources enumerated in the above table, an extensible consumable resource is defined by the Resource element.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or more times within a given set of job (or task group) properties.
- Like the other consumable resources, this property MAY be categorized as a requested or delivered property by being encompassed in the appropriate element.
- This element is of type Float.
- This element shares the *same properties and attributes as the other consumable resources* but it requires an additional `name` (and optional type) attribute to describe it.
- It MUST have a `name` attribute of type String that indicates the type of consumable resource being measured. A compliant implementation MUST support this attribute if the element is supported.
- It MAY have a `type` attribute of type String that distinguishes it within a general resource class. A compliant implementation SHOULD support this attribute if the element is supported.

The following are two examples for specifying a Resource element:

```
<Resource name="License" type="MATLAB">1</Resource>
<Resource name="Telescope" type="Zoom2000" duration="750" metric="KX">10</Resource>
```

Extension Element

The Extension element provides a means to pass extensible properties with the job object.

Some applications may find it easier to use a named extension property than discover and handle elements they do not understand or anticipate by name.

- A compliant implementation MAY support this element.
- This element MUST have a `name` attribute of type String that gives the extension property's name. A compliant implementation MUST support this attribute if this element is supported.
- This element MAY have a `type` attribute of type String that characterizes the context within which the property should be understood. A compliant implementation SHOULD support this attribute if this element is supported.
- This element's character content, which is of type String, is the extension property's value.

The following is an example of an Extension element:

```
<Extension type="Scheduler" name="Restartable">true</Extension>
```

TaskGroup

A job MAY specify one or more task groups.

See the next section for element details.

TaskGroupDefaults

A job MAY specify zero or more task group defaults.

See the next section for element details.

5.2 Job Reference

When a simple reference to a predefined job is needed in an encapsulating element, a Job element is used with the text content being the job id:

```
<Job> job123</Job>
```

6.0 TaskGroup and TaskGroupDefaults Element

The `TaskGroup` and `TaskGroupDefaults` elements have the same structure. A `TaskGroup` element aggregates tasks. A TaskGroupDefaults element may only appear within a Job (or JobDefaults) and represents the defaults to be taken by all task groups within the job. Task group properties in `TaskGroup` elements override any properties found in a sibling `TaskGroupDefaults` element.

- A compliant implementation MAY support the `TaskGroup` element.
- A compliant implementation MAY support the `TaskGroupDefaults` element.
- A task group MUST specify one or more TaskGroup Properties.
- One or more Task elements MAY appear at this level.
- Zero or one TaskDefaults elements MAY appear at this level.

The following illustrates this element's syntax:

```
<TaskGroup>
  <!-- TaskGroup Properties -->+
  <!-- Job Properties -->*
  <Task>+
  <TaskDefaults>?
</TaskGroup>
```

6.1 TaskGroup Properties

TaskGroup Properties apply to a particular task group or as default properties to encompassed task groups. These properties include the task group id, its tasks, task defaults, and other simple task group properties.

Simple TaskGroup Properties

Simple (unstructured) task group properties are enumerated in Table 6.

Table 3-11: Simple TaskGroup Properties

Element Name	Type	Description	Appearance	Compliance	Categories
TaskCount	Integer	Number of tasks in this taskgroup	MAY	MUST	
Id	String	A task group identifier unique within the job	MAY	MAY	
Name	String	A task group name (such as Master)	MAY	SHOULD	

Task

A task group MAY specify zero or more tasks.

See the next section for element details.

TaskDefaults

A task group MAY specify zero or more task defaults.

See the next section for element details.

6.2 TaskGroup Reference

When a simple reference to a predefined task group is needed in an encapsulating element, a TaskGroup element is used with the text content being the task group id:

```
<TaskGroup> tg1</TaskGroup>
```

7.0 Task and TaskDefaults Element

The `Task` and `TaskDefaults` elements have the same structure. A `Task` element contains information specific to a task (like the process id or the host it ran on). A `TaskDefaults` element may only appear within a `TaskGroup` (or `TaskGroupDefaults`) element and represents the defaults for all tasks within the task group. Task properties in `Task` elements override any properties found in a sibling `TaskDefaults` element.

- A compliant implementation MAY support the `TaskGroup` element.
- A compliant implementation MAY support the `TaskGroupDefaults` element.
- A task group MUST specify one or more `TaskGroup` Properties.
- One or more `Task` elements MAY appear at this level.
- Zero or one `TaskDefaults` elements MAY appear at this level.

The following illustrates this element's syntax:

```
<Task>
  <!-- Task Properties -->+
  <!-- Job Properties -->*
</Task>
```

7.1 Task Properties

Task Properties are properties that apply to a particular task or as default properties to encompassed tasks. These properties include the task id and other task properties.

Simple Task Properties

Simple (unstructured) task properties are enumerated in the table below.

Table 3-12: Simple Task Properties

Element Name	Type	Description	Appearance	Compliance	Categories
Node	String	Name of the node this task ran on	MAY	MUST	
Session	Integer	Session id for the task group or job	MAY	MAY	

Element Name	Type	Description	Appearance	Compliance	Categories
Id	String	A task identifier unique within the taskgroup	MAY	MAY	

7.2 Task Reference

When a simple reference to a predefined task is needed in an encapsulating element, a `Task` element is used with the text content being the task id:

```
<Task>1</Task>
```

8.0 Property Categories

Certain properties need to be classified as being in a particular category. This is done when it is necessary to distinguish between a property that is requested and a property that was delivered. When no such distinction is necessary, it is recommended that the property not be enveloped in one of these elements. In general, a property should be enveloped in a category element only if it is expected that the property will need to be attributed to more than one property category, or if it needs to make use of some of the special attributes inherited from the category.

8.1 Requested Element

A requested property reflects properties as they were requested. A disparity might occur between the requested value and the value delivered if a preference was expressed, if multiple options were specified, or if ranges or pattern matching was specified.

- A compliant implementation SHOULD support this element.

The following illustrates the syntax of this element:

```
<Requested>
  <!-- Requested Properties -->+
</Requested>
```

The following describes the attributes and elements for the example above:

`/Requested`

This element is used to encapsulate requested properties.

`/Requested/<Requested Property>`

Requested properties appear at this level.

Requested Properties inherit some additional attributes.

- A requested property MAY have an `op` attribute of type String that indicates a conditional operation on the value. A compliant implementation SHOULD support this attribute. Valid values for the `op` attribute include EQ meaning equals (which is the default), NE meaning not equal, LT meaning less than, GT meaning greater than, LE meaning less than or equal to, GE meaning greater than or equal to, Match which implies the value is a pattern to be matched.
- A requested property MAY have a `conj` attribute of type String that indicates a conjunctive relationship with the previous element. A compliant implementation MAY support this attribute. Valid values for the conj attribute include And (which is the default), Or, Nand meaning and not, and Nor meaning or not.
- A requested property MAY have a `group` attribute of type Integer that indicates expression grouping and operator precedence much like parenthetical groupings. A compliant implementation MAY support this attribute. A positive grouping indicates the number of nested expressions being opened with the property while a negative grouping indicates the number of nested expressions being closed with the property.
- A requested property MAY have a `preference` attribute of type Integer that indicates a preference for the property along with a weight (the weights are taken as a ratio to the sum of all weights in the same group). A compliant implementation MAY support this attribute. If a group of positive valued preference alternatives are specified, at least one of the preferences must be satisfied for the job to run. If a group of negative valued preferences are specified, the preferences will try to be met according to their weights but the job will still run even if it can't satisfy any of the preferred properties. (Weight ranking can be removed by making all weights the same value (1 or -1 for example).
- A requested property MAY have a `performanceFactor` attribute of type Float that provides a hint to the scheduler of what performance tradeoffs to make in terms of resources and start time. A compliant implementation MAY support this attribute.

The following are four examples of using Requested Properties:

```
<Requested>
  <Processors op=”GE”>8</Processors>
  <Processors op=”LE”>16</Processors>
  <Duration>3600</Duration>
</Requested>
<Requested>
  <NodeCount>1</NodeCount>
  <Node aggregation=”Pattern”>fr15.*</Node>
<Requested>
<Requested>
  <User group=”1”>scottmo</User>
  <Account group=”-1”>mscfops</Account>
  <User conj=”Or” group=”1”>amy</User>
  <Account group=”-1”>chemistry</Account>
</Requested>
<Requested>
  <Memory preference=”2”>1024</Memory>
  <Memory preference=”1”>512</Memory>
</Requested>
```

8.2 Delivered Element

A delivered property reflects properties as they were actually utilized, realized or consumed. It reflects the actual amounts or values that are used, as opposed to a limit, choice or pattern as may be the case with a requested property.

- A compliant implementation SHOULD support this element.

The following illustrates the syntax of this element:

```
<Delivered>
  <!-- Delivered Properties -->+
</Delivered>
```

The following describes the attributes and elements for the example above:

`/Delivered`

This element is used to encapsulate delivered properties.

`/Delivered/<Delivered Property>`

Delivered properties appear at this level.

Delivered Properties inherit some additional attributes.

- A delivered property MAY have a `group` attribute of type Integer that indicates expression grouping and operator precedence much like parenthetical groupings. A compliant implementation MAY support this attribute. A positive grouping indicates the number of nested expressions being opened with the property while a negative grouping indicates the number of nested expressions being closed with the property. The purpose of this attribute would be to logically group delivered properties if they were used in certain aggregations (like a job that spanned machines).

The following are the same four examples distinguishing the delivered amounts and values:

```
<Delivered>
  <Processors>12</Processors>
  <Duration>1234</Duration>
</Delivered>
<Delivered>
  <Node>fr15n03</Node>
</Delivered>
<Delivered>
  <User>scottmo</User>
  <Account>mscfops</Account>
</Delivered>
<Delivered>
  <Memory>1024</Memory>
</Delivered>
```

9.0 AwarenessPolicy Attribute

A word or two should be said about compatibility mechanisms. With all the leeway in the specification with regard to implementing various portions of the specification, problems might arise if an implementation simply ignores a portion of a job specification that is critical to the job function in certain contexts. Given this situation, it might be desirable in some circumstances for jobs to be rejected by sites that fail to fully support that job's element or attributes. At other times, it might be desirable for a job to run, using a best-effort approach to supporting unimplemented features. Consequently, we define an `awarenessPolicy` attribute which can be added as an optional attribute to the Job element or any other containment or property element to indicate how the property (or the default action for the elements that the containment element encloses) must react when the implementation does not understand an element or attribute.

An awareness policy of `Reject` will cause the server to return a failure if it receives a client request in which it does not support an associated element name or attribute name or value. It is reasonable for an implementation to ignore (not even look for) an element or attribute that would not be critical to its function as long as ignoring this attribute or element would not cause an incorrect result. However, any element or attribute that was present that would be expected to be handled in a manner that the implementation does not support must result in a failure.

An awareness policy of `Warn` will accept the misunderstood element or attribute and continue to process the job object on a best effort basis. However a warning MUST be sent (if possible) to the requestor enumerating the elements and attributes that are not understood.

An awareness policy of `Ignore` will accept the unsupported element or attribute and continue to process the job object on a best effort basis. The action could be to simply ignore the attribute.

- This name of this attribute is `awarenessPolicy`.
- This attribute is of type String.
- This attribute can have values of `Reject`, `Warn` or `Ignore`.
- A compliant implementation MAY support this attribute.

- An implementation that does not support an attribute MUST reject any job object which contains elements or attributes that it does not support. Furthermore, it SHOULD return a message to the requestor with an indication of the element or attribute name it did not understand.
- This attribute MAY be present in a property or containment element.
- If an implementation does support the attribute, but it is absent, the default value of `Reject` is implied.
- Individual elements in the job object may override the containing object's awareness policy default by including this attribute. For example, a job might specify an `awarenessPolicy` of Reject at its root (the Job element) but may want to allow a particular subset of elements or attributes to be ignored if not understood. Conversely, a job with a default awarenessPolicy of Ignore might want to classify a subset of its optional elements as Reject if they are indispensable to its correct interpretation. An implementation can opt to check or not check for this attribute at any level it wants but must assume a Reject policy for any elements it does not check.

10.0 References

ISO 8601

ISO (International Organization for Standardization). Representations of dates and times, 1988-06-15. http://www.iso.ch/markete/8601.pdf

DATATYPES

XML Schema Part 2: Datatypes. Recommendation, 02 MAY 2001. http://www.w3.org/TR/xmlschema-2/

Appendix A

Units of Measure Abbreviations

Abbreviation	Definition	Quantity
B	byte	1 byte
KB	Kilobyte	2^10 bytes
MB	Megabyte	2^20 bytes

Abbreviation	Definition	Quantity
GB	Gigabyte	2^30 bytes
TB	Terabyte	2^40 bytes
PB	Petabyte	2^50 bytes
EB	Exabyte	2^60 bytes
ZB	Zettabyte	2^70 bytes
YB	Yottabyte	2^80 bytes
NB	Nonabyte	2^90 bytes
DB	Doggabyte	2^100 bytes

Scalable Systems Software Resource Management and Accounting Protocol (SSSRMAP) Message Format

Resource Management Interface Specs
Release v. 3.0.4
18 JUL 2005

Scott Jackson
Brett Bode
David Jackson
Kevin Walker

Status of This Memo

This is a specification defining an XML message format used between Scalable Systems Software components. It is intended that this specification will continue to evolve as these interfaces are implemented and thoroughly tested by time and experience.

Abstract

This document is a specification describing a message format for the interaction of resource management and accounting software components developed as part of the Scalable Systems Software Center. The SSSRMAP Message Format defines a request-response syntax supporting both functional and object-oriented messages. The protocol is specified in XML Schema Definition. The message elements defined in this specification are intended to be framed within the Envelope and Body elements defined in the SSSRMAP Wire Protocol specification document.

Table of Contents

1.0 Introduction

A major objective of the Scalable Systems Software [SSS] Center is to create a scalable and modular infrastructure for resource management and accounting on terascale clusters including resource scheduling, grid-scheduling, node daemon support, comprehensive usage accounting and user interfaces emphasizing portability to terascale vendor operating systems. Existing resource management and accounting components feature disparate APIs (Application Programming Interfaces) requiring various forms of application coding to interact with other components.

This document proposes a common message format expressed in an XML request-response syntax to be considered as the foundation of a standard for communications between and among resource management and accounting software components. In this document this standard is expressed in two levels of generality. The features of the core SSSRMAP protocol common to all resource management and accounting components in general are described in the main body of this document. The aspects of the syntax specific to individual components are described in component-specific binding documents.

2.0 Conventions Used in This Document

2.1 Keywords

The keywords “MUST”, “MUST NOT”, “REQUIIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC2119 [RFC2119].

2.2 XML Case Conventions

In order to enforce a consistent capitalization and naming convention across all SSSRMAP specifications “Upper Camel Case” (UCC) and “Lower Camel Case” (LCC) Capitalization styles shall be used. UCC style capitalizes the first character of each word and compounds the name. LCC style capitalizes the first character of each word except the first word. [XML_CONV][FED_XML]

1. SSSRMAP XML Schema and XML instance documents SHALL use the following conventions:
   - Element names SHALL be in UCC convention (example: <UpperCamelCaseElement/>.
   - Attribute names SHALL be in LCC convention (example: <UpperCamelCaseElement lowerCamelCaseAttribute=”Whatever”/>.
2. General rules for all names are:
   - Acronyms SHOULD be avoided, but in cases where they are used, the capitalization SHALL remain (example: XMLSignature).
   - Underscores (_), periods (.) and dashes (-) MUST NOT be used (example: use JobId instead of JOB.ID, Job_ID or job-id).

2.3 Schema Definitions

```
SSSRMAP Schema Definitions appear like this
```

In case of disagreement between the schema file and this specification, the schema file takes precedence.

3.0 Encoding

Encoding tells how a message is represented when exchanged. SSSRMAP data exchange messages SHALL be defined in terms of XML schema [XML_SCHEMA].

3.1 Schema Header and Namespaces

The header of the schema definition is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:sssrmap="http://scidac.org/ScalableSystems/SSSRMAP"
  targetNamespace="http://www.scidac.org/ScalableSystems/SSSRMAP"
  elementFormDefault="qualified">
```

3.2 Element Descriptions

The following subsections describe the elements that make up SSSRMAP messages. SSSRMAP messages are transmitted in the Body and Envelope elements as described in the SSSRMAP Wire Protocol specification [WIRE_PROTOCOL].

The Request Element

The `Request` element specifies an individual request. An object-oriented request will have at least one `Object` element while a functional request will not have one. Depending on context, the `Request` element MAY contain one or more `Get` elements or one or more Set elements and any number of `Where` elements. `Option`, `Data`, `File` or `Count` elements may also be included. If a component supports it, chunking may be requested where large response data is possible. Setting the chunking attribute to “True” requests that the server break a large response into multiple chunks (each with their own envelope) so they can be processed in separate pieces.

Only an `action` attribute is required. All other attributes are optional.

Attribute	Description
action	Specifies the action or function to be performed
actor	The authenticated user sending the request
id	Uniquely maps the request to the appropriate response
chunking	Requests that segmentation be used for large response data if set to “True”
chunkSize	Requests that the segmentation size be no larger than the specified amount

```
<complexType name="RequestType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="sssrmap:Object" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:Option" minOccurs="0" maxOccurs="unbounded"/>
    <choice minOccurs="0" maxOccurs="1">
      <element ref="sssrmap:Get" minOccurs="1" maxOccurs="unbounded"/>
      <element ref="sssrmap:Set" minOccurs="1" maxOccurs="unbounded"/>
    </choice>
    <element ref="sssrmap:Where" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:Data" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:Count" minOccurs="0" maxOccurs="1"/>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </choice>
  <attribute name="action" type="string" use="required"/>
  <attribute name="actor" type="string" use="required"/>
  <attribute name="id" type="string" use="optional"/>
  <attribute name="chunking" type="sssrmap:BoolType" use="optional"/>
  <attribute name="chunkSize" type="positiveInteger" use="optional"/>
</complexType>

<element name="Request" type="sssrmap:RequestType"/>
```

The Object Element

The `Object` element is used in an object-oriented request to specify the object receiving the action. It is possible to have multiple `Object` elements in a request if an implementation supports multi-object queries.

The object class name is specified as text content. All attributes are optional.

- `join` – the type of join to be performed with the preceding object
  - A `join` attribute of “Inner” specifies an inner join. This is the default.
  - A `join` attribute of “FullOuter” specifies a full outer join.
  - A `join` attribute of “LeftOuter” specifies a left outer join.
  - A `join` attribute of “RightOuter” specifies a right outer join.
  - A `join` attribute of “Cross” specifies a cross join.
  - A `join` attribute of “Union” specifies a union join.

```
<complexType name="ObjectType">
  <simpleContent>
    <extension base="string">
      <attribute name="join" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Object" type="sssrmap:ObjectType"/>
```

The Get Element

The `Get` element is used to indicate the data fields to be returned in a query. Get is typically used within requests with `action=”query”`. Multiple Get elements cause the fields to be returned in the order specified. If no `Get` elements are specified, the query will return a default set of fields.

Only a `name` attribute is required. All other attributes are optional.

Attribute	Description
name	The name of the data field to be returned. This MUST be of the form of a "Modified XPATH expression" as described in a later section.
op	The operator to be used to aggregate or perform an operation on the returned values. • An *op* attribute of "Sort" specifies an ascending sort operation • An *op* attribute of "Tros" specifies a descending sort operation • An *op* attribute of "Sum" returns the sum (only valid for numeric values) • An *op* attribute of "Max" returns the maximum value • An *op* attribute of "Min" returns the minimum value • An *op* attribute of "Count" returns the number of values • An *op* attribute of "Average" returns the average of the values • An *op* attribute of "GroupBy" signifies that aggregates are grouped by this field
object	Specifies the object for which you want the named attribute in a multi-object query.
units	The units in which to return the value (if applicable)

```
<complexType name="GetType">
  <attribute name="name" type="string" use="required"/>
  <attribute name="object" type="string" use="optional"/>
  <attribute name="op" type="sssrmap:GetOperatorType" use="optional"/>
  <attribute name="units" type="string" use="optional"/>
</complexType>

<element name="Get" type="sssrmap:GetType"/>

<simpleType name="GetOperatorType">
  <restriction base="string">
    <enumeration value="Sort"/>
    <enumeration value="Tros"/>
    <enumeration value="Count"/>
    <enumeration value="Sum"/>
    <enumeration value="Max"/>
    <enumeration value="Min"/>
    <enumeration value="Average"/>
    <enumeration value="GroupBy"/>
  </restriction>
</simpleType>
```

The Set Element

The `Set` element is used to specify the object data fields to be assigned values. Set is typically used within requests with `action="Create"` or `action="Modify"`. The use of `Get` or `Set` elements within a request is mutually exclusive.

The assignment value (to which the field is being changed) is specified as the text content. A Set element without a value may be used as an assertion flag. Only the `name` attribute is required. All other attributes are optional.

Attribute	Description
name	The name of the field being assigned a value. This MUST be of the form of a “Modified XPATH expression” as described in a later section.
op	The operator to be used in assigning a new value to the name. If an `op` attribute is not specified and a value is specified, the specified value will be assigned to the named field (“assign”). • An `op` attribute of “Assign” assigns value to the named field • An `op` attribute of “Inc” increments the named field by the value • An `op` attribute of “Dec” decrements the named field by the value
units	The units corresponding to the value being set

```
<complexType name="SetType">
  <simpleContent>
    <extension base="string">
      <attribute name="name" type="string" use="required"/>
      <attribute name="op" type="sssrmap:SetOperatorType" use="optional"/>
      <attribute name="units" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Set" type="sssrmap:SetType"/>

<simpleType name="SetOperatorType">
  <restriction base="string">
    <enumeration value="Assign"/>
    <enumeration value="Inc"/>
    <enumeration value="Dec"/>
  </restriction>
</simpleType>
```

The Where Element

A `Request` element may contain one or more `Where` elements that specify the search conditions for which objects the action is to be performed on.

The condition value (against which the field is tested) is specified as the text content. A `Where` element without a value may be used as a truth test. Only the `name` attribute is required. All other attributes are optional.

Attribute	Description
name	The name of the data field to be tested. This MUST be of the form of a “Modified XPATH expression” as described in a later section.

Attribute	Description
op	The operator to be used to test the name against the value. If an op attribute is not specified and a value is specified, the field will be tested whether it is equal to the value ("EQ"). • An `op` attribute of "EQ" specifies an equality comparison • An `op` attribute of "LT" specifies a "less than" comparison • An `op` attribute of "GT" specifies a "greater than" comparison • An `op` attribute of "LE" specifies a "less than or equal to" test • An `op` attribute of "GE" specifies a "greater than or equal to" test • An `op` attribute of "NE" specifies a "not equal to" test • An `op` attribute of "Match" specifies a regular expression matching comparison
conj	Indicates whether this test is to be anded or ored with the immediately preceding where condition • A `conj` attribute of "And" specifies an "and" conjunction • A `conj` attribute of "Or" specifies an "or" condition • A `conj` attribute of "AndNot" specifies an "and not" conjunction • A `conj` attribute of "OrNot" specifies an "or not" condition
group	Indicates an increase or decrease of parentheses grouping depth • A positive number indicates the number of left parentheses to precede the condition, i.e. `group="2"` represents "((condition". • A negative number indicates the number of right parentheses to follow the condition, i.e. `group="-2"` represents "condition))".
object	Specifies the object for the first operand in a multi-object query.
subject	Specifies the object for the second operand in a multi-object query.
units	Indicates the units to be used in the value comparison

```
<complexType name="WhereType">
  <simpleContent>
    <extension base="string">
      <attribute name="name" type="string" use="required"/>
      <attribute name="op" type="sssrmap:OperatorType" use="optional"/>
      <attribute name="conj" type="sssrmap:ConjunctionType" use="optional"/>
      <attribute name="group" type="integer" use="optional"/>
      <attribute name="units" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Where" type="sssrmap:WhereType"/>

<simpleType name="WhereOperatorType">
  <restriction base="string">
    <enumeration value="EQ"/>
    <enumeration value="GT"/>
    <enumeration value="LT"/>
    <enumeration value="GE"/>
    <enumeration value="LE"/>
    <enumeration value="NE"/>
    <enumeration value="Match"/>
  </restriction>
</simpleType>
```

The Option Element

The `Option` element is used to indicate processing options for the command. An option might be used to indicate that command usage or special formatting is desired, or that the command is to be invoked with particular options.

The option value is specified as the text content. An `Option` element without a value may be used as an assertion flag. Only the `name` attribute is required. All other attributes are optional.

Attribute	Description
name	The name of the field being assigned a value
op	The operator to be used to disassert the option • An `op` attribute of “Not” specifies that the option is not asserted
conj	Indicates whether this test is to be anded or ored with the immediately preceding where condition • A `conj` attribute of “And” specifies an “and” conjunction • A `conj` attribute of “Or” specifies an “or” condition • A `conj` attribute of “AndNot” specifies an “and not” conjunction • A `conj` attribute of “OrNot” specifies an “or not” condition

```
<complexType name="OptionType">
  <simpleContent>
    <extension base="string">
      <attribute name="name" type="string" use="required"/>
      <attribute name="op" type="sssrmap:OptionOperatorType" use="optional"/>
      <attribute name="conj" type="sssrmap:ConjunctionType" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Option" type="sssrmap:OptionType"/>

<simpleType name="OptionOperatorType">
  <restriction base="string">
    <enumeration value="Not"/>
  </restriction>
</simpleType>
```

The Data Element

A `Request` or `Response` element may have one or more `Data` elements that allow the supplying of context-specific data. A request might pass in a structured object via a `Data` element to be acted upon. Typically a query will result in a response with the data encapsulated within a `Data` element.

The following attributes are optional:

Attribute	Description
name	Object name describing the contents of the data
type	Describing the form in which the data is represented • A `type` attribute of "XML" indicates the data has internal xml structure and can be recursively parsed by an XML parser • A `type` attribute of "Binary" indicates the data is an opaque dataset consisting of binary data • A `type` attribute of "String" indicates the data is an ASCII string • A `type` attribute of "Int" indicates the data is an integer • A `type` attribute of "Text" indicates the data is in formatted human-readable text • A `type` attribute of "HTML" indicates the data is represented in HTML

```
<complexType name="DataType">
  <sequence>
   <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="name" type="string" use="optional"/>
  <attribute ref="sssrmap:Type" use="optional"/>
</complexType>

<element name="data" type="sssrmap:DataType"/>
```

The File Element

A `Request` or `Response` element may have one or more `File` elements of type String that allow the inclusion of files. The files may be either text or binary and may be referenced by objects inside the Data element. A file may be compressed using the gzip algorithm [ZIP]. A binary file or a compressed file must be base64 encoded as defined in XML Digital Signatures ("`http://www.w3.org/2000/09/xmldsig#base64`"). Metadata describing the modes and properties of the resulting file are passed as parameters. The text or base64 encoded file data forms the string content of the `File` element.

The following attributes are optional:

Attribute	Description
id	Specifies an identifier that allows the file to be referenced from within another object. If more than one File elements are specified, this attribute is REQUIRED in each of them.
name	Specifies the name to give the file upon creation on the target system. This can be an absolute or relative pathname (relative to the InitialWorkingDirectory).
owner	Indicates what owner the file should be changed to. By default it will be changed to the UserId that the authenticated actor maps to on the target system. Note that this function should succeed only if the requestor has the privileges to do so (i.e. authenticated as root).
group	Indicates what group the file should be changed to. By default it will be set to the primary groupid of the UserId that the authenticated actor maps to on the target system. Note that this function should succeed only if the requestor has the proper privileges.
mode	Indicates the permissions the file should possess. By default it will be set according to the default umask for the UserId that the authenticated actor maps to on the target system. Note that this function should not set permissions for the file that exceed the privileges for the actor. These permissions can be specified using either an octal number or symbolic operations (as accepted by the GNU chmod(1) command).
compressed	Indicates whether the file has been compressed • A `compressed` attribute of "True" indicates the file has been compressed. • A `compressed` attribute of "False" indicates the file has not been compressed. This is the default.
encoded	Indicates whether the file has been base64 encoded • An `encoded` attribute of "True" indicates the file has been encoded. • An `encoded` attribute of "False" indicates the file has not been encoded. This is the default.

```
<complexType name="FileType">
  <sequence>
    <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="name" type="string" use="optional"/>
  <attribute name="owner" type=="string" use="optional"/>
  <attribute name="group" type="string" use="optional"/>
  <attribute name="mode" type="string" use="optional"/>
  <attribute name="compressed" type="boolean" use="optional"/>
  <attribute name="encoded" type="boolean" use="optional"/>
</complexType>

<element name="file type="sssrmap:FileType"/>
```

The Count Element

A single `Count` element may be included within a `Request` or `Response` and is context-specific. This can be used to represent the number of objects acted upon or returned.

```
<element name="Count" type="positiveInteger"/>
```

The Response Element

The `Response` element specifies an individual response. It MUST contain a `Status` element. It MAY also contain `Count` and any number of `Data` or `File` elements. If chunking has been requested and is supported by the server, a large response may be broken up into multiple chunks (each with their own envelope). The `chunkNum` attribute can be used to indicate which chunk the current one is. The `chunkMax` attribute can be used to determine when all the chunks have been received (all chunks have been received if `chunkNum=chunkMax` or `chunkMax=0`).

It MAY have any of the following attributes:

Attribute	Description
id	Uniquely maps the response to the corresponding request
chunkNum	Integer indicating the current chunk number [1 is implied when this attribute is missing or blank]
chunkMax	Integer indicating the number of chunks expected [-1 means unknown but more chunks to follow; 0 means unknown but this is the last chunk; 0 is implied if this attribute is missing or blank]

```
<complexType name="ResponseType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="sssrmap:Status" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:Count" minOccurs="0" maxOccurs="1"/>
    <element ref="sssrmap:Data" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:File" minOccurs="0" maxOccurs="unbounded"/>
    <any minOccurs="0" maxOccurs="unbounded" namespace="##other"/>
  </choice>
  <attribute name="object" type="string" use="optional"/>
  <attribute name="action" type="string" use="optional"/>
  <attribute name="id" type="string" use="optional"/>
  <attribute name="chunkNum" type="integer" use="optional"/>
  <attribute name="chunkMax" type="integer" use="optional"/>
</complexType>

<element name="Response" type="sssrmap:ResponseType"/>
```

The Status Element

A `Response` element MUST contain a single `Status` element that indicates whether the reply represents a success, warning or failure. This element is composed of the child elements `Value`, `Code` and `Message`. Of these, `Value` and `Code` are required, and `Message` is optional.

```
<complexType name="StatusType">
  <choice minOccurs="1" maxOccurs="unbounded">
    <element ref="sssrmap:Value" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:Code" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:Message" minOccurs="0" maxOccurs="1"/>
    <any minOccurs="0" maxOccurs="unbounded" namespace="##other"/>
  </choice>
</complexType>

<element name="Status" type="sssrmap:StatusType"/>
```

The Value Element

The `Value` element is of type String and MUST have a value of “Success”, “Warning” or “Failure”.

```
<simpleType name="StatusValueType">
  <restriction base="string">
    <enumeration value="Success"/>
    <enumeration value="Warning"/>
    <enumeration value="Failure"/>
  </restriction>
</simpleType>

<element name="Value" type="sssrmap:StatusValueType"/>
```

The Code Element

A `Response` element must contain a single `Code` element that specifies the 3-digit status code for the response. Refer to the next section on Error Reporting for a description and listing of supported status codes.

```
<simpleType name="CodeType">
  <restriction base="string">
    <pattern value="[0-9]{3}"/>
  </restriction>
</simpleType>

<element name="Code" type="sssrmap:CodeType"/>
```

The Message Element

A `Response` element may contain a single `Message` element that is context specific to the success or failure response. The message should be an error message if status is false. If present for a successful response, it may be used as a human readable message for a user interface.

```
<element name="Message" type="string"/>
```

3.3 Modified XPATH Expressions

The `name` attribute used within the `Get`, `Set` and `Where` Elements MUST have the form of a modified XPATH expression as defined in this section. Usually this will just be the simple name of the object property. Some complex objects, such as the SSS Job Object and the SSS Node Object, however, are represented in a structured way with nested elements. In order to define a consistent and flexible way to access and manipulate these objects as well as keeping the flat XML objects simple and straightforward, SSSRMAP specifies that a "Modified XPATH" syntax be used.

In essence, "Modified XPATH" is defined to be an XPATH [XPATH] expression with the exception that the "//" may be omitted from the beginning of the expression when a document search is desired. Thus, on the server side, a standard XPATH routine can be used by prepending "//" to any expression that does not begin with a "/".

The response data should always include all of the structure of the queried object necessary to place the requested data in its proper context.

See the XPATH specification for a full description of XPATH. The XPath 1.0 Recommendation is `http://www.w3.org/TR/1999/REC-xpath-19991116`. The latest version of XPath 1.0 is available at http://www.w3.org/TR/xpath.

Sample Modified XPATH Expressions

Consider the following hypothetical object(s) (which might be returned within a Data element).

```
<Job>
  <JobId>PBS.1234.0</JobId>
  <Requested>
    <Memory op="GE">512</Memory>
    <Processors>2</Processors>
    <WallDuration>P3600S</WallDuration>
  </Requested>
  <Utilized>
    <Memory metric="Average">488</Memory>
    <WallDuration>P1441S</WallDuration>
  </Utilized>
</Job>
```

To get everything above for this job you would not need a `Get` element:

```
<Request action="Query">
  <Object>Job</Object>
  <Where name="JobId">PBS.1234.0</Where>
</Request>
```

If you used `<Get name="JobId"/>` you would get back:

```
<Job>
  <JobId>PBS.1234.0</JobId>
</Job>
```

If you used `<Get name="Memory"/>` (or `name="/Job/*/Memory"`) you would get:

```
<Job>
  <Requested>
    <Memory op="GE">512</Memory>
  </Requested>
  <Utilized>
    <Memory metric="Average">488</Memory>
  </Utilized>
</Job>
```

If you used `<Get name="Requested/Memory"/>` (or `name="/Job/Requested/Memory"`) you would get:

```
<Job>
  <Requested>
    <Memory op="GE">512</Memory>
  </Requested>
</Job>
```

If you used `<Get name="Memory[@metric='Average']"/>` (or `name="Memory[@metric]"`) you would get:

```
<Job>
  <Utilized>
    <Memory metric="Average">488</Memory>
  </Utilized>
</Job>
```

3.4 Examples

Sample Requests

Requesting a list of nodes with a certain configured memory threshold (batch format):

```
<Request action="Query" id="1">
  <Object>Node</Object>
  <Get name="Name" />
  <Get name="Configured/Memory" />
  <Where name="Configured/Memory" op="GE" units="MB">512</Where>
</Request>
```

Activating a couple of users:

```
<Request action="Modify">
  <Object>User</Object>
  <Set name="Active">True</Set>
  <Where name="Name">scott</Where>
  <Where name="Name" conj="Or"/>brett</Where>
</Request>
```

Submitting a simple job:

```
<Request action="Submit">
  <Object>Job</Object>
  <Data>
    <Job>
      <User>xdp</User>
      <Account>youraccount</Account>
      <Command>myprogram</Command>
      <InitialWorkingDirectory>/usr/home/scl/xdp</InitialWorkingDirectory>
      <RequestedNodes>4</RequestedNodes>
      <RequestedWCTime>100</RequestedWCTime>
    </Job>
  </Data>
</Request>
```

Sample Responses

A response to the available memory nodes query (batch format)

```
<Response id="1">
  <Status>
    <Value>Success</Value>
    <Code>000</Code>
  </Status>
  <Count>2</Count>
  <Data>
    <Node>
      <Name>fr01n01</Name>
      <Configured>
        <Memory>512</Memory>
      </Configured>
    </Node>
    <Node>
      <Name>fr12n04</Name>
      <Configured>
        <Memory>1024</Memory>
      </Configured>
    </Node>
  </Data>
</Response>
```

Two users successfully activated

```
<Response>
  <Status>
    <Code>000</Code>
    <Message>Two users were successfully modified</Message>
  </Status>
  <Count>2</Count>
</Response>
```

A failed job submission:

```
<Response>
  <Status>
    <Value>Failure</Value>
    <Code>711</Code>
    <Message>Invalid account specified. The job was not submitted.</Message>
  </Status>
</Response>
```

4.0 Error Reporting

SSSRMAP requests will return a status and a 3-digit response code to signify success or failure conditions. When a `request` is successful, a corresponding `response` is returned with the `status` element set to Success and the `code` element set to "000". When a `request` results in an error detected by the server, a `response` is returned with the `status` element set to Failure and a 3-digit error code in the `code` element. An optional human-readable `message` may also be include in a failure response providing context-specific detail about the failure. The default message language is US English. (The status flag makes it easy to signal success or failure and allows the receiving peer some freedom in the amount of parsing it wants to do on failure [BXXP]).

Success codes:

Code	Response Text in US English
0xx	Request was successful
000	General Success
010	Help/usage reply
020	Status reply
030	Subscription successful
035	Notification successful (Ack)
040	Registration successful
050-079	Component-defined
080-099	Application-defined

Warning codes:

Code	Response Text in US English
1xx	Request was successful but includes a warning
100	General warning (examine message for details)
102	Check result (Did what you asked but may not have been what you intended -- or information is suspect)
110	Wire Protocol or Network warning
112	Redirect
114	Protocol warning (something was wrong with the protocol, but best effort guesses were applied to fulfill the request)
120	Message Format warning

Code	Response Text in US English
122	Incomplete specification (request missing some essential information -- best effort guess applied)
124	Format warning (something was wrong with the format but best effort guesses were applied to fulfill the request)
130	Security warning
132	Insecure request
134	Insufficient privileges (Response was sanitized or reduced in scope due to lack of privileges)
140	Content or action warning
142	No content (The server has processed the request but there is no data to be returned
144	No action taken (nothing acted upon -- i.e. deletion request did not match any objects)
146	Partial content
148	Partial action taken
150-179	Component-defined
180-199	Application-defined

Wire protocol codes:

Code	Response Text in US English
2xx	A problem occurred in the wire protocol or network
200	General wire protocol or network error
210	Network failure
212	Cannot resolve host name

Code	Response Text in US English
214	Cannot resolve service port
216	Cannot create socket
218	Cannot bind socket
220	Connection failure
222	Cannot connect
224	Cannot send data
226	Cannot receive data
230	Connection rejected
232	Timed out
234	Too busy
236	Message too large
240	Framing failure
242	Malformed framing protocol
244	Invalid payload size
246	Unexpected end of file
250-279	Component-defined
280-299	Application-defined

Message format codes:

Code	Response Text in US English
3xx	A problem occurred in the message format
300	General message format error
302	Malformed XML document
304	Validation error(XML Schema)
306	Namespace error
308	Invalid message type (Something other than Request or Response in Body
310	General syntax error in request
311	Object incorrectly (or not) specified
312	Action incorrectly (or not) specified
313	Invalid Action
314	Missing required element or attribute
315	Invalid Object (or Object-Action combination
316	Invalid element or attribute name
317	Illegal value for element or attribute
318	Illegal combination
319	Malformed Data
320	General syntax error in response
321	Status incorrectly (or not)specified
322	Code incorrectly (or not)specified

Code	Response Text in US English
324	Missing required element or attribute
326	Invalid element or attribute name
327	Illegal value for element or attribute
328	Illegal combination
329	Malformed Data
340	Pipelining failure
342	Request identifier is not unique
344	Multiple messages not supported
346	Mixed messages not supported (Both requests and responses in same batch)
348	Request/response count mismatch
350-379	Component-defined
380-399	Application-defined

Security codes:

Code	Response Text in US English
4xx	A security requirement was not fulfilled
400	General security error
410	Negotiation failure
412	Not understood
414	Not supported

Code	Response Text in US English
416	Not accepted
420	Authentication failure
422	Signature failed at client
424	Authentication failed at server
426	Signature failed at server
428	Authentication failed at client
430	Encryption failure
432	Encryption failed at client
434	Decryption failed at server
436	Encryption failed at server
438	Decryption failed at client
440	Authorization failure
442	Authorization failed at client
444	Authorization failed at server
450-479	Component-defined
480-499	Application-defined

Event management codes:

Code	Response Text in US English
5xx	Failure conditions in event messaging

Code	Response Text in US English
500	General Event Management failure
510	Subscription failed
520	Notification failed
550-579	Component-defined
580-599	Application-defined

Reserved codes:

Code	Response Text in US English
6xx	Reserved for future use

Server application codes:

Code	Response Text in US English
7xx	A server-side application-specific error occurred
700	General failure
710	Not supported
712	Not understood
720	Internal error
730	Resource unavailable (insufficient resources -- software, hardware or a service I rely upon is down)
740	Business logic
750-779	Component-defined
780-799	Application-defined

Client application codes:

Code	Response Text in US English
8xx	A client-side application-specific error occurred
800	General failure
810	Not supported
812	Not understood
820	Internal error
830	Resource unavailable
840	Business logic
850-879	Component-defined
880-899	Application-defined

Miscellaneous codes:

Code	Response Text in US English
9xx	Miscellaneous failures
999	Unknown failure

5.0 References

[BEEP] M. Rose, "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

[FED_XML] "U.S. Federal XML Guidelines".

[HMAC] H. Krawczyk, M. Bellare, R. Canetti, "HMAC, Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[HTTP] "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616, June 1999.

[RFC2119] S. Bradner, "Key Words for Use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[RFC3117] M. Rose, "On the Design of Application Protocols", Informational RFC 3117, November 2001.

[SHA-1] U.S. Department of Commerce/National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-1.

[SSS] "Scalable Systems Software", http://www.scidac.org/ScalableSystems

[WIRE_PROTOCOL] S. Jackson, B. Bode, D. Jackson, K. Walker, "Systems Software Resource Management and Accounting Protocol (SSSRMAP) Wire Protocol", SSS Resource Management and Accounting Documents, January 2004.

[XML] Bray, T., et al, "Extensible Markup Language (XML) 1.0 (Second Edition)", 6 October 2000.

[XML_CONV] "I-X and <I-N-CA> XML Conventions".

[XML_DSIG] D. Eastlake, J. Reagle Jr., D. Solo, "XML Signature Syntax and Processing", W3C Recommendation, 12 February 2002.

[XML_ENC] T. Imamura, B. Dillaway, E. Smon, "XML Encryption Syntax and Processing", W3C Candidate Recommendation, 4 March 2002.

[XML_SCHEMA] D. Beech, M. Maloney, N. Mendelshohn, "XML Schema Part 1: Structures Working Draft", April 2000.

[XPath 1.0] J. Clark, S. DeRose, "XML Path Language (XPath) Version 1.0", 16 November 1999.

[XRP] E. Brunner-Williams, A. Damaraju, N. Zhang, "Extensible Registry Protocol (XRP)", Internet Draft, expired August 2001.

[ZIP] J. Gailly, M. Adler, "The gzip home page", http://www.gzip.org/

Scalable Systems Software Node Object Specification

SSS Node Object Specification
Release Version 3.1.0
26 April 2011

Scott Jackson, PNNL
David Jackson, Ames Lab
Brett Bode, Ames Lab

Status of This Memo

This is a specification of the node object to be used by Scalable Systems Software compliant components. It is envisioned for this specification to be used in conjunction with the SSSRMAP protocol with the node object passed in the Data field of Requests and Responses. Queries can be issued to a node-cognizant component in the form of modified XPATH expressions to the Get field to extract specific information from the node object as described in the SSSRMAP protocol.

Abstract

This document describes the syntax and structure of the SSS node object. This node model takes into account various node property categories such as whether it represents a configured, available or utilized property.

Table of Contents

1.0 Introduction

This specification proposes a standard XML representation for a node object for use by the various components in the SSS Resource Management System. This object will be used in multiple contexts and by multiple components. It is anticipated that this object will be passed via the Data Element of SSSRMAP Requests and Responses.

1.1 Goals

There are several goals motivating the design of this representation.

It needs to be inherently flexible. We recognize we will not be able to exhaustively include the ever-changing node properties and capabilities that constantly arise.

The same node object should be used at all stages of its lifecycle. This object needs to distinguish between configured, available and utilized properties of a node.

Its design takes into account the properties and structure required to function in a meta or grid environment. It should eventually include the capability of resolving namespace and locality issues, though the earliest versions will ignore this requirement.

One should not have to make multiple queries to obtain a single piece of information — i.e. there should not be two mutually exclusive ways to represent a node resource.

It needs to support resource metric as well as unit specifications.

1.2 Examples

Simple Example

This example shows a simple expression of the Node object.

```
<Node>
  <Id>Node64</Id>
  <Configured>
    <Processors>2</Processors>
    <Memory>512</Memory>
  </Configured>
</Node>
```

Elaborate Example

This example shows a more elaborate Node object.

```
<Node>
  <Id>64</Id>
  <Name>Netpipe2</Name>
  <Feature>BigMem</Feature>
  <Feature>NetOC12</Feature>
  <Opsys>AIX</Opsys>
  <Arch>Power4</Arch>
  <Configured>
    <Processors>16</Processors>
    <Memory units=”MB”>512</Memory>
    <Swap>512</Swap>
  </Configured>
  <Available>
    <Processors>7</Processors>
    <Memory metric=”Instantaneous”>143</Memory>
  </Available>
  <Utilized>
    <Processors wallDuration=”3576”>8</Processors>
    <Memory metric=”Average” wallDuration=”3576”>400</Memory>
  </Utilized>
</Node>
```

2.0 Conventions Used in This Document

2.1 Keywords

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC2119.

2.2 Table Column Interpretations

In the property tables, the columns are interpreted to have the following meanings:

Property	Description
Element Name	Name of the XML element (xsd:element)
Type	Data type defined by xsd (XML Schema Definition) as: • String — xsd:string(a finite length sequence of printable characters) • Integer — xsd:integer(a signed finite length sequence of decimal digits) • Float — xsd:float (single-precision 32-bit floating point) • Boolean — xsd:boolean (consists of the literals "true" or "false") • DateTime — xsd:dateTime (discreet time values are represented in ISO 8601 extended format `CCYY-MM-DDThh:mm:ss` where `CC` represents the century, `YY` the year, `MM` the month and `DD` the day. The letter `T` is the date/time separator and `hh`, `mm`, `ss` represent hour, minute and second respectively. This representation may be immediately followed by a `Z` to indicate Coordinated Universal Time (UTC) or, to indicate the time zone, i.e. the difference between the local time and Coordinated Universal Time, immediately followed by a sign, + or −, followed by the difference from UTC.) • Duration — xsd:duration (a duration of time is represented in ISO 8601 extended format PnYnMnDTnHnMnS, where nY represents the number of years, nM the number of months, nD the number of days, `T` is the date/time separator, nH the number of hours, nM the number of minutes and nS the number of seconds. The number of seconds can include decimal digits to arbitrary precision.)
Description	Brief description of the meaning of the property
Appearance	This column indicates whether the given property has to appear within the parent element. It assumes the following meanings: • MUST — This property is REQUIRED when the parent is specified. • SHOULD — A compliant implementation SHOULD support this property. • MAY — A compliant implementation MAY support this property.

Property	Description
Compliance	The column indicates whether a compliant implementation has to support the given property. • MUST — A compliant implementation MUST support this property. • SHOULD — A compliant implementation SHOULD support this property. • MAY — A compliant implementation MAY support this property.
Categories	Some properties may be categorized into one of several categories. Letters in this column indicate that the given property can be classified in the following property categories. • C — This property can be encompassed in a Configured element. • A — This property can be encompassed in an Available element. • U — This property can be encompassed in a Utilized element.

2.3 Element Syntax Cardinality

The cardinality of elements in the element syntax sections may make use of regular expression wildcards with the following meanings:

Wildcard	Description
*	Zero or more occurrences
+	One or more occurrences
?	Zero or one occurrences

The absence of one of these symbols implies one and only one occurrence.

3.0 The Node Model

The primary element within the node model is a node. One can speak of some node properties as being a configured, available or utilized property of the node.

4.0 Node Element

The Node element is the root element of a node object and is used to encapsulate a node.

- A node object MUST have exactly one Node element.
- A compliant implementation MUST support this element.
- A node MUST specify one or more Node Properties.

4.1 Uncategorized Node Properties

Uncategorized Node Properties are properties that apply to the node as a whole and do not need to be distinguished between being configured, available or utilized. These include the node id and other optional node properties.

Simple Node Properties

Simple (unstructured) node properties are enumerated in the table below.

Table 3-13: Simple Node Properties

Element Name	Type	Description	Appearance	Compliance
Id	String	Node identifier	MUST	MUST
Name	String	Node name or pattern	MAY	MAY
OpSys	String	Operating System	MAY	SHOULD
Arch	String	Architecture	MAY	SHOULD
Description	String	Description of the node	MAY	MAY
State	String	State of the node. Valid states may include Offline, Configured, Unknown, Idle, and Busy.	SHOULD	MUST
Features	String	Arbitrary named features of the node (comma-delimited string)	MAY	SHOULD

Extension Element

The `Extension` element provides a means to pass extensible properties with the node object. Some applications may find it easier to deal with a named extension property than discover and handle elements for which they do not understand or anticipate by name.

- A compliant implementation MAY support this element.
- This element MUST have a name attribute that is of type String and represents the name of the extension property. A compliant implementation MUST support this attribute if this element is supported.
- This element MAY have a type attribute that is of type String and provides a hint about the context within which the property should be understood. A compliant implementation SHOULD support this attribute if this element is supported.
- The character content of this element is of type String and is the value of the extension property.

The following is an example of an `Extension` element:

```
<Extension type="Chemistry" name="Software">NWChem</Extension>
```

4.2 Property Categories

Certain node properties (particularly consumable resources) need to be classified as being in a particular category. This is done when it is necessary to distinguish between a property that is configured versus a property that is available or utilized. For example, a node might be configured with 16 processors. At a particular time, 8 might be utilized, 7 might be available and 1 disabled. When a node property must be categorized to be understood properly, the property MUST be enveloped within the appropriate Property Category Element.

Configured Element

A configured node property reflects resources pertaining to the node that could in principle be used though they may not be available at this time. This information could be used to determine if a job could ever conceivably run on a given node.

- A compliant implementation MUST support this element.

The following is an example of using Configured Properties:

```
<Configured>
  <Processors>16</Processors>
  <Memory units="MB">512</Memory>
</Configured>
```

Available Element

An available node property refers to a resource that is currently available for use.

- A compliant implementation SHOULD support this element.

The following is an example of specifying available properties:

```
<Available>
  <Processors>7</Processors>
  <Memory units="MB">256</Memory>
</Available>
```

Utilized Element

A utilized node property reflects resources that are currently utilized.

- A compliant implementation SHOULD support this element.

The following is an example of specifying utilized properties:

```
<Utilized>
  <Processors>8</Processors>
  <Memory metric="Average">207</Memory>
</Utilized>
```

4.3 Categorized Node Properties

Consumable Resources

Consumable Resources are a special group of node properties that can have additional attributes and can be used in multiple categories. In general a consumable resource is a resource that can be consumed in a measurable quantity.

- A consumable resource MUST be categorized as being a configured, available or utilized node property by being a child element of a Configured, Available or Utilized element respectively.
- A consumable resource MAY have a units attribute that is of type String that specifies the units by which it is being measured. If this attribute is omitted, a default unit is implied. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a metric attribute that is of type String that specifies the type of measurement being described. For example, the measurement may be a Total, an Average, a Min or a Max. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a wallDuration attribute of type Duration that indicates the amount of time for which that resource was used. This need only be specified if the resource was used for a different amount of time than the wallDuration for the step. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a consumptionRate attribute of type Float that indicates the average percentage that a resource was used over its wallDuration. For example, an overbooked SMP running 100 jobs across 32 processors may wish to scale the usage and charge by the average fraction of processor usage actually delivered. A compliant implementation MAY support this attribute if the element is supported.

A list of simple consumable resources is listed in the table below.

Table 3-14: Consumable Resource Node Properties

Element Name	Type	Description	Appearance	Compliance	Categories
Processors	Integer	Number of processors	MAY	MUST	CAU
Memory	Float	Amount of memory	MAY	SHOULD	CAU
Disk	Float	Amount of disk	MAY	SHOULD	CAU
Swap	Float	Amount of virtual memory	MAY	MAY	CAU
Network	Float	Amount of network	MAY	MAY	CAU

The following are two examples for specifying a consumable resource:

```
<Memory metric="Max" units="GB">483</Memory>
<Processors wallDuration="1234" consumptionRate="0.63">4</Processors>
```

Resource Element

In addition to the consumable resources enumerated in the above table, an extensible consumable resource is defined by the Resource element.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or more times within a given set of node properties.
- Like the other consumable resources, this property MUST be categorized as a configured, available or utilized property by being encompassed in the appropriate elements.
- This element is of type Float.
- It shares the other same properties and attributes as the other consumable resources but it requires an additional name (and optional type) attribute to describe it.
- This element MUST have a name attribute of type String that indicates the type of consumable resource being measured. A compliant implementation MUST support this attribute if the element is supported.
- This element MAY have a type attribute of type String that distinguishes it within a general resource class. A compliant implementation SHOULD support this attribute if the element is supported.

The following are two examples for specifying a `Resource` element:

```
<Resource name="License" type="MATLAB">1</Resource>
<Resource name="Telescope" type="Zoom2000" wallDuration="750"
metric="KX">10</Resource>
```

4.4 Node Reference

When a simple reference to a predefined node is needed in an encapsulating element, a Node element is used with the text content being the node id:

```
<Node>node1</Node>
```

- This element MAY have an aggregation attribute of type String that provides a way to indicate multiple values with a single expression. A compliant implementation MAY support the aggregation attribute if the Feature element is supported. Possible values for this attribute include:
  - List a comma-separated list of features
  - Pattern a regular expression (perl5) matching desired features
  - Range a range of nodes of the form: `<prefix>[5-23,77]`
- If an aggregation attribute is specified with the value of List, this element MAY also have a delimiter attribute of type String that indicates what delimiter is used to separate list elements. The default list delimiter is a comma.

- This element MAY have a count attribute of type Integer that indicates the instance count of the specified node(s).

The following is another example of a `Node` element:

```
<Node aggregation=”Pattern”>node[1-5]</Node>
```

Appendix A

Units of Measure Abbreviations

Abbreviation	Definition	Quantity
B	byte	1 byte
KB	Kilobyte	2^10 bytes
MB	Megabyte	2^20 bytes
GB	Gigabyte	2^30 bytes
TB	Terabyte	2^40 bytes
PB	Petabyte	2^50 bytes
EB	Exabyte	2^60 bytes
ZB	Aettabyte	2^70 bytes
YB	Yottabyte	2^80 bytes
NB	Nonabyte	2^90 bytes
DB	Doggabyte	2^100 bytes

Scalable Systems Software Resource Management and Accounting Protocol (SSSRMAP) Wire Protocol

Resource Management Interface Specs
Release v. 3.0.3
13 May 2004

Scott Jackson
Brett Bode

David Jackson
Kevin Walker

Status of This Memo

This is a specification defining a wire level protocol used between Scalable Systems Software components. It is intended that this specification will continue to evolve as these interfaces are implemented and thoroughly tested by time and experience.

Abstract

This document is a specification describing a connection-oriented XML-based application layer client-server protocol for the interaction of resource management and accounting software components developed as part of the Scalable Systems Software Center. The SSSRMAP Wire Protocol defines a framing protocol that includes provisions for security. The protocol is specified in XML Schema Definition and rides on the HTTP protocol.

Table of Contents

1.0 Introduction

A major objective of the Scalable Systems Software [SSS] Center is to create a scalable and modular infrastructure for resource management and accounting on terascale clusters including resource scheduling, grid-scheduling, node daemon support, comprehensive usage accounting and user interfaces emphasizing portability to terascale vendor operating systems. Existing resource management and accounting components feature disparate APIs (Application Programming Interfaces) requiring various forms of application coding to interact with other components.

This document proposes a wire level protocol expressed in an XML envelope to be considered as the foundation of a standard for communications between and among resource management and accounting software components. Individual components additionally need to define the particular XML binding necessary to represent the message format for communicating with the component.

2.0 Conventions Used in this Document

2.1 Keywords

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC2119.

2.2 XML Case Conventions

In order to enforce a consistent capitalization and naming convention across all SSSRMAP specifications "Upper Camel Case" (UCC) and "Lower Camel Case" (LCC) Capitalization styles shall be used. UCC style capitalizes the first character of each word and compounds the name. LCC style capitalizes the first character of each word except the first word. [XML_CONV][FED_XML]

1. SSSRMAP XML Schema and XML instance documents SHALL use the following conventions:
    - Element names SHALL be in UCC convention (example: `<UpperCamelCaseElement/>`.
    - Attribute names SHALL be in LCC convention (example: `<UpperCamelCaseElement lowerCamelCaseAttribute="Whatever"/>`.
2. General rules for all names are:
    - Acronyms SHOULD be avoided, but in cases where they are used, the capitalization SHALL remain (example: XMLSignature).
    - Underscores (_), periods (.) and dashes (-) MUST NOT be used (example: use JobId instead of JOB.ID, Job_ID or job-id).

2.3 Schema Definitions

```
SSSRMAP Schema Definitions appear like this
```

In case of disagreement between the schema file and this specification, the schema file takes precedence.

3.0 Encoding

Encoding tells how a message is represented when exchanged. SSSRMAP data exchange messages SHALL be defined in terms of XML schema [XML_SCHEMA].

3.1 Schema Header and Namespaces

The header of the schema definition is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<schema
  xmlns="http://www.w3.org/201/XMLSchema"
  xmlns:sssrmap="http://www.scidac.org/ScalableSystems/SSSRMAP"
  targetNamespace="http://www.scidac.org/ScalableSystems/SSSRMAP"
  elementFormDefault="qualified">
```

3.2 The Envelope Element

SSSRMAP messages and replies are encapsulated in the `Envelope` element. There are two possibilities for the contents of this element. If the contents are unencrypted, this element MUST contain a `Body` element and MAY contain a `Signature` element (refer to the section on Security). If the contents are encrypted, this element MUST contain exactly one `EncryptedData` element (refer to the section on Security). The `Envelope` element MAY contain namespace and other xsd-specific information necessary to validate the document against the schema. In addition, it MAY have any of the following attributes which may serve as processing clues to the parser:

Attribute	Description
type	A message type providing a hint as to the body contents such as “Request” or “Notification”
component	A component type such as “QueueManager” or “LocalScheduler”
name	A component name such as “OpenPBS” or “Maui”
version	A component version such as “2.2p12” or “3.2.2”

```
<complexType name=EnvelopeType">
  <choice minOccurs="1" maxOccurs="1">
    <choice minOccurs="1" maxOccurs="2">
      <element ref="sssrmap:Signature" minOccurs="0" maxOccurs="1"/>
      <element ref="sssrmap:Body" minOccurs="1" maxOccurs="1"/>
    </choice>
    <element ref="sssrmap:EncryptedData" minOccurs="1" maxOccurs="1"/>
  </choice>
  <attribute name="type" type="string" use="optional"/>
  <attribute name="component" type="string" use="optional"/>
  <attribute name="name" type="string" use="optional"/>
  <attribute name="version" type="string" use="optional"/>
</complexType>

<element name="Envelope" type="sssrmap:EnvelopeType"/>
```

3.3 The Body Element

- SSSRMAP messages and replies are encapsulated in the `Body` element. This element MUST contain exactly one `Request` or `Response` element.

```
<complexType name="BodyType">
  <choice minOccurs="1" maxOccurs="1">
    <element ref="sssrmap:Request" minOccurs="0" maxOccurs="1"/>
    <element ref="sssrmap:Response" minOccurs="0" maxOccurs="1"/>
    <any minOccurs="0" maxOccurs="1" namspace="##other"/>
  </choice>
</complexType>

<element name="Body" type="sssrmap:BodyType"/>
```

4.0 Transport Layer

This protocol will be built over the connection-oriented reliable transport layer TCP/IP. Support for other transport layers could also be considered, but native support for TCP/IP can be found on most terascale clusters and automatically handles issues such as reliability and connection fullness for the application developer implementing the SSSRMAP protocol.

5.0 Framing

Framing specifies how the beginning and ending of each message is delimited. Given that the encoding will be expressed as one or more XML documents, clients and servers need to know when an XML document has been fully read in order to be parsed and acted upon.

SSSRMAP uses the HTTP 1.1 [HTTP] protocol for framing. HTTP uses a byte-counting mechanism to delimit the message segments. HTTP chunked encoding is used. This allows for optional support for batched messages, large message segmentation and persistent connections.

5.1 Message Header Requirements

The HTTP request line (first line of the HTTP request header) begins with POST and is followed by a URI and the version of the HTTP protocol that the client understands. It is suggested for this protocol that the URI consist of a single slash, followed by the protocol name in uppercase (i.e. /SSSRMAP), though this field is not checked and could be empty, a single slash or any URI.

The Content-Type must be specified as test/xml. Charset may be optionally specified and defaults to US-ASCII. It is recommended that charset be specified as “utf-8” for maximum interoperability.

The Transfer-Encoding must be specified as chunked. The Content-Length must NOT be specified as the chunk size is specified in the message chunk.

Other properties such as User-Agent, Host and Date are strictly optional.

5.2 Message Chunk Format

A message chunk consists of a chunk size in hexadecimal format (whose value is the number of bytes in the XML message not including the chunk size and delimiter) delimited by a CR/LF “\r\n” and followed by the message payload in XML that consists of a single XML document having a root element of `Envelope`.

5.3 Reply Header Requirements

The HTTP response line (first line of the HTTP response header) begins with HTTP and a version number, followed by a numeric code and a message indicating what sort of response is made. These response codes and messages indicate the status of the entire response and are as defined by the HTTP standard. The most common response is 200 OK, indicating that the message was received and an appropriate response is being returned.

The Content-Type must be specified as text/xml. Charset may be optionally specified and defaults to US-ASCII. It is recommended that charset be specified as "utf-8" for maximum interoperability.

The Transfer-Encoding MUST be specified as chunked. The Content-Length must NOT be specified.

Other properties such as Server, Host and Date are strictly optional.

5.4 Reply Chunk Format

A reply chunk consists of a chunk size in hexadecimal format (whose value is the number of bytes in the XML reply not including the chunk size and delimiter) delimited by a CR/LF "\r\n" and followed by the reply payload in XML that consists of a single XML document having a root element of `Envelope`.

5.5 Message and Reply Tail Requirements and Multiple Chunks

This specification only requires that single chunks be supported. A server may optionally be configured to handle requests with persistent connections (multiple chunks). It will be the responsibility of clients to know whether a particular server supports this additional functionality. After all chunks have been sent, a connection is terminated by sending a zero followed by a carriage return-linefeed combination (0\r\n) and closing the connection.

5.6 Examples

Sample SSSRMAP Message Embedded in HTTP Request

```
POST /SSSRMAP HTTP/1.1\r\n
Content-Type: text/xml; charset="utf-8"\r\n
Transfer-Encoding: chunked\r\n
\r\n
9A\r\n
<Envelope …/>
0\r\n
```

Sample SSSRMAP Reply Embedded in HTTP Response

```
HTTP/1.1 200 OK\r\n
Content-Type: text/xml; charset="utf-8"\r\n
Transfer-Encoding: chunked\r\n
\r\n
2B4\r\n
<Envelope …/>
0\r\n
```

6.0 Asynchrony

Asynchrony (or multiplexing) allows for the handling of independent exchanges over the same connection. A widely-implemented approach is to allow pipelining (or boxcarring) by aggregating requests or responses within the body of the message or via persistent connections and chunking in HTTP 1.1. Pipelining helps reduce network latency by allowing a client to make multiple requests of a server, but requires the requests to be processed serially [RFC3117]. Parallelism could be employed to further reduce server latency by allowing multiple requests to be processed in parallel by multi-threaded applications.

Segmentation may become necessary if the messages are larger than the available window. With support for segmentation, the octet-counting requirement that you need to know the length of the whole message before sending it can be relegated to the segment level – and you can start sending segments before the whole message is available. Segmentation is facilitated via “chunking” in HTTP 1.1.

The current SSSRMAP strategy supports only a single request or response within the Body element. A server may optionally support persistent connections from a client via HTTP chunking. Segmentation of large responses is also optionally supported via HTTP chunking. Later versions of the protocol could allow pipelined requests and responses in a single Body element.

7.0 Security

SSSRMAP security features include capabilities for integrity, authentication, confidentiality, and non-repudiation. The absence or presence of the various security features depend upon the type of security token used and the protection methods you choose to specify in the request.

For compatibility reasons, SSSRMAP specifies six supported security token types. Extensibility features are included allowing an implementation to use alternate security algorithms and security tokens. It is also possible for an implementation to ignore security features if it is deemed nonessential for the component. However, it is highly RECOMMENDED that an implementation support at least the default security token type in both authentication and encryption.

7.1 Security Token

A security token may be included in either the Signature block, and/or in the EncryptedData block (both described later) as an implicit or explicit cryptographic key. If this element is omitted, the security token is assumed to be a secret key shared between the client and the server.

The SecurityToken Element

This element is of type String. If the security token conveys an explicit key, this element’s content is the value of the key. If the key is natively expressed in a binary form, it must be converted to base64 encoding as defined in XML Digital Signatures (“http://www.w3.org/2000/09/xmldsig#base64”). If the type is not specified, it is assumed to be of type “Symmetric”.

It may have any of the following optional attributes:

Attribute	Description
type	The type of security token (described subsequently) • A `type` attribute of "Symmetric" specifies a shared secret key between the client and server. This is the default. • A `type` attribute of "Asymmetric" specifies the use of public private key pairs between the client and server. • A `type` attribute of "Password" encrypts and authenticates with a user password known to both the client and server. • A `type` attribute of "Cleartext" allows the passing of a cleartext username and password and depends on the use of a secure transport (such as SSL or IPSec). • A `type` attribute of "Kerberos5" specifies a kerberos token. • A `type` attribute of "X509v3" specifies an X.509 certificate.
name	The name of the security token which serves as an identifier for the actor making the request (useful when the key is a password, or when the key value is implicit as when a public key is named but not included)

```
<complexType name="SecurityTokenType" mixed="true">
  <simpleContent>
    <extension base="string">
      <attribute name="type" type="string" use="optional">
      <attribute name="name" type="string" use="optional">
    </extension>
  </simpleContent>
</complexType>

<element name="SecurityToken" type="sssrmap:SecurityTokenType"/>
```

Security Token Types

SSSRMAP defines six standard security token types:

Symmetric Key

The default security token specifies the use of a shared secret key. The secret key is up to 128-bits long and known by both client and server. When using a symmetric key as a security token, it is not necessary to specify the `type` attribute with value "Symmetric" because this is assumed when the attribute is absent. The `name` attribute should be specified indicating the actor issuing the request. If the user provides a password to be sent to the server for authentication, then the password is encrypted with the secret key using a default `method="kw-tripledes"` (XML ENCRYPTION http://www.w3.org/2001/04/xmlenc#kw-tripledes), base64 encoded and included as the string content of the `SecurityToken` element. If the client authenticated the user, then the `SecurityToken` element is empty. The same symmetric key is used in both authentication and encryption.

Asymmetric Key

Public and private key pairs can be used to provide non-repudiation of the client (or server). The client and the server must each have their own asymmetric key pairs. This mode is indicated by specifying the type attribute as "Asymmetric". The `name` attribute should be specified indicating the actor issuing the request. If the user provides a password to be sent to the server for authentication, then the password is encrypted with the server's public key using a default `method="rsa-1_5"` (XML ENCRYPTION http://www.w3.org/2001/04/xmlenc#rsa-1_5), base64 encoded and included as the string content of the *SecurityToken* element. If the client authenticated the user, then the `SecurityToken` element is empty .The sender's private key is used in authentication (signing) while the recipient's public key is used for encryption.

Password

This mode allows for a username password combination to be used under the assumption that the server also knows the password for the user. This security token type is indicated by specifying a value of "Password" for the *type* attribute. The password itself is used as the cryptographic key for authentication and encryption. The *name* attribute contains the user name of the actor making the request. The *SecurityToken* element itself is empty.

Cleartext

This security mode is equivalent to passing the username and password in the clear and depends upon the use of a secure transport (such as SSL or IPSec). The purpose of including this security token type is to enable authentication to occur from web browsers over SSL or over internal LANs who use IPSec to encrypt all traffic. The password (or a hash of the password like in /etc/passwd) would have to be known by the server for authentication to occur. In this mode, neither encryption nor signing of the hash is performed at the application layer. This mode is indicated by specifying a value of "Cleartext" for the *type* attribute. The `name` attribute contains the user name of the actor making the request and the string content of the `SecurityToken` element is the unencrypted plaintext password.

Kerberos

The use of a Kerberos version 5 token is indicated by specifying "Kerberos5" in the `type` attribute. The `name` attribute is used to contain the kerberos user id of the actor making the request. The `SecurityToken` element contains two sub elements. The *Authenticator* element contains the authenticator encoded in base64. A `Ticket` element contains the service-granting ticket, also base64 encoded.

GSI (X.509)

The Grid Security Infrastructure (GSI) which is based on public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol can be indicated by specifying a `type` attribute of "X509v3". The name attribute contains the userid used that the actor was mapped to in the local system. The string content of the `SecurityToken` element is the GSI authentication message including the X.509 identity of the sender encoded in base64.

Example

```
<SecurityToken type="Asymmetric" name="scottmo">
MIIEZzCCA9CggAwIBAgIQEmtJZc0rqrKh5i...
</SecurityToken>
```

7.2 Authentication

Authentication entails how the peers at each end of the connection are identified and verified. Authentication is optional in an SSSRMAP message or reply. SSSRMAP uses a digital signature scheme for authentication that borrows from concepts in XML Digital Signatures [XML_DSIG]. In addition to authentication, the use of digital signatures also ensures integrity of the message, protecting exchanges from third-party modification.

When authentication is used, a Signature element is prepended as the first element within the Envelope element. All of the security modes will create a digest of the data for integrity checking and store this in base64 encoding in a DigestValue element as a child of the Signature element. The digital signature is created by encrypting the hash with the appropriate security token and storing this value in a SignatureValue element as a child of the Signature element. The security token itself is included as a child of the Security element within a SecurityToken element.

There are a number of procedural practices that must be followed in order to standardize this approach. The digest (or hash) is created over the contents of the Envelope element (not including the Element tag or its attributes). This might be over one or more Request or Notify elements (or Response or Ack elements) and necessarily excludes the Signature Element itself. (Note that any data encryption is performed after the creation of the digital signature and any decryption is performed before authenticating so the EncryptedData element will not interfere with this process. Hence, the signature is always based on the (hashed but) unencrypted data). For the purposes of generating the digest over the same value, it is assumed that the data is first canonicalized to remove extraneous whitespace, comments, etc according to the XML Digital Signature algorithm ("http://www.w3.org/TR/2001/REC-xml-c14n-20010315") and a transform is applied to remove namespace information. As a rule, any binary values are always transformed into their base64 encoded values when represented in XML.

The Signature Element

The Signature element MUST contain a DigestValue element that is used for integrity checking. It MUST also contain a SecurityToken element that is used to indicate the security mode and token type, and to verify the signature. It MUST contain a SignatureValue element that contains the base64 encrypted value of the signature wrought on the hash UNLESS the security token type indicates Cleartext mode where a signature would be of no value with the encryption key being sent in the clear -- in this case we use the password itself for authentication).

```
<complexType name="SignatureType">
  <choice minOccurs="2" maxOccurs="3">
    <element ref="sssrmap:DigestValue" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:SignatureValue" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:SecurityToken" minOccurs="0" maxOccurs="1"/>
  </choice>
</complexType>

<element name="Signature" type="sssrmap:SignatureType"/>
```

The DigestValue Element

The DigestValue element contains the cryptographic digest of the message data. As described above, the hash is generated over the Body element. The data to be hashed must first be canonicalized and appropriately transformed before generating the digest since typically an application will read in the XML document into an internal binary form, then marshal (or serialize) the data into a string which is passed as input to the hash algorithm. Different implementations marshal the data differently so it is necessary to convert this to a well-defined format before generating the digest or the clients will generate different digest values for the same XML. The SHA-1 [SHA-1] message digest algorithm (http://www.w3.org/2000/09/xmldsig#sha1) SHALL be used as the default method for generating the digest. A *method* attribute is defined as an extensibility option in case an implementation wants to be able to specify alternate message digest algorithms.

It MAY have a method attribute:

Attribute	Description
method	The message digest algorithm. • A `method` attribute of "sha1" specifies the SHA-1 message digest algorithm. This is the default and is implied if this attribute is omitted.

```
<complexType name="DigestValueType">
  <simpleContent>
    <extension base="string">
      <attribute name="method" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="DigestValue" type="sssrmap:DigestValueType"/>
```

The SignatureValue Element

The SignatureValue element contains the digital signature that serves the authentication (and potentially non-repudiation) function. The string content of the SignatureValue element is a base64 encoding of the encrypted digest value. The HMAC algorithm [HMAC] based on the SHA1 message digest (http://www.w3.org/2000/09/xmldsig#hmac-sha1) SHALL be used as the default message authentication code algorithm for user identification and message integrity. A `method` attribute is defined as an extensibility option in case an implementation wants to be able to specify alternate digital signature algorithms.

It MAY have a method attribute:

Attribute	Description
method	The digest signature algorithm. • A `method` attribute of "hmac-sha1" specifies the HMAC SHA-1 digital signature algorithm. This is the default and is implied if this attribute is omitted.

```
<complexType name="SignatureValueType">
  <simpleContent>
    <extension base="string">
      <attribute name="method" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="SignatureValue" type="sssrmap:SignatureValueType"/>
```

Signature Example

Pre-authentication:

```
<Envelope>
  <Body>
    <Request action="Query" actor="kenneth">
      <Object>User</Object>
      <Get name="EmailAddress"></Get>
      <Where name="Name">scott</Where>
    </Request>
  </Body>
</Envelope>
```

Post-authentication:

```
<Envelope>
  <Signature>
    <DigestValue>
      LyLsF0Pi4wPU...
    </DigestValue>
    <SignatureValue>
      DJbchm5gK...
    </SignatureValue>
    <SecurityToken type="Asymmetric" name="kenneth">
      MIIEZzCCA9CggAwIBAgIQEmtJZc0rqrKh5i...
    </SecurityToken>
  </Signature>
  <Body>
    <Request action="Query" actor="kenneth">
      <Object>User</Object>
      <Get name="EmailAddress"></Get>
      <Where name="Name">scottmo</Where>
    </Request>
  </Body>
</Envelope>
```

7.3 Confidentiality

Confidentiality involves encrypting the sensitive data in the message, protecting exchanges against third-party interception and modification. Confidentiality is optional in an SSSRMAP message or reply. When confidentiality is required, SSSRMAP sessions use block cipher encryption with concepts borrowed from the emerging XML Encryption [XML_ENC] standard.

When confidentiality is used, encryption is performed over all child elements of the `Envelope` element, i.e. on the message data as well as any signature (The encrypted data is not signed -- rather the signature is encrypted). This data is replaced in-place within the envelope with an `EncryptedData` element. The data is first compressed using the gzip algorithm [ZIP]. Instead of encrypting this compressed data with the security token directly, a 192-bit random session key is generated by the sender and used to perform symmetric encryption on the compressed data. This key is itself encrypted with the security token and included with the encrypted data as the value of the `EncryptedKey` element as a child of the `EncryptedData` element. The ciphertext resulting from the data being encrypted with the session key is passed as the value of a `CipherValue` element (also a child of the `EncryptedData` element). As in the case with authentication, the security token itself is included as a child of the `Security` element within a `SecurityToken` element.

The EncryptedData Element

When SSSRMAP confidentiality is required, the `EncryptedData` element MUST appear as the only child element in the Envelope element. It directly replaces the contents of these elements including the data and any digital signature. It MUST contain an `EncryptedKey` element that is used to encrypt the data. It MUST contain a `CipherValue` element that holds the base64 encoded ciphertext. It MAY also contain a `SecurityToken` element that is used to indicate the security mode and token type. If the `SecurityToken` element is omitted, a Symmetric key token type is assumed. Confidentiality is not used when a security token type of “Cleartext” is specified since it would be pointless to encrypt the data with the encryption key in the clear.

```
<complexType name="EncryptionDataType">
  <choice minOccurs="0" maxOccurs="1">
    <element ref="sssrmap:EncryptedKey" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:CipherValue" minOccurs="1" maxOccurs="1"/>
    <element ref="sssmap:SecurityToken" minOccurs="1" maxOccurs="1"/>
  </choice>
</complexType>

<element name="EncryptedData" type="sssrmap:EncryptedDataType"/>
```

The EncryptedKey Element

The `EncryptedKey` element is a random session key encrypted with the security token. This approach is used for a couple of reasons. In the case where public key encryption is used, asymmetric encryption is much slower than symmetric encryption and it makes sense to use a symmetric key for encryption and pass along it along by encrypting it with the recipient’s public key. It is also useful in that the security token which does not change very often (compared to the session key which changes for every connection) is used on a very small sampling of data (the session key), whereas if it was used to encrypt the whole message an attacker could more effectively exploit an attack against the ciphertext. The CMS Triple DES Key Wrap algorithm “kw-tripledes” SHALL be used as the default method for key encryption. The session key is encrypted using the security token, base64 encoded and specified as the string content of the `EncryptedKey` element. A `method` attribute is defined as an extensibility option in case an implementation wants to be able to specify alternate key encryption algorithms.

It is REQUIRED that an implementation use a cryptographically secure Pseudo-Random number generator. It is RECOMMENDED that the session key be cryptographically generated (such as cyclic encryption, DES OFB, ANSI X9.17 PRNG, SHA1PRNG, or ANSI X12.17 (used by PGP)).

It MAY have a method attribute:

Attribute	Description
method	The key encryption algorithm. • A `method` attribute of “kw-tripledes” specifies the CMS Triple DES Key Wrap algorithm. This algorithm is specified by the XML Encryption [XML_ENC] URI “`http://www.w3.org/2001/04/xmlenc#kw-tripledes`”. It involves two Triple DES encryptions, a random and known Initialization Vector (IV) and a CMS key checksum. A 192-bit key encryption key is generated from the security token, lengthened as necessary by zero-padding. No additional padding is performed in the encryptions. This is the default and is implied if this attribute is omitted.

```
<complexType name="EncryptedKeyType">
  <simpleContent>
    <extension base="string">
      <attribute name="method" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="EncryptedKey" type="sssrmap:EncryptedKeyType"/>
```

The CipherValue Element

The CipherValue element contains the message (and possibly signature) data encrypted with the random session key. The ciphertext is compressed using the gzip algorithm [ZIP], encrypted by the designated method, base64 encoded and included as the string content of the CipherValue element. The Triple DES algorithm with Cipher Block Chaining (CBC) feedback mode SHALL be used as the default method for encryption. A *method* attribute is defined as an extensibility option in case an implementation wants to be able to specify alternate data encryption algorithms.

It MAY have a method attribute:

Attribute	Description
method	The data encryption algorithm. • A `method` attribute of "tripledes-cbc" specifies the Triple DES algorithm with Cipher Block Chaining (CBC) feedback mode. This algorithm is specified by the XML Encryption [XML_ENC] URI identifier "http://www.w3.org/2001/04/xmlenc#tripledes-cbc". It specifies the use of a 192-bit encryption key and a 64-bit Initialization Vector (IV). Of the key bits, the first 64 are used in the first DES operation, the second 64 bits in the middle DES operation, and the third 64 bits in the last DES operation. The plaintext is first padded to a multiple of the block size (8 octets) using the padding scheme described in [XMLENC] for Block Encryption Algorithms (Padding per PKCS #5 will suffice for this). The resulting cipher text is prefixed by the IV. This is the default and is implied if this attribute is omitted.

```
<complexType name="CipherValueType">
  <simpleContent>
    <extension base="string">
      <attribute name="method" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="CipherValue" type="sssrmap:CipherValueType"/>
```

Encryption Example

In this example, a simple request is demonstrated without a digital signature for the sake of emphasizing the encryption plaintext replacement.

Pre-encryption:

```
<Envelope>
  <Body>
    <Response>
      <Status>true</Status>
      <Code>000</Code>
      <Count>1</Count>
      <Data>
        <User>
          <EmailAddress>Scott.Jackson@pnl.gov</EmailAddress>
        </User>
      </Data>
    </Response>
  </Body>
</Envelope>
```

Post-encryption:

```
<Envelope>
  <EncryptedData>
    <EncryptedKey>
      NAkE9iQofYhyOfiHZ29kkEFVJ30CAwEAAaMSM...
    </EncryptedKey>
    <CipherValue>
      mPCadVfOMx1NzDaKMHNgFkR9upTW4kgBxyPW...
    </CipherValue>
    <SecurityToken type="Asymmetric" name="kenneth">
      MIIEZzCCA9CggAwIBAgIQEmtJZc0rqrKh5i...
    </SecurityToken>
  </EncryptedData>
</Envelope>
```

8.0 Acknowledgements

9.0 References

[RFC2119] S. Bradner, "Key Words for Use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[BEEP] M. Rose, "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

[HMAC] H. Krawczyk, M. Bellare, R. Canetti, "HMAC, Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[SHA-1] U.S. Department of Commerce/National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-1.

[SSS] "Scalable Systems Software", http://www.scidac.org/ScalableSystems

[HTTP] "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616, June 1999.

[XML_CONV] "I-X and <I-N-CA> XML Conventions".

[FED_XML] "U.S. Federal XML Guidelines".

[RFC3117] M. Rose, "On the Design of Application Protocols", Informational RFC 3117, November 2001.

[XML_DSIG] D. Eastlake, J. Reagle Jr., D. Solo, "XML Signature Syntax and Processing", W3C Recommendation, 12 February 2002.

[XML_ENC] T. Imamura, B. Dillaway, E. Smon, “XML Encryption Syntax and Processing”, W3C Candidate Recommendation, 4 March 2002.

[XRP] E. Brunner-Williams, A. Damaraju, N. Zhang, “Extensible Registry Protocol (XRP)”, Internet Draft, expired August 2001.

[XML] Bray, T., et al, “Extensible Markup Language (XML) 1.0 (Second Edition)”, 6 October 2000.

[XML_SCHEMA] D. Beech, M. Maloney, N. Mendelshohn, “XML Schema Part 1: Structures Working Draft”, April 2000.

[ZIP] J. Gailly, M. Adler, “The gzip home page”, http://www.gzip.org/

Appendix W: Moab Resource Manager Language Interface Overview

The Moab RM Language (formerly called WIKI) is the language that some resource managers use to communicate with Moab, specifically a native RM. Generally each line represents a single resource or workload in Moab. The line contains the name of the resource or workload followed by a set of *<attr>=<val>* pairs. Although the Moab RM language follows the same data format for all RMs, each RM type receives and returns it differently. For instructions and examples on using Moab RM language with SLURM or a native RM, see **W.2 Managing Resources with SLURM** on page 1350 and **Managing Resources Directly with the Native Interface** on page 650 respectively.

- W.1 Moab Resource Manager Language Data Format
- W.2 Managing Resources with SLURM
- W.3 Moab RM Language Socket Protocol Description

W.1 Moab Resource Manager Language Data Format

- W.1.1 Query Resources Data Format
- W.1.2 Query Workload Data Format

W.1.1 Query Resources Data Format

NAME	FORMAT	DEFAULT	DESCRIPTION
ADISK	*<INTEGER>*	0	Available local disk on node (in MB)
AFS	`<fs id="X" size="X" io="Y" rcount="X" wcount="X" ocount="X"></fs>[...]`	0	Available filesystem state

NAME	FORMAT	DEFAULT	DESCRIPTION
AMEMORY	*<INTEGER>*	0	Available/free RAM on node (in MB)
APROC	*<INTEGER>*	1	Available processors on node
ARCH	*<STRING>*	---	Compute architecture of node
ARES	one or more comma delimited *<NAME>*:*<VALUE>* pairs (ie, `MATLAB:6,COMPILER:100`)	---	Arbitrary consumable resources currently available on the node
ASWAP	*<INTEGER>*	0	Available swap on node (in MB)
CCLASS	one or more bracket enclosed *<NAME>*:*<COUNT>* pairs (ie, `[batch:5][sge:3]`)	---	Run classes supported by node. Typically, one class is 'consumed' per task. Thus, an 8 processor node may have 8 instances of each class it supports present, ie `[batch:8][interactive:8]`
CDISK	*<INTEGER>*	0	Configured local disk on node (in MB)
CFS	*<STRING>*	0	Configured filesystem state
CMEMORY	*<INTEGER>*	0	Configured RAM on node (in MB)
CONTAINERNODE	*<STRING>*	---	The physical machine that is hosting the virtual machine. Only valid on VMs.
CPROC	*<INTEGER>*	1	Configured processors on node
CPULOAD	*<DOUBLE>*	0.0	One minute BSD load average
CPUSPEED	*<INTEGER>*	---	The node's processor speed in MHz

NAME	FORMAT	DEFAULT	DESCRIPTION
CRES	one or more comma delimited `<NAME>:<VALUE>` pairs (ie, `MATLAB:6,COMPILER:100`)	---	Arbitrary consumable resources supported and tracked on the node, ie software licenses or tape drives
CSWAP	*<INTEGER>*	0	Configured swap on node (in MB)
FEATURE	one or more colon delimited *<STRING>*'s (ie, `WIDE:HSM`)	---	Generic attributes, often describing hardware or software features, associated with the node
GEVENT	`GEVENT[<EVENTNAME>]=<STRING>`	---	Generic event occurrence and context data
GMETRIC	`GMETRIC[<METRICNAME>]=<DOUBLE>`	---	Current value of generic metric, i.e., '`GMETRIC[temp]=103.5`'.
IDLETIME	*<INTEGER>*	---	Number of seconds since last detected keyboard or mouse activity (often used with desktop harvesting)
MAXTASK	*<INTEGER>*	*<CPROC>*	Maximum number of tasks allowed on the node at any given time
NETADDR	*<STRING>*	---	The IP address of the machine
NODEINDEX	*<INTEGER>*	---	The node's index
OS	*<STRING>*	---	Operating system running on node
OSLIST	One or more comma delimited *<STRING>*'s with quotes if the string has spaces (ie. `"SAS7 AS3 Core Baseline Build v0.1.0","RedHat AS3-U5Development Build v0.2"`).	---	Operating systems accepted by node

NAME	FORMAT	DEFAULT	DESCRIPTION
OTHER	*<ATTR>*=*<VALUE>* [,*<ATTR>*=*<VALUE>*]...	---	Opaque node attributes assigned to node
PARTITION	*<STRING>*	DEFAULT	Partition to which node belongs
POWER	*<BOOLEAN>*		Whether the machine is on or off
PRIORITY	*<INTEGER>*	---	Node allocation priority
RACK	*<INTEGER>*	0	Rack location of the node
SLOT	*<INTEGER>*	0	Slot location of the node
STATE*	one of the following: `Idle`, `Running`, `Busy`, `Unknown`, `Drained`, `Draining`, or `Down`	Down	State of the node
UPDATETIME*	*<EPOCHTIME>*	0	Time node information was last updated
VARATTR	*<ATTR1>*=*<VAL1>*[=<displayName1>] [+*<ATTR2>*=*<VAL2>* [=<displayName2>]]...	---	Plus-delimited (+) list of *<ATTR>*=*<VAL>* [=<displayName>] pairs that jobs can request. You can replace any of the equals signs with colons if desired. Specifying a display name allows you to choose a name that will be displayed in the Mongo database instead of the unique ID (the <VALUE>). If you give two different attributes the same value and one of them also has a display name specified, both attributes will appear with the same display name.
VARIABLE	*<ATTR>*=*<VAL>*	---	Generic variables to be associated with node

NAME	FORMAT	DEFAULT	DESCRIPTION
VMOSLIST	*<STRING>*	---	Comma-delimited list (`,`) of supported virtual machine operating systems for this node
XRES	one or more comma delimited *<NAME>*`:`*<VALUE>* pairs (ie, `MATLAB:6,COMPILER:100`)	---	Amount of external usage of a particular generic resource

* indicates required field

Node states have the following definitions:

State	Description
Busy	Node is running some jobs and will not accept additional jobs
Down	Resource Manager problems have been detected. Node is incapable of running jobs.
Draining	Node is responding but will not accept new jobs
Idle	Node is ready to run jobs but currently is not running any.
Running	Node is running some jobs and will accept additional jobs
Unknown	Node is capable of running jobs but the scheduler will need to determine if the node state is actually Idle, Running, or Busy.

W.1.2 Query Workload Data Format

NAME	FORMAT	DEFAULT	DESCRIPTION
ACCOUNT	*<STRING>*	---	AccountID associated with job
ARGS	*<STRING>*	---	job command-line arguments
COMMENT	*<STRING>*	0	job resource manager extension arguments including qos, dependencies, reservation constraints, etc
COMPLETETIME*	*<EPOCHTIME>*	0	time job completed execution

NAME	FORMAT	DEFAULT	DESCRIPTION
DDISK	*<INTEGER>*	0	quantity of local disk space (in MB) which must be dedicated to each task of the job
DGRES	`name:value [,name:value]`	---	Dedicated generic resources per task.
DPROCS	*<INTEGER>*	1	number of processors dedicated per task
DSWAP	*<INTEGER>*	0	quantity of virtual memory (swap, in MB) which must be dedicated to each task of the job
ENDDATE	*<EPOCHTIME>*	[ANY]	time by which job must complete
ENV	*<STRING>*	---	job environment variables
ERROR	*<STRING>*	---	file to contain STDERR
EVENT	*<EVENT>*	---	event or exception experienced by job
EXEC	*<STRING>*	---	job executable command
EXITCODE	*<INTEGER>*	---	job exit code
FLAGS	*<STRING>*	---	job flags
GEOMETRY	*<STRING>*	---	String describing task geometry required by job
GNAME*	*<STRING>*	---	GroupID under which job will run

NAME	FORMAT	DEFAULT	DESCRIPTION
HOSTLIST	comma or colon delimited list of hostnames - suffix the hostlist with a carat (^) to mean superset; suffix with an asterisk (*) to mean subset; otherwise, the hostlist is interpreted as an exact set	`[ANY]`	list of required hosts on which job must run. (see TASKLIST) A subset means the specified hostlist is used first to select hosts for the job. If the job requires more hosts than are in the hostlist, they will be obtained from elsewhere if possible. If the job does not require all of the jobs in the hostlist, it will use only the ones it needs. A superset means the hostlist is the *only* source of hosts that should be considered for running the job. If the job can't find the necessary resources in the hosts in this list it should *not* run. No other hosts should be considered in allocating the job.
INPUT	*<STRING>*	---	file containing STDIN
IWD	*<STRING>*	---	job's initial working directory
NAME	*<STRING>*	---	User specified name of job
NODES	*<INTEGER>*	1	Number of nodes required by job (See Node Definition for more info)
OUTPUT	*<STRING>*	---	file to contain STDOUT
PARTITIONMASK	one or more colon delimited *<STRING>*s	`[ANY]`	list of partitions in which job can run
PREF	colon delimited list of *<STRING>*s	---	List of preferred node features or variables. (See PREF for more information.)
PRIORITY	*<INTEGER>*	---	system priority (absolute or relative - use '+' and '-' to specify relative)
QOS	*<INTEGER>*	0	quality of service requested
QUEUETIME*	*<EPOCHTIME>*	0	time job was submitted to resource manager
RARCH	*<STRING>*	---	architecture required by job

NAME	FORMAT	DEFAULT	DESCRIPTION
RCLASS	list of bracket enclosed *<STRING>* : *<INTEGER>* pairs	---	list of *<CLASSNAME>*:*<COUNT>* pairs indicating type and number of class instances required per task. (ie, `[batch:1]` or `[batch:2][tape:1]`)
RDISK	*<INTEGER>*	0	local disk space (in MB) required to be configured on nodes allocated to the job
RDISKCMP	one of >=, >, ==, <, or <=	>=	local disk comparison (ie, node must have > 2048 MB local disk)
REJCODE	*<INTEGER>*	0	reason job was rejected
REJCOUNT	*<INTEGER>*	0	number of times job was rejected
REJMESSAGE	*<STRING>*	---	text description of reason job was rejected
REQRSV	<STRING>	---	Name of reservation in which job must run
RESACCESS	<STRING>	---	List of reservations in which job can run
RFEATURES	colon delimited list <STRING>'s	---	List of features required on nodes
RMEM	<INTEGER>	0	real memory (RAM, in MB) required to be configured on nodes allocated to the job
RMEMCMP	one of '>=', '>', '==', '<', or '<='	>=	real memory comparison (ie, node must have >= 512MB RAM)
ROPSYS	<STRING>	---	operating system required by job
RSOFTWARE	<RESTYPE>[{+|:}<COUNT>][@<TIMEFRAME>]	---	software required by job
RSWAP	<INTEGER>	0	virtual memory (swap, in MB) required to be configured on nodes allocated to the job

NAME	FORMAT	DEFAULT	DESCRIPTION
RSWAPCMP	one of '>=', '>', '==', '<', or '<='	>=	virtual memory comparison (ie, node must have ==4096 MB virtual memory)
SID	<STRING>	---	system id (global job system owner)
STARTDATE	<EPOCHTIME>	0	earliest time job should be allowed to start
STARTTIME*	<EPOCHTIME>	0	time job was started by the resource manager
STATE*	one of Idle, Running, Hold, Suspended, Completed, or Removed	Idle	State of job
SUSPENDTIME	<INTEGER>	0	Number of seconds job has been suspended
TASKLIST	one or more comma-delimited <STRING>'s	---	list of **allocated** tasks, or in other words, comma-delimited list of node ID's associated with each active task of job (i.e., cl01, cl02, cl01, cl02, cl03) The tasklist is initially selected by the scheduler at the time the StartJob command is issued. The resource manager is then responsible for starting the job on these nodes and maintaining this task distribution information throughout the life of the job. (see HOSTLIST)
TASKS*	<INTEGER>	1	Number of tasks required by job (See Task Definition for more info)
TASKPERNODE	<INTEGER>	0	exact number of tasks required per node
UNAME*	<STRING>	---	UserID under which job will run
UPDATETIME*	<EPOCHTIME>	0	Time job was last updated
WCLIMIT*	[[HH:]MM:]SS	864000	walltime required by job

* indicates required field

Job states have the following definitions:

State	Definition
Completed	Job has completed
Hold	Job is in the queue but is not allowed to run
Idle	Job is ready to run
Removed	Job has been canceled or otherwise terminated externally
Running	Job is currently executing
Suspended	job has started but execution has temporarily been suspended

Completed and canceled jobs should be maintained by the resource manager for a brief time, perhaps 1 to 5 minutes, before being purged. This provides the scheduler time to obtain all final job state information for scheduler statistics.

Related topics

- Managing Resources with SLURM
- Managing Resources Directly with the Native Interface

W.2 Managing Resources with SLURM

This section demonstrates how Moab uses the Moab RM language (formerly called WIKI) to communicate with SLURM. For SLURM configuration instructions, see the Moab-SLURM Integration Guide.

W.2.1 Commands

All commands are requested via a socket interface, one command per socket connection. All fields and values are specified in ASCII text.

Supported Commands are:

- Query Resources
- Query Workload
- Start Job
- Cancel Job
- Suspend Job
- Resume Job
- Requeue Job
- JOBADDTASK
- JOBRELEASETASK

W.2.1.1 Moab RM Language Query Resources

W.2.1.1.1 Moab RM Language Query Resources Request Format

```
CMD=GETNODES ARG={<UPDATETIME>:<NODEID>[:<NODEID>]... | <UPDATETIME>:ALL}
```

Only nodes updated more recently than *<UPDATETIME>* will be returned where *<UPDATETIME>* is specified as the epoch time of interest. Setting *<UPDATETIME>* to `0` will return information for all nodes. Specify a colon delimited list of *NODEID*s if specific nodes are desired or use the keyword `ALL` to receive information for all nodes.

W.2.1.1.2 Moab RM Language Resources Response Format

The query resources response format is one or more line of the following format (separated with a new line):

```
<NODEID><ATTR>=<VALUE>[;<ATTR>=<VALUE>]...
```

<ATTR> is one of the names in the table below and the format of *<VALUE>* is dependent on *<ATTR>*.

Example 3-184: Moab RM language resource query and response

Request:

```
CMD=GETNODES ARG=0:node001:node002:node003
```

Response:

```
node001 UPDATETIME=963004212;STATE=Busy;OS=AIX43;ARCH=RS6000...
node002 UPDATETIME=963004213;STATE=Busy;OS=AIX43;ARCH=RS6000...
...
```

W.2.1.2 Moab RM Language Query Workload

W.2.1.2.1 Moab RM Language Query Workload Request Format

`CMD=GETJOBS ARG={<UPDATETIME>:<JOBID>[:<JOBID>]... | <UPDATETIME>:ALL }`

Only jobs updated more recently than *<UPDATETIME>* will be returned where *<UPDATETIME>* is specified as the epoch time of interest. Setting *<UPDATETIME>* to `0` will return information for all jobs. Specify a colon delimited list of JOBID's if information for specific jobs is desired or use the keyword `ALL` to receive information about all jobs.

W.2.1.2.2 Moab RM Language Query Workload Response Format

`SC=<STATUSCODE> ARG=<JOBCOUNT>#<JOBID>:<FIELD>=<VALUE>;[<FIELD>=<VALUE>;]... [#<JOBID>:<FIELD>=<VALUE>;[<FIELD>=<VALUE>;]...]...`

or

`SC=<STATUSCODE> RESPONSE=<RESPONSE>`

FIELD is either the text name listed below or `A<FIELDNUM>`
(ie, `UPDATETIME` or `A2`)

STATUSCODE values:

- `0 SUCCESS`
- `-1 INTERNAL ERROR`

RESPONSE is a statuscode sensitive message describing error or state details.

W.2.1.2.3 Moab RM Language Query Workload Example

Request:

```
CMD=GETJOBS ARG=0:ALL
```

Response:

```
ARG=2#nebo3001.0:UPDATETIME=9780000320;STATE=Idle;WCLIMIT=3600;...
```

W.2.1.3 StartJob

The StartJob command may only be applied to jobs in the Idle state. It causes the job to begin running using the resources listed in the NodeID list.

send CMD=STARTJOB ARG=<*JOBID*> TASKLIST=<*NODEID*>[:<*NODEID*>]...

receive SC=<*STATUSCODE*> RESPONSE=<*RESPONSE*>

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message possibly further describing an error or state

Example 3-185: Job start

```
# Start job nebo.1 on nodes cluster001 and cluster002

# send
CMD=STARTJOB ARG=nebo.1 TASKLIST=cluster001:cluster002
# receive
SC=0;RESPONSE=job nebo.1 started with 2 tasks
```

W.2.1.4 CancelJob

The CancelJob command, if applied to an active job, will terminate its execution. If applied to an idle or active job, the CancelJob command will change the job's state to Canceled.

send CMD=CANCELJOB ARG=<*JOBID*> TYPE=<*CANCELTYPE*>

<*CANCELTYPE*> is one of the following:

ADMIN (command initiated by scheduler administrator)
WALLCLOCK (command initiated by scheduler because job exceeded its specified wallclock limit)

receive SC=<*STATUSCODE*> RESPONSE=<*RESPONSE*>

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message further describing an error or state

Example 3-186: Job cancel

```
# Cancel job nebo.2

# send
CMD=CANCELJOB ARG=nebo.2 TYPE=ADMIN'
# receive
SC=0 RESPONSE=job nebo.2 canceled
```

W.2.1.5 SuspendJob

The SuspendJob command can only be issued against a job in the state Running. This command suspends job execution and results in the job changing to the Suspended state.

send CMD=SUSPENDJOB ARG=<*JOBID*>

receive SC=<*STATUSCODE*> RESPONSE=<*RESPONSE*>

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message possibly further describing an error or state

Example 3-187: Job suspend

```
# Suspend job nebo.3

# send
CMD=SUSPENDJOB ARG=nebo.3
# receive
SC=0 RESPONSE=job nebo.3 suspended
```

W.2.1.6 ResumeJob

The ResumeJob command can only be issued against a job in the state `Suspended`. This command resumes a suspended job returning it to the `Running` state.

send `CMD=RESUMEJOB ARG=<JOBID>`

receive `SC=<STATUSCODE> RESPONSE=<RESPONSE>`

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message further describing an error or state

Example 3-188: Job resume

```
# Resume job nebo.3

# send
CMD=RESUMEJOB ARG=nebo.3
# receive
SC=0 RESPONSE=job nebo.3 resumed
```

W.2.1.7 RequeueJob

The RequeueJob command can only be issued against an active job in the state `Starting` or `Running`. This command requeues the job, stopping execution and returning the job to an idle state in the queue. The requeued job will be eligible for execution the next time resources are available.

send `CMD=REQUEUEJOB ARG=<JOBID>`

receive `SC=<STATUSCODE> RESPONSE=<RESPONSE>`

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message further describing an error or state

Example 3-189: job requeue

```
# Requeue job nebo.3

# send
CMD=REQUEUEJOB ARG=nebo.3
# receive
SC=0 RESPONSE=job nebo.3 requeued
```

W.2.1.8 SignalJob

The SignalJob command can only be issued against an active job in the state `Starting` or `Running`. This command signals the job, sending the specified signal to the master process. The signaled job will be remain in the same state it was before the signal was issued.

send `CMD=SIGNALJOB ARG=<JOBID> ACTION=signal VALUE=<SIGNAL>`

receive `SC=<STATUSCODE> RESPONSE=<RESPONSE>`

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message further describing an error or state

Example 3-190: Job signal

```
# Signal job nebo.3

# send
CMD=SIGNALJOB ARG=nebo.3 ACTION=signal VALUE=13
# receive
SC=0 RESPONSE=job nebo.3 signaled
```

W.2.1.9 ModifyJob

The ModifyJob command can be issued against any active or queued job. This command modifies specified attributes of the job.

send `CMD=MODIFYJOB ARG=<JOBID> [BANK=name] [NODES=num] [PARTITION=name] [TIMELIMIT=minutes]`

receive `SC=<STATUSCODE> RESPONSE=<RESPONSE>`

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message further describing an error or state

Example 3-191: Job modify

```
# Signal job nebo.3

# send
CMD=MODIFYJOB ARG=nebo.3 TIMELIMIT=9600
# receive
SC=0 RESPONSE=job nebo.3 modified
```

W.2.1.10 JobAddTask

The JobAddTask command allocates additional tasks to an active job.

send CMD=JOBADDTASK ARG=<*JOBID*> <*NODEID*> [<*NODEID*>]...

receive SC=<*STATUSCODE*> RESPONSE=<*RESPONSE*>

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message possibly further describing an error or state

Example 3-192: Job addtask

```
# Add 3 default tasks to job nebo30023.0 using resources located on nodes cluster002,
cluster016, and cluster112.

# send
CMD=JOBADDTASK ARG=nebo30023.0 DEFAULT cluster002 cluster016 cluster112
# receive
SC=0 RESPONSE=3 tasks added
```

W.2.1.11 JobRemoveTask

The JobRemoveTask command removes tasks from an active job.

send CMD=JOBREMOVETASK ARG=<*JOBID*> <*TASKID*> [<*TASKID*>]...

receive SC=<*STATUSCODE*> RESPONSE=<*RESPONSE*>

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message further describing an error or state

Example 3-193: Job removetask

```
# Free resources allocated to tasks 14, 15, and 16 of job nebo30023.0

# send
CMD=JOBREMOVETASK ARG=nebo30023.0 14 15 16
# receive
SC=0 RESPONSE=3 tasks removed
```

W.2.2 Rejection Codes

- 0xx - success - no error
  - 00x - success
    - 000 - success
  - 01x - usage/help reply
    - 010 - usage/help reply
  - 02x - status reply
    - 020 - general status reply

- 1xx - warning
  - 10x - general warning
    - 100 - general warning
  - 11x - no content
    - 110 - general wire protocol or network warning
    - 112 - redirect
    - 114 - protocol warning
  - 12x - no matching results
    - 120 - general message format warning
    - 122 - incomplete specification (best guess action/response applied)
  - 13x - security warning
    - 130 - general security warning
    - 132 - insecure request
    - 134 - insufficient privileges (response was censored/action reduced in scope)
  - 14x - content or action warning
    - 140 - general content/action warning
    - 142 - no content (server has processed the request but there is no data to be returned)
    - 144 - no action (no object to act upon)
    - 146 - partial content
    - 148 - partial action
  - 15x - component defined
  - 18x - application defined
- 2xx - wire protocol/network failure
  - 20x - protocol failure
    - 200 - general protocol/network failure
  - 21x - network failure
    - 210 - general network failure
    - 212 - cannot resolve host
    - 214 - cannot resolve port

    - 216 - cannot create socket
    - 218 - cannot bind socket
  - 22x - connection failure
    - 220 - general connection failure
    - 222 - cannot connect to service
    - 224 - cannot send data
    - 226 - cannot receive data
  - 23x - connection rejected
    - 230 - general connection failure
    - 232 - connection timed-out
    - 234 - connection rejected - too busy
    - 236 - connection rejected - message too big
  - 24x - malformed framing
    - 240 - general framing failure
    - 242 - malformed framing protocol
    - 244 - invalid message size
    - 246 - unexpected end of file
  - 25x - component defined
  - 28x - application defined
- 3xx - messaging format error
  - 30x - general messaging format error
    - 300 - general messaging format error
  - 31x - malformed XML document
    - 310 - general malformed XML error
  - 32x - XML schema validation error
    - 320 - general XML schema validation
  - 33x - general syntax error in request
    - 330 - general syntax error in response
    - 332 - object incorrectly specified
    - 334 - action incorrectly specified
    - 336 - option/parameter incorrectly specified

    - 34x - general syntax error in response
        - 340 - general response syntax error
        - 342 - object incorrectly specified
        - 344 - action incorrectly specified
        - 346 - option/parameter incorrectly specified
    - 35x - synchronization failure
        - 350 - general synchronization failure
        - 352 - request identifier is not unique
        - 354 - request id values do not match
        - 356 - request id count does not match
- 4xx - security error occurred
    - 40x - authentication failure - client signature
        - 400 - general client signature failure
        - 402 - invalid authentication type
        - 404 - cannot generate security token key - inadequate information
        - 406 - cannot canonicalize request
        - 408 - cannot sign request
    - 41x - negotiation failure
        - 410 - general negotiation failure
        - 412 - negotiation request malformed
        - 414 - negotiation request not understood
        - 416 - negotiation request not supported
    - 42x - authentication failure
        - 420 - general authentication failure
        - 422 - client signature failure
        - 424 - server authentication failure
        - 426 - server signature failure
        - 428 - client authentication failure
    - 43x - encryption failure
        - 430 - general encryption failure
        - 432 - client encryption failure

        - 434 - server decryption failure
        - 436 - server encryption failure
        - 438 - client decryption failure
    - 44x - authorization failure
        - 440 - general authorization failure
        - 442 - client authorization failure
        - 444 - server authorization failure
    - 45x - component defined failure
    - 48x - application defined failure
- 5xx - event management request failure
    - 50x - reserved
        - 500 - reserved
- 6xx - reserved for future use
    - 60x - reserved
        - 600 - reserved
- 7xx - server side error occurred
    - 70x - server side error
        - 700 - general server side error
    - 71x - server does not support requested function
        - 710 - server does not support requested function
    - 72x - internal server error
        - 720 - general internal server error
    - 73x - resource unavailable
        - 730 - general resource unavailable error
        - 732 - software resource unavailable error
        - 734 - hardware resource unavailable error
    - 74x - request violates policy
        - 740 - general policy violation
    - 75x - component-defined failure
    - 78x - application-defined failure

- 8xx - client side error occurred
  - 80x - general client side error
    - 800 - general client side error
  - 81x - request not supported
    - 810 - request not supported
  - 82x - application specific failure
    - 820 - general application specific failure
- 9xx - miscellaneous
  - 90x - general miscellaneous error
    - 900 - general miscellaneous error
  - 91x - general insufficient resources error
    - 910 - general insufficient resources error
  - 99x - general unknown error
    - 999 - unknown error

Related topics

- Moab Resource Manager Language Data Format
- Managing Resources Directly with the Native Interface

W.3 Moab RM Language Socket Protocol Description

Moab RM language is formerly known as WIKI. The Moab scheduler uses a simple protocol for socket connections to the user client and the resource manager as described below:

`<SIZE><CHAR>CK=<CKSUM><WS>TS=<TIMESTAMP><WS>AUTH=<AUTH><WS>DT=<DATA>`

Attribute	Description
<SIZE>	8 character decimal ASCII representation of the size of the packet following '*<SIZE><CHAR>*' Leading zeroes must be used to pad this value to 8 characters if necessary.
<CHAR>	A single ASCII character
<CKSUM>	A 16 character hexadecimal ASCII DES-based checksum calculated using the algorithm below* and *<SEED>* selected and kept secret by the site admins. The checksum is performed on the line from *TS=* to the end of the message including *<DATA>*.
<WS>	a series of white space characters consisting of either tabs and/or space characters.

Attribute	Description
<TIMESTAMP>	ASCII representation of epoch time
<AUTH>	Identifier of user requesting service (i.e., `USERNAME`)
<DT>	Data to be sent

An example header follows:

```
00001057 CK=cdf6d7a7ad45026f TS=922401962 AUTH=sched DT=<DATA>
```

where *<DATA>* is replaced by actual message data.

W.3.1 Checksum Algorithm ('C' version)

```
#define MAX_CKSUM_ITERATION 4

int GetChecksum(
 char *Buf,
 int BufSize,
 char *Checksum,
 char *CSKey) /* Note: pass in secret key */
 {
 unsigned int crc;
 unsigned int lword;
 unsigned int irword;
 int index;
 unsigned int Seed;
 Seed = (unsigned int)strtoul(CSKey,NULL,0);
 crc = 0;
 for (index = 0;index < BufSize;index++)
 {
 crc = (unsigned int)DoCRC((unsigned short)crc,Buf[index]);
 }
 lword = crc;
 irword = Seed;
 PSDES(&lword,&irword);
 sprintf(Checksum,"%08x%08x",
 lword,
 irword);
 return(SUCCESS);
 }


unsigned short DoCRC(
 unsigned short crc,
 unsigned char onech)
 {
 int index;
 unsigned int ans;
 ans = (crc ^ onech << 8);
 for (index = 0;index < 8;index++)
 {
 if (ans & 0x8000)
 ans = (ans <<= 1) ^ 4129;
 else
 ans <<= 1;
 }
 return((unsigned short)ans);
 }


int PSDES(
 unsigned int *lword,
 unsigned int *irword)
 {
 int index;
 unsigned int ia;
 unsigned int ib;
 unsigned int iswap;
 unsigned int itmph;
 unsigned int itmpl;
 static unsigned int c1[MAX_CKSUM_ITERATION] = {
 0xcba4e531, 0x537158eb, 0x145cdc3c, 0x0d3fdeb2 };
 static unsigned int c2[MAX_CKSUM_ITERATION] = {
 0x12be4590, 0xab54ce58, 0x6954c7a6, 0x15a2ca46 };
 itmph = 0;
```

```
itmpl = 0;
for (index = 0;index < MAX_CKSUM_ITERATION;index++)
{
iswap = *irword;
ia = iswap ^ c1[index];
itmpl = ia & 0xffff;
itmph = ia >> 16;
ib = (itmpl * itmpl) + ~(itmph*itmph);
ia = (ib >> 16) | ((ib & 0xffff) << 16);
*irword = (*lword) ^ ((ia ^ c2[index]) + (itmpl * itmph));
*lword = iswap;
}
return(SUCCESS);
}
```

W.3.2 Header Creation (PERL code)

(taken from PNNL's QBank client code)

```
################################################################################
#
# subroutine wiki($COMMAND)
#
# Sends command to Moab server and returns the parsed result and status
#
################################################################################
sub wiki
{
 my($COMMAND,$REQUEST,$result);
 my($sockaddr,$hostname);
 my($name,$aliases,$proto,$port,$type,$len,$thisaddr);
 my($thisport,$thatport,$response,$result);
 $COMMAND = shift;
 #
 # Establish socket connection
 #
 $sockaddr = 'S n a4 x8';
 chop ($hostname = `hostname`);
 ($name,$aliases,$proto)=getprotobyname('tcp');
 ($name,$aliases,$type,$len,$thisaddr)=gethostbyname($hostname);
 ($name,$aliases,$type,$len,$thataddr)=gethostbyname($BANKHOST);
 $thisport=pack($sockaddr, &AF_INET,0,$thisaddr);
 $thatport=pack($sockaddr, &AF_INET,$BANKPORT,$thataddr);
 socket(S, &PF_INET,&SOCK_STREAM,$proto) || die "cannot create socket\n";
 bind(S,$thisport) || die "cannot bind socket\n";
 connect(S,$thatport) || die "cannot connect socket\n";
 select(S); $| = 1; # Turn on autoflushing
 select(stdout); $| = 1; # Select STDOUT as default output
 #
 # Build and send command
 #
 $REQUEST="COMMAND=$COMMAND AUTH=$AUTH";
 chomp($CHECKSUM = `$QSUM "$REQUEST"`);
 $REQUEST .= " CHECKSUM=$CHECKSUM";
 my $command=pack "a8 a1 A*",sprintf("%08d",length($REQUEST))," ",$REQUEST;
 print S "$command"; # Send Command to server
 @REPLY=();
 while () { push(@REPLY,$_); } # Listen for Reply
 $STATUS=grep(/STATUSCODE=(\d*)/&&$1,@REPLY); # STATUSCODE stored in $STATUS
 grep(s/.*RESULT=//,@REPLY); # Parse out the RESULT
 return @REPLY;
}
```

W.3.3 Header Processing (PERL code)

```
sysread(NS,$length,8); # Read length string
sysread(NS,$delimiter,1); # Read delimiter byte
$DEBUG && print STDERR "length=[$length]\tdelimiter=[$delimiter]\n";
while($length) {
 $DEBUG && print STDERR "Awaiting $length bytes -- ".`date`;
 $length-=sysread(NS,$request,$length); # Read request
 sleep 1;
 }
%REQUEST=();
chomp($request);
foreach (@REQUEST=&shellwords($request)) # Parse arguments into array
 {
 ($key,$value)=split(/=/,$_);
 $REQUEST{$key}=$value unless defined $REQUEST{$key};
 }
$request =~ s/\s+CHECKSUM=.*//; # Strip off the checksum
print STDERR "REQUEST=$request\n";
chomp($checksum=`$QSUM "$request"`);
$me=$REQUEST{AUTH};
$command=$REQUEST{COMMAND};
if (!grep($command eq $_,@VALIDCMDS))
 { $REPLY = "STATUSCODE=0 RESULT=$command is not a valid command\n";}
elsif ($checksum ne $REQUEST{CHECKSUM})
 { $REPLY = "STATUSCODE=0 RESULT=Invalid Checksum\n";}
else
 { $REPLY = do $command(@REQUEST); }
$len=sprintf("%08d",length($REPLY)-1);
$delim=' ';
$DEBUG && print STDERR "REPLY=${len}${delim}$REPLY\n";
$buf="$len"."$delim"."$REPLY";
syswrite(NS,$buf,length($buf));
close NS;
```

SCHEDCFG flags

Flag	Description
AGGREGATENODEFEATURES	*AGGREGATENODEFEATURES* causes Moab to aggregate features reported by the different RMs. For example, if you have two RMs reporting different features for the same node, Moab will add both features together (instead of one being overwritten by the other). In order to set features manually, you can use mnodectl -m features (for details, see **mnodectl** on page 233).
ALLOWINFINITEJOBS	*ALLOWINFINITEJOBS* allows infinite wallclock times to be accepted. Previously, jobs with infinite job times were allowed by default.

Flag	Description
ALLOWMULTICOMPUTE	*ALLOWMULTICOMPUTE* tells Moab how to resolve conflicting information from different resource managers. If *ALLOWMULTICOMPUTE* is specified, Moab will use the STATE and OS information from the resource manager that reports the node as online.
CANCELFAILEDDEPENDENCYJOBS	Automatically cancels dependency jobs that will never run because of an unmet requirement. For example, if you ran a job with both an afterok and afternotok job attached to it and that job was successful, the afterok job would run, leaving the afternotok job idle in the queue. If you set *CANCELFAILEDDEPENDENCYJOBS*, Moab will cancel the job with the failed dependency and remove it from the queue. For more information about job dependencies, see **Job Dependencies** on page 529.
DISABLEPERJOBNODESETS	Disables a job's ability to override the system specified node set. See 13.3 Resource Manager Extensions for more information.
DISABLEPARTIALNODERESERVATIONS	Blocks partial node reservations.
ENABLESLURMMEMPERCPU	By default Moab calls sbatch or srun with a `--mem=` request in a SLURM environment. When you set *ENABLESLURMMEMPERCPU*, Moab instead calls `--mem-per-cpu=`. This is to allow sites with policies that require the other parameter to use `--mem-per-cpu`.
ENFORCERESERVEDNODES	Without this flag Moab tries to optimize the reservation for a job before it starts, meaning a job may start on nodes that weren't part of its reservation. With this flag Moab tries to start jobs only on the nodes that were reserved.
ENFORCESAMENODESET	The same nodeset is not enforced across job requirements by default, rather each requirement is scheduled separately and the nodesets are determined on a per-req basis. To have Moab enforce the same nodeset across all job requirements set this flag.
EXTENDEDGROUPSUPPORT	Allows Moab to consider a user's secondary Linux groups when dealing with reservation ACLs.
FASTGROUPLOOKUP	Moab will use the system call `getgrouplist` to gather group information. This can significantly improve performance on some LDAP systems.

Flag	Description
FASTRSVSTARTUP	Speeds up start time if there are existing reservations. *FASTRSVSTARTUP* is incompatible with partial node reservations. On very large systems, if there is a reservation in the checkpoint file on all the nodes, it would take a really long time for Moab to start up. For every node in the reservation, Moab checks every other node. With this flag, Moab just uses the nodelist that was checkpointed to create the reservation. It speeds up the startup process because it doesn't have to check every node. Where Moab would take 8 - 10 minutes to start up with an 18,000 node reservation without the flag, Moab can start up in 2-3 minutes with the flag. With the flag you will see one difference in checknode. A reservation that uses all the procs on a node initially shows that all the procs are blocked. Without the flag, and as jobs fill on the node, the blocked resources will be configured - dedicated (ex. 5/6). With the flag, the blocked resources will always be what the reservation is blocking and won't change when jobs fill on the node. Without flag: Reservations: brian.1x1 User -00:12:52 -> INFINITY (INFINITY) Blocked Resources@-00:00:02 Procs: 5/6 (83.33%) Mem: 0/5000 (0.00%) Blocked Resources@00:04:58 Procs: 6/6 (100.00%) Mem: 0/5000 (0.00%) m.2x1 Job:Running -00:00:02 -> 00:04:58 (00:05:00) Jobs: m.2 With flag: Reservations: brian.1x1 User -00:00:15 -> INFINITY (INFINITY) Blocked Resources@-00:00:02 Procs: 6/6 (100.00%) Mem: 0/5000 (0.00%) Blocked Resources@00:04:58 Procs: 6/6 (100.00%) Mem: 0/5000 (0.00%) m.1x1 Job:Running -00:00:02 -> 00:04:58 (00:05:00) Jobs: m.1 When you set the FASTRVSSTARTUP flag, Moab will also set the DISABLEPARTIALNODERESERVATIONS flag.
FILELOCKHA	This is a High Availability feature. *FILELOCKHA* prevents scheduling conflicts between multiple Moab servers.

<table>
<tr><th>Flag</th><th>Description</th></tr>
<tr><td>FREECOMPLETEDJOBSUBMITSTRING</td><td>Moab frees the job submit string for completed jobs, decreasing the amount of memory needed during operation. This is useful in environments with large job scripts that can create a large memory footprint.</td></tr>
<tr><td>IGNOREPIDFILELOCK</td><td>Moab will not fail if it cannot get a lock on the <code>.moab.pid</code> file. This is useful when Moab is running on a shared filesystem where file locking can be unpredictable.</td></tr>
<tr><td>JOBSUSERSVWALLTIME</td><td>Allows jobs submitted without a walltime request or default walltime received from a class or queue but with an <code>ADVRES:reservation</code> to inherit their walltime limit from the reservation instead of the Moab default. The job walltime limit is then the remaining time of the reservation to which the job was submitted.</td></tr>
<tr><td>NOCLASSUPDATE</td><td>While running against TORQUE, Moab will not update classes when it refreshes each iteration. Moab loads the classes at startup, but does not refresh them until the next time it is restarted.</td></tr>
<tr><td>NORMALIZETASKDEFINITIONS</td><td>Instructs Moab to normalize all tasks that it receives via an mshow -a command. Moab normalizes the task definition to one processor and then changes the tasks requested to the number of processors requested. For example, when the following is received by Moab:
<code>mshow -a -w mintasks=1@procs:4+mem:4096</code>
It is changed to this:
<code>mshow -a -w mintasks=4@procs:1+,mem:1024,tpn=4</code></td></tr>
<tr><td>OPTIMIZEDBACKFILL</td><td>On large systems that utilize system-wide reservations, backfill can take a considerable amount of time. This flag speeds up backfill scheduling by using an alternative BETA backfill algorithm. This flag will be the default in future versions of Moab.</td></tr>
</table>

Flag	Description
PRIORITYPOLICYBLOCKING	By default, a job that violates a policy is placed into the blocked queue. Jobs with a lower priority, but that do not violate the policy, will run. This can lead to situations in which small jobs starve out larger, higher priority jobs. When you set the *PRIORITYPOLICYBLOCKING* flag, Moab allows the job that violates the policy to continue consuming the policy slots while it remains blocked. With the policy slots consumed, the smaller, lower priority jobs will not run. The higher priority job will continue to consume the policy slots until it has consumed enough to actually run. Note that because the blocked job consumes policy slots, this will inevitably lead to lower system utilization.
SHOWREQUESTEDPROCS	Shows requested processors regardless of NodeAccessPolicy in showq. When *SINGLEJOB* **NODEACCESSPOLICY** is used and the job requests one processor, showq displays the job with one processor.
SHOWUSERJOBSONLY	Causes Moab, when a non-admin user runs showq, to return only that user's jobs. If an administrator runs showq when this flag is set, Moab returns the jobs of all users; no restrictions are placed on administrators.
STRICTSPOOLDIRPERMISSIONS	Enforces at least a 511 permission on the Moab spool directory.
UNMIGRATEONDEFER	Forces Moab to unmigrate a job in a grid if it enters a deferred state.

Moab Web Services

Moab Web Services overview

Moab Web Services (MWS) is a component of Adaptive Computing Suites that enables programmatic interaction with Moab Workload Manager via a RESTful interface. MWS lets you create and interact with Moab objects and properties such as jobs, nodes, virtual machines, and reservations. MWS is the preferred method for those wishing to create custom user interfaces for Moab and is the primary method by which Moab Viewpoint communicates with Moab.

MWS communicates with the Moab Workload Manager (Moab) server using the same wire protocol as the Moab command-line interface. By publishing a standard interface into Moab's intelligence, MWS significantly reduces the amount of work required to integrate Moab into your solution.

This documentation is intended for developers performing such integrations. If you are a Moab administrator, and for conceptual information about Moab, see **Moab Workload Manager overview** on page 97.

Setup

Moab Web Services setup

This section explains what you need to know in order to get MWS configured, and secured correctly. It contains the following topics:

- **Configuring Moab Web Services** on page 1373
- **Setting up MWS security** on page 1388
- **Version and build information** on page 1396

Related topics

- **Moab Web Services overview** on page 1373
- **Access control** on page 1398

Configuring Moab Web Services

This section describes the location of the MWS configuration files. It also shows some examples of how to configure logging.

> To see a full reference to all configuration and logging parameters available in MWS, see **Configuration** on page 1750.

This topic contains these sections:

Home directory

The MWS home directory contains configuration files, log files, and files that serve features of MWS such as hooks and plugins. You should set the location of the MWS home directory using the **MWS_HOME** property. If you do not set **MWS_HOME** as a Java property or as an environment variable, then MWS will use `/opt/mws` as the default **MWS_HOME**.

Configuration files

The primary configuration file is `MWS_HOME/etc/mws-config.groovy`. If this file is missing or contains errors, MWS will not start.

Configuration files can also be placed in the `MWS_HOME/etc/mws.d directory`. Any configuration files here get merged with `MWS_HOME/etc/mws-config.groovy`. In case of conflict, the configuration in `MWS_HOME/etc/mws.d` takes precedence.

If `MWS_HOME/etc/log4j.properties` exists, MWS will load it as well.

Logging configuration using `mws-config.groovy`

Shown below is an example that logs all error messages and fatal messages to `/opt/mws/log/mws.log` (For information about the format of the MWS logs, see "Standard Log Format" in the *Moab Workload Manager Administrator Guide*.). It also logs all stack traces to `/opt/mws/log/stacktrace.log`. Note that this example is not configured to log events; for details on logging events, see **Configuring an event log** on page 1375.

```
Minimal logging configuration
-----------------------------

log4j = {
   appenders {
      rollingFile name: 'stacktrace',
         file: '/opt/mws/log/stacktrace.log',
         maxFileSize: '1GB'
      rollingFile name: 'rootLog',
         file: '/opt/mws/log/mws.log',
         threshold: org.apache.log4j.Level.ERROR,
         maxFileSize: '1GB'
   }
   root {
      debug 'rootLog'
   }
}
```

Alternatively, you may configure a console appender instead of a rolling file, as shown below.

```
Console logging configuration
-----------------------------

log4j = {
   appenders {
      rollingFile name: 'stacktrace',
         file: '/opt/mws/log/stacktrace.log',
         maxFileSize: '1GB'
      console name: 'consoleLog',
         threshold: org.apache.log4j.Level.ERROR
   }
   root {
      debug 'consoleLog'
   }
}
```

You may configure logging by using either `MWS_HOME/etc/mws-config.groovy` or `MWS_HOME/etc/log4j.properties`.

If you do not define any `log4j` configuration, MWS will write its log files to `java.io.tmpdir`. For Tomcat, `java.io.tmpdir` is generally set to `$CATALINA_BASE/temp` or `CATALINA_TMPDIR`.

Configuring an event log

Logging events to a flat file requires that you make a few changes to the configuration in the **log4j** section of the `mws-config.groovy` file so that events will be logged to the `events.log` file, and all other MWS logging information will be sent to the `mws.log` file.

Causing events.log to roll based on a time window

You can specify how often the `events.log` file rolls. The following example illustrates the configuration changes you will need make to `mws-config.groovy` to cause the `events.log` file to roll based on a time window. Note the following three examples:

- In this example, `mws-config.groovy` is configured so that `events.log` rolls daily at midnight.

```
Daily rolling events.log configuration in mws-config.groovy
------------------------------------

log4j = {
  def eventAppender = new org.apache.log4j.rolling.RollingFileAppender(name:
'events', layout: pattern(conversionPattern: "%m%n"))
  def rollingPolicy = new org.apache.log4j.rolling.TimeBasedRollingPolicy
(fileNamePattern: '/tmp/events.%d{yyyy-MM-dd}', activeFileName:
'/tmp/events.log')
  rollingPolicy.activateOptions()
  eventAppender.setRollingPolicy(rollingPolicy)

  appenders {
    appender eventAppender

    rollingFile name: 'rootLog',
      file: '/tmp/mws.log',
      maxFileSize: '1GB'
  }

  root {
    warn 'rootLog'
  }

  trace additivity:false, events:'com.ace.mws.events.EventFlatFileWriter'
}
```

  Note the **RollingFileAppender** and the **TimeBasedRollingPolicy** lines. These lines configure MWS to write the event log to the `events.log` file. Rolled log files will have a date appended to their name in this format: "yyyy-MM-dd" (for example, `events.log.2012-02-28`).

- If you want the event log file to roll at the beginning of each month, change the **fileNamePattern** TimeBasedRollingPolicy date format to `yyyy-MM`. For example:

```
Monthly event logs
------------------------------------

def rollingPolicy = new org.apache.log4j.rolling.TimeBasedRollingPolicy
(fileNamePattern: '/tmp/events.%d{yyyy-MM}', activeFileName: '/tmp/events.log')
```

- If you want the event log file to roll at the beginning of each hour, change the date format to `yyyy-MM-dd_HH:00`. For example:

```
Hourly event logs
------------------------------------

def rollingPolicy = new org.apache.log4j.rolling.TimeBasedRollingPolicy
(fileNamePattern: '/tmp/events.%d{yyyy-MM-dd_HH:00}', activeFileName:
'/tmp/events.log')
```

Configuring events.log to roll based on a file size threshold

You can also configure the events.log file to roll when the log size exceeds a specified threshold. The following example illustrates the configuration changes you will need to make to `mws-config.groovy` to cause the `events.log` file to roll on a size threshold. (In this example, `mws-config.groovy` is configured so that `events.log rolls` when its size exceeds 50 MB.)

```
mws-config.groovy configuration that rolls events.log based on file size
------------------------------------

log4j = {
  appenders {
    rollingFile name: 'events',
      file: '/tmp/events.log',
      maxFileSize: '50MB',
      maxBackupIndex:10

    rollingFile name: 'rootLog',
      file: '/tmp/mws.log',
      maxFileSize: '1GB'
  }

  root {
    warn 'rootLog'
  }

  trace additivity:false, events:'com.ace.mws.events.EventFlatFileWriter'
}
```

Note that **maxFileSize** is set to "50MB." This means that when the `events.log` file exceeds 50 MB, it will roll.

The name for the rolled log will be "events.log.1". When the *new* events.log file exceeds 50 MB, *it* will roll and be named "events.log.1", while the old "events.log.1" file will be renamed "events.log.2". This process will continue until the optional **maxBackupIndex** value is met. In the example above, **maxBackIndex** is set to 10. This means that MWS will delete all but the ten most recent `events.log` files. Using this feature helps prevent hard drives from filling up.

Additivity

The **additivity** attribute of the EventFlatFileWriter logger can be either `true` or `false`. If you specify `true`, events will be logged to the `events.log` file *and* the `mws.log` file. If you specify `false`, events will be logged to the `events.log` file only. (All other MWS logging information will be logged to the `mws.log` file, as configured by the `rootLog` appender.)

To log events to the `mws.log` file in addition to the `events.log` file, make the `additivity:true` configuration. For example:

```
Logging events to both events.log and mws.log
------------------------------------

trace additivity:true, events:'com.ace.mws.events.EventFlatFileWriter'
```

For more configuration options, see Apache Extras Companion for log4j.

Deleting old events

If your MongoDB server is version 2.2 or later, MongoDB will automatically delete events older than 30 days (by default). For more information, including how to change this default, see `mws.events.expireAfterSeconds` in **Configuration** on page 1750.

If your MongoDB server is older than version 2.2, MongoDB will store event data indefinitely. However, if disk space is limited, you may want to regularly delete old, unneeded events from MongoDB. This section contains some examples of how you can do this.

Let's say that you want to delete events that are older than 90 days. (There are 86,400,000 milliseconds in a day, so in this example, 90*86400000 corresponds to 90 days in milliseconds.):

- You could run this script:

```
Delete events older than 90 days
------------------------------------

$ mongo
MongoDB shell version: 2.4.8
connecting to: test
> use mws
> db.event.remove({eventTime:{$lt:new Date(new Date().getTime()-90*86400000)}})
> exit
```

- To create a script to perform this task:

```
deleteOldEvents.sh
------------------------------------

#!/bin/bash
printf 'use mws_dev\ndb.event.remove({eventTime:{$lt:new Date(new Date().getTime
()-90*86400000)}})\nexit' | mongo
```

- Now say that you want to set up a cron job (`$crontab -e`) so that old events are automatically deleted on a certain day of the week (for example, every Sunday at 2:00 a.m.), you would add an entry like this:

```
cron table entry to delete old events
------------------------------------

00 02 * * 0 /root/deleteOldEvents.sh
```

Configuring an audit trail log

Audit logging enables you to track changes to **Permissions** on page 1571, **Roles** on page 1633, and **Principals** on page 1605.

```
mws-config.groovy configuration that enables audit logging
------------------------------------

	def auditAppender = new org.apache.log4j.rolling.RollingFileAppender(
			name: 'audit',
			layout: new com.ace.mws.logging.ACPatternLayout("%j\t\t\t%c{1}\t\t\t%m%n"))
	def auditRollingPolicy = new org.apache.log4j.rolling.TimeBasedRollingPolicy(
			fileNamePattern: '/opt/mws/log/audit.%d{yyyy-MM-dd}',
			activeFileName: '/opt/mws/log/audit.log')
	auditRollingPolicy.activateOptions()
	auditAppender.setRollingPolicy(auditRollingPolicy)

	appenders {
		rollingFile name: 'stacktrace',
				file: '/opt/mws/log/stacktrace.log',
				maxFileSize: '100MB'
		rollingFile name: 'rootLog',
				file: '/opt/mws/log/mws.log',
				maxFileSize: '100MB', //The maximum file size for a single log file
				maxBackupIndex: 10, //Retain only the 10 most recent log files, delet
to save space
				layout: new com.ace.mws.logging.ACPatternLayout(), //Configures the c
of each log entry
				threshold: org.apache.log4j.Level.ERROR //Ignore any logging entries
than this threshold

		appender eventAppender
		appender auditAppender
	}
```

You can customize audit logging in ways you can customize event logging. For example, you can specify how often the `audit.log` file rolls. You can also configure the `audit.log` file to roll when the log size exceeds a specified threshold.

> Follow the same steps indicated in the previous section on **Configuring an event log** on page 1375 for instruction on customizing audit logging; customization processes are the same for audit logging as for events logging.

audit.log file format

The default location to which the audit trail log is written is `/opt/mws/log/audit.log`. The log format is `yyyy-MM-dd HH:mm:ss resource username action data`. The following table offers a description for attributes included in the log format:

Parameter	Description
resource	The resource--permission, role, principal, or tenant--that changed.
username	The user's user name.
action	The type of change (create, update, or delete).
data	Dependent on what changed.

Sample `audit.log` format:

```
Audit trail log format
-----------------------------------

2013-10-30 14:39:32,120 TENANT 'admin' updated resource named 'Engineering2' with
values:
     "name": "Engineering3",
     "attachedPrincipals": [{"name": "Engineering"}]
```

LDAP Configuration using `mws-config.groovy`

Using a supported LDAP directory type

To configure an MWS connection to an LDAP server, add the following parameters to `mws-config.groovy`:

> Throughout the following examples in this topic, you will see **dc=acme,dc=com**. "acme" is only used as an example to illustrate what you would use as your own domain controller if your domain name was "acme.com." You should replace any references to "acme" with your own organization's domain name.

Parameter	Description
ldap.server	The hostname or IP address of the LDAP server.
ldap.port	The port the LDAP server is listening on.
ldap.baseDNs	A list of distinguished names that are the root entries for LDAP searches.
ldap.bindUser	The distinguished name of the bind user.
ldap.password	The password of the ldap.bindUser.
ldap.directory.type	The type of LDAP directory (e.g. "Microsoft Active Directory"). This parameter can have the following values: • Microsoft Active Directory • OpenLDAP Using InetOrgPerson Schema • OpenLDAP Using NIS Schema • OpenLDAP Using Samba Schema

Here is a sample configuration for OpenLDAP.

> If you followed the Adaptive Computing tutorial [link]"Setting up OpenLDAP on CentOS 6" your **ldap.directory.type** should be set to "OpenLDAP Using InetOrgPerson Schema".

```
Sample OpenLDAP configuration
-----------------------------

ldap.server  = "192.168.0.5"
ldap.port = 389
ldap.baseDNs = ["dc=acme,dc=com"]
ldap.bindUser = "cn=Manager,dc=acme,dc=com"
ldap.password = "*****"
ldap.directory.type = "OpenLDAP Using InetOrgPerson Schema"
```

Here is a sample configuration for Microsoft Active Directory.

```
Sample Active Directory configuration
-------------------------------------

ldap.server  = "192.168.0.5"
ldap.port = 389
ldap.baseDNs = ["CN=Users,DC=acme,DC=com","OU=Europe,DC=acme,DC=com"]
ldap.bindUser = "cn=Administrator,cn=Users,DC=acme,DC=com"
ldap.password = "*****"
ldap.directory.type = "Microsoft Active Directory"
```

To see how to configure a secure connection to the LDAP server, see **Securing the LDAP connection** on page 1394.

Using an unsupported LDAP directory type

If you are not using one of the supported directory types, you can explicitly configure MWS to work with your LDAP schema by using the following parameters:

Parameter	Description
ldap.user.objectClass	The name of the class used for the LDAP user object. For example: • user • person • inetOrgPerson • posixAccount
ldap.group.objectClass	The name of the class used for the LDAP group object. For example: • group • groupOfNames • posixGroup
ldap.ou.objectClass	The name of the class used for the LDAP organizational unit object. for example: • organizationalUnit

Parameter	Description
ldap.user.membership.attribute	The attribute field in a user entry to use when loading the user's groups (optional if **ldap.group.membership.attribute** is defined). For example: • memberOf
ldap.group.membership.attribute	The attribute field in a group entry to use when loading the group's members (optional if **ldap.user.membership.attribute** is defined). For example: • member • memberUid
ldap.user.name.attribute	The attribute field to use when loading the username. This field must uniquely identify a user. For example: • sAMAccountName • uid

For example:

```
Advanced Active Directory configuration
-----------------------------

ldap.server  = "myldaphostname"
ldap.port = 389
ldap.baseDNs = ["CN=Users,DC=acme,DC=com","OU=Europe,DC=acme,DC=com"]
ldap.bindUser = "cn=Administrator,cn=Users,DC=acme,DC=com"
ldap.password = "*****"
ldap.user.objectClass = "person"
ldap.group.objectClass = "group"
ldap.ou.objectClass = "organizationalUnit"
ldap.user.membership.attribute = "memberof"
ldap.group.membership.attribute = "member"
ldap.user.name.attribute = "sAMAccountName"
```

Here is a similar example for OpenLDAP. Note there is no user membership attribute in the OpenLDAP InetOrgPerson schema and thus **ldap.user.membership.attribute** is set to null. This is allowable because the **ldap.group.membership.attribute** is set.

```
Advanced OpenLDAP configuration
-----------------------------

ldap.server  = "myldaphostname"
ldap.port = 389
ldap.baseDNs = ["dc=acme,dc=com"]
ldap.bindUser = "cn=Manager,dc=acme,dc=com"
ldap.password = "*****"
ldap.user.objectClass = "inetOrgPerson"
ldap.group.objectClass = "groupOfNames"
ldap.ou.objectClass = "organizationalUnit"
ldap.user.membership.attribute = null
ldap.group.membership.attribute = "memberUid"
ldap.user.name.attribute = "uid"
```

Overriding attributes in a supported LDAP directory type

You can also override attributes in supported directory types. For example, say you are using OpenLDAP with an NIS Schema. The group objectClass for NIS defaults to "groupOfNames," but you want to use "groupOfUniqueNames" instead while retaining all other defaults for NIS. You can do this by setting **ldap.directory.type** to "OpenLDAP Using NIS Schema" and overriding the **ldap.group.objectClass** attribute as follows:

```
Advanced OpenLDAP configuration
-----------------------------

ldap.directory.type = "OpenLDAP Using NIS Schema"
ldap.group.objectClass = "groupOfUniqueNames"
```

> LDAP is *not* currently used to authenticate users to MWS. LDAP is only used to map principals to roles, as explained in **Principals** on page 1605.
>
> The user class in your LDAP schema must have an attribute that uniquely identifies a user (for example: "uid" or "sAMAccountName").

PAM (pluggable authentication module) configuration using `mws-config.groovy`

PAM functions as bridge to the underlying Unix authentication system. PAM treats the user as if it is local to the Unix machine doing the authenticating and uses whatever the Unix user is authenticating with, whether it be LDAP or NIS. PAM uses configuration files that specify the how, when, or what for authentication, session management, and account management. Each configuration file can be different. For example, `sudo` configuration file for the "sudo" command will handle authentication differently than the `login` configuration file. These configuration files are dynamically read for `/etc/pam.d`.

Requirements for PAM

In order to use PAM with MWS, the following is required:

- The PAM application package must be installed. For example:

  ```
  yum install pam
  ```

- You must have a PAM configuration file in the `/etc/pam.d` directory. The following is an example of what a PAM configuration file might look like:

```
#%PAM-1.0
auth        required      pam_env.so
auth        sufficient    pam_unix.so nullok try_first_pass
auth        requisite     pam_succeed_if.so uid >= 1000 quiet_success
auth        required      pam_deny.so


account     required      pam_unix.so
account     sufficient    pam_localuser.so
account     sufficient    pam_succeed_if.so uid < 1000 quiet
account     required      pam_permit.so


password    requisite     pam_pwquality.so try_first_pass retry=3 authtok_type=
password    sufficient    pam_unix.so sha512 shadow nullok try_first_pass use_
authtok
password    required      pam_deny.so


session     optional      pam_keyinit.so revoke
session     required      pam_limits.so
-session     optional      pam_systemd.so
session     [success=1 default=ignore] pam_succeed_if.so service in crond quiet
use_uid
session     required      pam_unix.so
```

- (Optional) You must have PAM modules installed for your specific needs.

  The PAM application comes with default modules—for example, **pam_unix.xo**—that will check username and password credentials with Unix. You may have to install others for your distribution.

Configuring MWS to use PAM

To configure an MWS connection to PAM, add the following parameter to `mws-config.groovy`:

Parameter	Description
pam.configuration.service	The name of the PAM configuration file located in `/etc/pam.d`. This parameter and specification tells MWS which PAM configuration file you want to use.

For example:

```
pam.configuration.service = "system-auth"
```

You can configure only one authentication method in `mws-config.groovy`—LDAP or PAM, but not both. If you have configured both LDAP and PAM, MWS defaults to using LDAP.

If you need multiple authentication methods, you must add them to your local PAM configuration. See your distribution documentation for details.

> There is a security risk when authenticating local users through your PAM configuration. This behavior is highly discouraged and not supported by Adaptive Computing.

For more information about PAM, please see the following SLES and RedHat documentation.

OAuth configuration using `mws-config.groovy`

OAuth is a security framework designed to simplify authentication in web technologies. In the case of MWS, OAuth allows trusted client applications to securely delegate authentication to MWS. Once MWS has authenticated a user by verifying the username and password in LDAP, PAM, or NIS, MWS returns an access token to the client. The client then presents this access token to MWS to access resources. OAuth is very flexible and allows MWS to work in many different scenarios by use of grant types. For more information on OAuth and grant types, please see the following OAuth documentation.

Example using 'password' grant type

Terminology

Resource Owner: The person accessing and manipulating data. For MWS, this would be the person who logs into the client (the user).

Service Provider: The site or service where protected resources live. This can be (but is not necessarily) also the identify provider, where usernames and passwords are stored. This is the MWS service itself.

Client: The application that wants to access a resource. For MWS this is the user interface, potentially including APIs and command-line tools.

Protected Resource: The data for which protection is desired. For MWS this would be Moab itself, and interaction with Moab.

Access Token: Instead of user credentials, OAuth uses tokens to issue requests, and the tokens get signed to indicate authorization.

Register a client in MWS

Oauth requires client registration. Its client credentials are used to validate that the client is allowed to authenticate on behalf of a resource owner. It involves giving the client its own credentials (username and password). MWS will first authenticate the client using a client id (username) and client secret (password), then will authenticate the resource owner.

Add the following line to /opt/mws/etc/mws-config.groovy:

```
grails.plugin.springsecurity.oauthProvider.clients = [
      [
      clientId:"THE_CLIENT_ID",
      clientSecret:"THE_CLIENT_SECRET",
      authorizedGrantTypes:["password"]
      ]
]
```

Replace THE_CLIENT_ID with client id (username). For example: clientId:"iris". Also, replace THE_CLIENT_SECRET with client secret (password). For example: clientSecret:"irisclientpassword",. Note that the values for clientId and clientSecret are case sensitive.

You can register more than one client. For example:

```
grails.plugin.springsecurity.oauthProvider.clients = [
      [
      clientId:"client_id_1",
      clientSecret:"client_secret_1",
      authorizedGrantTypes:["password"]
      ],
      [
      clientId:"client_id_2",
      clientSecret:"client_secret_1",
      authorizedGrantTypes:["password"]
      ]
]
```

Obtain an access token from MWS for a resource owner (logging in)

Before the client can access private data in MWS, the client must obtain an access token that grants access to the API. The token endpoint url is only used to gain an access token and log in a user.

Getting an access token:

```
POST http://localhost:8080/mws/rest/oauth/token?api-version=3
Adding header:
      "Content-Type: application/x-www-form-urlencoded"
Request body (String):
grant_type=password&client_id=THE_CLIENT_ID&client_secret=THE_CLIENT_
SECRET&username=RESOURCE_OWNER_USERNAME&password=RESOURCE_OWNER_PASSWORD
```

Example using curl:

```
curl -X POST -H "Content-Type: application/x-www-form-urlencoded" -v -d 'grant_
type=password&client_id=iris&client_secret=irisclientpassword&username=moab-
admin&password=secret' 'http://localhost:8080/mws/oauth/token'
```

Produces the following response:

```
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
> POST /mws/oauth/token HTTP/1.1
> User-Agent: curl/7.19.7 (x86_64-redhat-linux-gnu) libcurl/7.19.7 NSS/3.14.0.0
zlib/1.2.3 libidn/1.18 libssh2/1.4.2
> Host: localhost:8080
> Accept: */*
> Content-Type: application/x-www-form-urlencoded
> Content-Length: 126
>
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Cache-Control: no-store
< Pragma: no-cache
< Set-Cookie: JSESSIONID=6CE8F9E7C454575FABCF3D156B153CFD; Path=/mws
< Content-Type: application/json;charset=UTF-8
< Transfer-Encoding: chunked
< Date: Fri, 18 May 2014 18:16:42 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
{"access_token":"b693eec0-6c93-4540-8b2f-1e170be08046","token_type":"bearer","expires_
in":43096}
```

Send the access token to MWS when requesting protected resource

After the client obtains an access token, it will send the access token to MWS in an HTTP authorization header for each rest call.

The client is responsible for handling user sessions with each access token, meaning the client has to request a new access token when a new user logs in.

Requesting an MWS resource (getting list of all nodes for example):

```
GET http://localhost:8080/mws/rest/nodes?api-version=3&fields=name
Adding authorization header:
     "Authorization: Bearer ACCESS_TOKEN"
```

Example using curl:

```
curl -X GET -H "Authorization: Bearer b693eec0-6c93-4540-8b2f-1e170be08046" -v
'http://localhost:8080/mws/rest/nodes?api-version=3&fields=name'
```

Produces the following response:

```
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /mws/rest/nodes?api-version=3&fields=name HTTP/1.1
> User-Agent: curl/7.19.7 (x86_64-redhat-linux-gnu) libcurl/7.19.7 NSS/3.14.0.0
zlib/1.2.3 libidn/1.18 libssh2/1.4.2
> Host: localhost:8080
> Accept: */*
> Authorization: Bearer b693eec0-6c93-4540-8b2f-1e170be08046
>
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/json;charset=UTF-8
< Pragma: no-cache
< Set-Cookie: JSESSIONID=6CE8F9E7C454575FABCF3D156B153CFD; Path=/mws
< Content-Type: application/json;charset=UTF-8
< Content-Language: en-US
< Transfer-Encoding: chunked
< Date: Fri, 18 May 2014 18:39:07 GMT
<
{"totalCount":3,"resultCount":3,"results":[{"name":"node1"},{"name":"node2"},
{"name":"node3"}]}
```

Related topics

- **Setting up MWS security** on page 1388
- **Version and build information** on page 1396

Setting up MWS security

When running MWSin production environments, security is a major concern. This section focuses on securing the connections with MWS:

- The connection between MWS and Moab Workload Manager (see **Securing the connection with Moab** on page 1389).
- The connection between MWS and MongoDB (see **Securing the connection with MongoDB** on page 1389).
- The connections between clients and MWS (see **Securing client connections to MWS** on page 1390).
- The connection between MWS and LDAP (see **Securing the LDAP connection** on page 1394).
- The connection with the message queue (see **Securing the connection with the message queue** on page 1395).

Related topics

- **Configuring Moab Web Services** on page 1373
- **Version and build information** on page 1396

Securing the connection with Moab

MWS communicates with Moab via the Moab Wire Protocol, which uses a direct connection between the two applications. The communication over this connection uses a shared secret key, which is discussed in the installation instructions (see "**Installing Moab Web Services** on page 38" in the installation documentation). However, the communication is not encrypted and is therefore susceptible to eavesdropping and replay attacks. For this reason, MWS is supported only when running on the same machine as Moab. This assures that any connections between the two applications occur internally on the server and are not exposed to external users.

Related topics

- **Setting up MWS security** on page 1388

Securing the connection with MongoDB

By default, the connection between MWS and MongoDB is not authenticated. To enable authentication, follow the instructions below. For further reading, see the MongoDB tutorial "Control Access to MongoDB Instances with Authentication."

To enable an authenticated connection between MWS and MongoDB

1. Add an administrative user to the `admin` database.
2. Add an MWS user to the `mws` database.
3. To support MWS API version 2, add an MWS user with "read-only" rights to the `moab` database.

   Here is an example of how to create all the required users. The users in the `moab` database are required only for MWS API version 2.

   ```
   [root]# service mongod start
   [root]# mongo
   > use admin;
   > db.addUser("admin_user", "secret1");
   > use moab;
   > db.addUser("moab_user", "secret2");
   > db.addUser("mws_user", "secret3", true);
   > use mws;
   > db.addUser("mws_user", "secret3");
   > exit;
   ```

   The passwords used here ("secret1," "secret2," and "secret3") are examples. Choose your own passwords for these users.

4. Add the MWS user credentials (the ones you just created) to the `mws-config.groovy` file. For example:

   ```
   grails.mongo.username = "mws_user"
   grails.mongo.password = "secret3"
   ```

5. Enable authentication in the MongoDB configuration file (called `/etc/mongodb.conf` on many Linux distributions). In that file, look for **#auth = true** and uncomment it.
6. Restart MongoDB.
7. Restart Tomcat.

If authentication is enabled in MongoDB, but the MWS user was not properly created or configured, MWS will not start. In this case, see the log file(s) for additional information.

Related topics

- **Setting up MWS security** on page 1388

Securing client connections to MWS

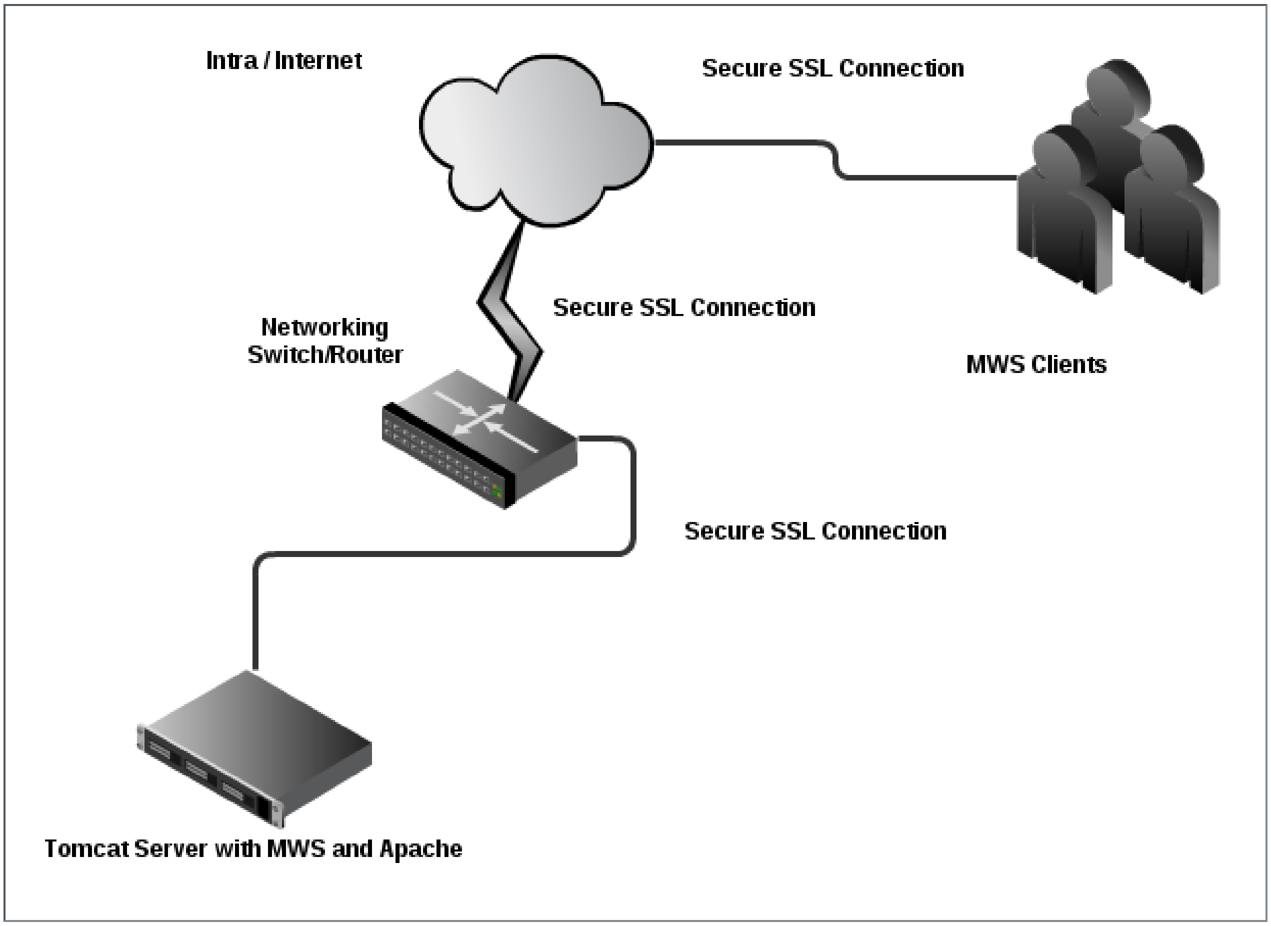
All connections to MWS, except those requesting the documentation or the main page, must be authenticated properly. MWS uses a single-trusted-user authentication model, meaning a single user exists that has access to all aspects of MWS. The username and password for this user are configured with the `auth.defaultUser` properties in the configuration file. For more information, see **Configuration** on page 1750.

When using the MWS user interface in a browser, the user will be prompted for username and password. For information on how to authenticate requests when not using a browser, see **Authentication** on page 1421.

> The username and password in the Basic Authentication header are encoded but not encrypted. Therefore, it is *strongly* recommended that MWS be run behind a proxy (like Apache) with SSL enabled. The instructions below provide an example of how to do this.

Encrypting client connections using Apache and SSL

This section shows how to encrypt client connections to MWS using Apache and SSL. These instructions have been tested on CentOS™ 6.2 with the "Web Server" software set installed. The same ideas are applicable to other operating systems, but the details might be different. As shown in the diagram below, these instructions assume that Tomcat and Apache are running on the same server.



To encrypt client connections using Apache and SSL

1. Create a self-signed certificate. (If desired, see http://www.openssl.org/docs/HOWTO/certificates.txt for more information.)

   > Instead of creating a self-signed certificate, you can buy a certificate from a certificate vendor. If you do, then the vendor will provide instructions on how to configure Apache with your certificate.

2. Do the following:

   a. Run these commands:

   ```
   cd /etc/pki/tls/certs
   cp -p make-dummy-cert make-dummy-cert.bak
   cp -p localhost.crt localhost.crt.bak
   ```

   b. Edit `make-dummy-cert` and replace the **answers()** function with code similar to this:

```
answers() {
   echo US
   echo Utah
   echo Provo
   echo Adaptive Computing Enterprises, Inc.
   echo Engineering
   echo test1.adaptivecomputing.com
   echo
}
```

c. Run this command:

```
./make-dummy-cert localhost.crt
```

3. Configure Apache to use the new certificate and to redirect MWS requests to Tomcat. To do so, edit `/etc/httpd/conf.d/ssl.conf`. Do the following"

   a. Comment out this line:

   ```
   SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
   ```

   b. Add these lines near the end, just above **</VirtualHost>**:

   ```
   ProxyPass /mws http://127.0.0.1:8080/mws retry=5
   ProxyPassReverse /mws http://127.0.0.1:8080/mws
   ```

4. Configure Apache to use SSL for all MWS requests. Add these lines to the end of `/etc/httpd/conf/httpd.conf`:

   ```
   RewriteEngine On
   RewriteCond %{HTTPS} off
   RewriteRule (/mws.*) https://%{HTTP_HOST}%{REQUEST_URI}
   ```

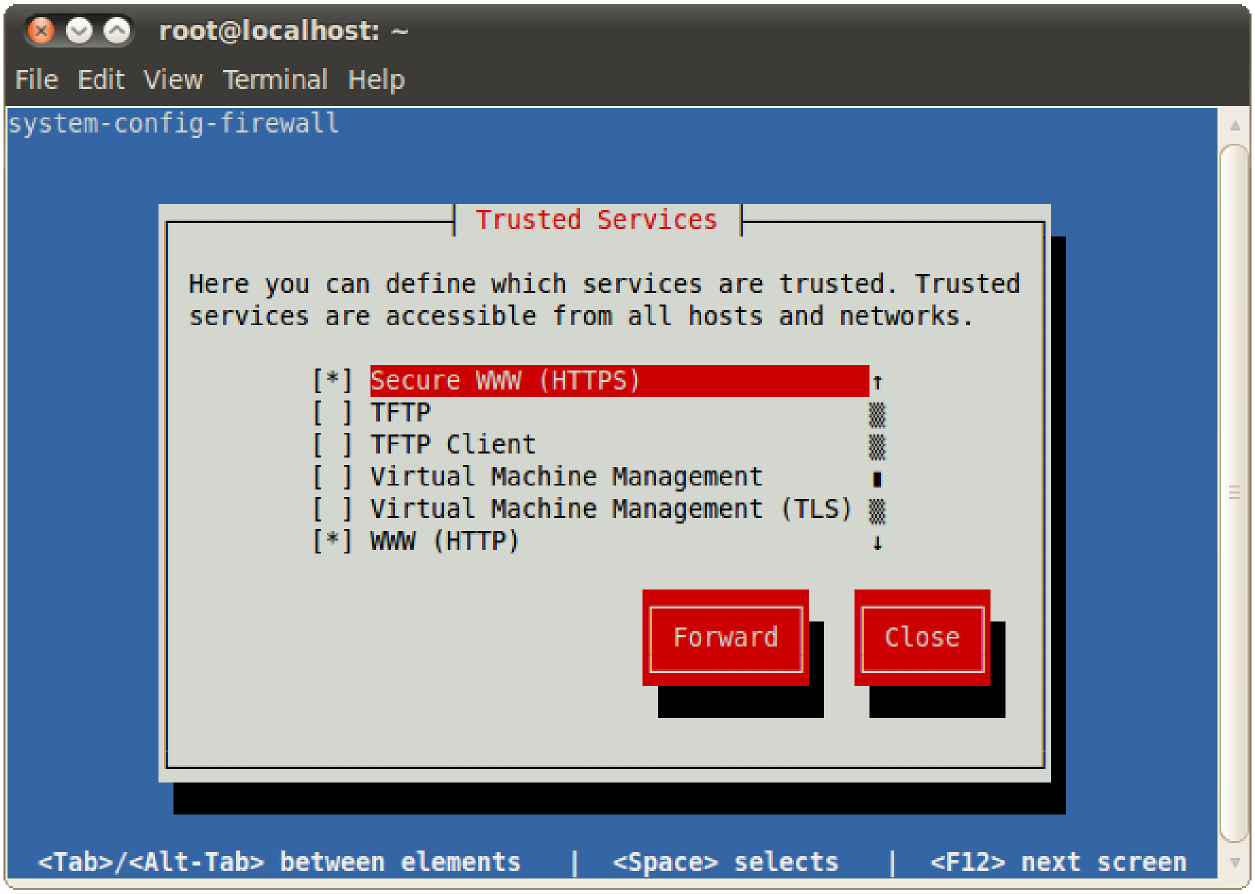
5. Give Apache permission to connect to Tomcat.

   ```
   setsebool -P httpd_can_network_connect 1
   ```

6. Turn on Apache.

   ```
   chkconfig httpd on
   service httpd start
   ```

7. Using `system-config-firewall-tui`, enable "Secure WWW (HTTPS)" and "WWW (HTTP)" as trusted services.



Encrypting client connections using Tomcat and SSL

This section shows how to encrypt client connections to MWS using Tomcat and SSL but without requiring the use of Apache. These instructions have been tested on CentOS™ 6.2 with Tomcat 6.0.

To encrypt client connections using Tomcat and SSL

1. First, you must generate a certificate. Do the following:

   a. Use the keytool utility that is shipped with the Oracle Java Runtime Environment. As the Tomcat user, run the following:

   ```
   keytool -genkey -alias tomcat -keyalg RSA
   ```

   b. Specify a password value of "changeit". This will create a .keystore file that contains the new certificate in the user's home directory.

2. Enable the Tomcat SSL connector. Do the following:

   a. Open the `server.xml` file, usually located in `$CATALINA_HOME/conf/` (`$CATALINA_HOME` represents the directory where Tomcat is installed).

b. Verify the SSL HTTP/1.1 Connector entry is enabled. To do so locate the SSL HTTP/1.1 Connector entry and uncomment it.

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true" maxThreads="150"
scheme="https" secure="true" clientAuth="false" sslProtocol="TLS" />
```

> The code above enables SSL access on port 8443. The default for HTTPS is 443, but just as Tomcat uses 8080 instead of 80 to avoid conflicts, 8443 is used instead of 443.

c. Save the `server.xml` file.

d. Verify that `server.xml` is owned by the Tomcat user.

```
chown -R tomcat:tomcat server.xml
```

e. Next modify the MWS`web.xml` file. Add a security-constraint section to the `$CATALINA_HOME/webapps/mws/WEB-INF/web.xml` file found in your Tomcat directory.

```
<web-app>
   ...
      <security-constraint>
         <web-resource-collection>
            <web-resource-name>MWS Secure URLs</web-resource-name>
            <url-pattern>/*</url-pattern>
         </web-resource-collection>
         <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
         </user-data-constraint>
      </security-constraint>
</web-app>
```

f. Now restart tomcat.

Related topics

- **Setting up MWS security** on page 1388

Securing the LDAP connection

All connections from MWS to the LDAP server should be secured with SSL or StartTLS to ensure passwords and other sensitive information are encrypted as they pass to and from the LDAP server. If the LDAP server does not support SSL or StartTLS, the rest of this section is irrelevant.

Determine whether the LDAP server's certificate is trusted

If the LDAP server's X.509 certificate has been signed by a trusted certificate authority such as Verisign, Thawte, GeoTrust, and so on, Java will trust the certificate automatically and you won't need to add the certificate to Java's keystore. Consult your IT department to determine whether the LDAP server certificate has been signed by a trusted certificate authority. If the LDAP server certificate is signed by a trusted certificate authority, skip ahead to **Configure MWS to connect to LDAP server using SSL or**

StartTLS on page 1395. Otherwise, follow the instructions in Trusting servers in Java on page 79 to add the certificate to Java's keystore.

Configure MWS to connect to LDAP server using SSL or StartTLS

To configure MWS to connect to LDAP using SSL/TLS

1. Update the `ldap.port` and `ldap.security.type` parameters in `/opt/mws/etc/mws-config.groovy`.

```
ldap.port = 636
ldap.security.type = "SSL"
```

To configure MWS to connect to LDAP using StartTLS

1. Update the `ldap.port` and `ldap.security.type` parameters in `/opt/mws/etc/mws-config.groovy`.

```
ldap.port = 389
ldap.security.type = "StartTLS"
```

The table below lists the possible values for `ldap.security.type`:

ldap.security.type	Default port	Notes
None	389	This is the default if no security type is configured. All data is sent in plain text.
SSL	636	Requires server certificate. All data is encrypted.
StartTLS	389	Starts as an insecure connection and is upgraded to an SSL/TLS connection. Requires server certificate. After upgrade all data is encrypted.

Securing the connection with the message queue

MWS supports message queue security with AES. If the **moab.messageQueue.secretKey** property is set, then all messages MWS publishes on the message queue will be encrypted. Additionally, MWS can read messages from Moab Workload Manager that are encrypted with the same key using the MESSAGEQUEUESECRETKEY parameter. For more information, see Configuration on page 1750.

Encryption is done with AES in CBC mode where inputs are padded with PKCS5 padding. Only 128-bit (16-byte) keys are supported. Keys should be encoded in Base64.

For example:

```
moab.messageQueue.secretKey = "1r6RvfqJa6voezy5wAx0hw==" //must be a Base64-encoded
128-bit key
```

> Important: If MWS is configured to encrypt the message queue and Moab is not (or vice versa) then the messages from Moab will be ignored. Furthermore, all attempts to access the MWS service resource will fail.

Related topics

- **Resources introduction** on page 1424
- **Events** on page 1506
- **Notifications** on page 1563
- **Notification conditions** on page 1558
- **Creating events and notifications** on page 1682
- **Plugin developer's guide** on page 1657
- **Fields: Events** on page 1837
- **Plugin event service** on page 1733
- **Handling events** on page 1689
- **System events** on page 1422
- **Securing the connection with the message queue** on page 1395

Version and build information

To get detailed version information about MWS, use one of the following three methods:

- **Browser** on page 1396
- **REST request** on page 1397
- **MANIFEST.MF file** on page 1397

Browser

Using a browser, visit the MWS home page (for example, `http://localhost:8080/mws/`). At the bottom of the page is the MWS version information. See the screenshot below:

REST request

Using a REST client or other HTTP client software, send a GET request to the `rest/diag/about` resource. Here is an example:

```
curl -u username:password http://localhost:8080/mws/rest/diag/about?api-version=3
```

This resource is also described under **Diagnostics** on page 1499.

MANIFEST.MF file

If MWS fails to start, version and build information can be found in the `META-INF/MANIFEST.MF` file inside the MWS WAR file. The version properties begin with **Implementation**. Below is an excerpt of a `MANIFEST.MF` file:

```
Implementation-Build: 26
Implementation-Build-Date: 2012-06-19_14-18-59
Implementation-Revision: 376079a5e5f552f2fe25e6070fd2e84c646a98fd

Name: Grails Application
Implementation-Title: mws
Implementation-Version: 7.1.0-rc2
Grails-Version: 2.0.3
```

Related topics

- **Setting up MWS security** on page 1388

Access control

About access control

Access control

This section describes how to manage access control in MWS. Applications are the consumers of MWS. They include Moab Viewpoint and other applications that need the resources provided by MWS. An application account consists of four editable fields and resource-specific access control settings:

Table 4-1: Field information

Field	Required	Default value	Value type	Maximum length	Description
Application Name	Yes	--	String	32	The name of the application. Must start with a letter and may contain letters, digits, underscores, periods, hyphens, apostrophes, and spaces.
Username	Yes	--	String	32	Used for authentication. Must start with a letter and may contain letters, digits, underscores, periods, and hyphens.
Description	No	--	String	1000	The description of the application.
Enabled	--	true	Boolean	--	Controls whether the application is allowed to access MWS.
Access Control Settings	Yes	All Permissions	--	--	The permissions granted to the application. This is controlled by selecting specific check boxes in a grid.

An application account also contains an auto-generated password that is visible only when creating the account or when resetting its password. Whenever an application sends a REST request to MWS, it needs to pass its credentials (username and password) in a Basic Authentication header. For more information, see **Authentication** on page 1421.

The **Application Name** is a human-friendly way to identify an application account, but MWS does not use it during authentication (or at any other time, for that matter).

The **Enabled** field is set to true automatically when an application account is created. To change the value of this field, see **Modifying an application account** on page 1400.

Here is an example of how you might set the fields when creating an application account:

- **Application Name**: Moab Viewpoint
- **Username**: viewpoint
- **Description**: This application account grants access to Moab Viewpoint for Moab Cloud Suite.

The permissions granted to an application account may be customized while creating or modifying the account. For more information, see **Creating an application account** on page 1399 and **Modifying an application account** on page 1400.

Managing application accounts

Application accounts are used to grant access to MWS. Every application with an application account must be granted at least one access control permission to a resource in MWS. To manage application accounts, see **Listing application accounts** on page 1399.

Listing application accounts

To list all applications accounts, browse to the MWS home page (for example, `https://servername/mws`). Log in as the admin user, click **Admin** and then **Application Accounts**.

Each column (except **Password**) can be sorted in ascending or descending order by clicking on the column heading.

Creating an application account

To create an application account, go to the **Application List** page and click **Add Application**. The "Application Name" and "Username" are required fields. For more details, see **Field information** on page 1398.

Access to specific resources and plugin custom web services is granted or revoked by checking or unchecking the check boxes in the respective resources or plugin web services access control sections. For each resource, access may be granted to a resource for each method supported by MWS, including GET, POST, PUT, and DELETE. See the figure below for an example.

In this example, the application has access to all available methods for the **Access Control Lists** and **Accounts** resources as well as to retrieve the **Events** resource through the GET method, but is denied the permission to create new events through the POST method.

Access may also be granted to each plugin type's custom web service(s). When new plugin types or plugin web services are added to MWS, applications must be updated with the new access control settings. See below for an example.

Plugin Type	☑ Can Access
☑ Test	
customService	☑
unsecuredService (Unsecured)	✓

In this example, the application has access to all the custom web services defined for the **Test** plugin type. Note that though unsecured web services are listed, access to them cannot be denied (for more information, see **Exposing web services** on page 1674).

Displaying an application account

To show information about an application account, go to the **Application List** page and click the desired application name.

In addition to displaying the values for fields, grids are also displayed which represent the application's access control permissions defined for resources and plugin custom web services. Examples of the resources and the plugin web services access control displays are shown below:

	GET	POST	PUT	DELETE
Access Control Lists			✓	✓
Accounts	✓			
Events	✓	⊖		

Plugin Type	Can Access
Test	
customService	✓
unsecuredService (Unsecured)	✓

Modifying an application account

To modify an application account, go to the **Application List** page, click the desired application name, and then click **Edit**. See **Creating an application account** on page 1399 for more information on available fields and access control settings.

Resetting an application password

To reset an application password, go to the **Application List** page and click the **Reset** link for the desired application. Alternatively, go to the **Display Application** page for the desired application and click the **Reset** link.

Deleting an application account

To delete an application account, go to the **Application List** page, click the desired application name, and then click **Delete**. A confirmation message is shown. If the **OK** button is clicked, the application account is deleted from the system and cannot be recovered.

Related topics

- **Moab Web Services overview** on page 1373
- **Setting up MWS security** on page 1388

API documentation

About the API

Moab Web Services provide a set of RESTful resources that can be used to create, read, update, and delete various objects in the Moab Workload Manager. This section describes how to use RESTful web services, explains the JSON data format used for all communications with MWS, describes global URL parameters used in MWS calls, and contains other helpuful information for using the Moab Web Services API.

This section contains these topics:

Related topics

- **Resources introduction** on page 1424
- **About Moab Web Services plugins** on page 1650

RESTful web services

In order to understand how to use MWS, it is first necessary to give a brief introduction to REST. REST (Representational State Transfer) is a set of guidelines which utilizes the full HTTP (Hypertext Transfer

Protocol) specification along with endpoint URLs that describe resources. The HTTP methods used in REST are comprised of the following:

Method	Description
GET	Query for a list or a single resource.
POST	Creating a resource.
PUT	Modifying a resource.
DELETE	Deleting a resource.

In comparison to other architectures of web services which use a single HTTP method and service endpoint to perform multiple types of operations (such as a POST operation to a URL), REST utilizes all of the available HTTP methods and URLs that directly correlate to resources. For example, RESTful web services for books in a library may expose many URL endpoints and the HTTP methods available for each such as GET, POST, PUT, and DELETE. The list below gives the methods, URLs, and descriptions for a sample set of services. The number 1 represents a unique identifier for books in each case.

Method	URL	Description
GET	`/books`	Retrieves a list of all books in the library.
POST	`/books`	Creates a new book.
GET	`/books/1`	Retrieves a single book.
PUT	`/books/1`	Modifies a single books.
DELETE	`/books/1`	Deletes a single book.

> Note that in the cases of the POST and PUT operations, additional information may be needed to describe the resource to be created or the fields that should be modified.

Moab Web Services provides RESTful web services for many resources. The methods and URLs available are documented in **Resources introduction** on page 1424.

Related topics

- **About the API** on page 1401

Data format

JSON (JavaScript Object Notation) is the data format used for all communication with MWS. This format makes use of two main structures: collections of key/value pairs called *objects* and ordered lists of values called *arrays*. Objects are defined by using curly braces ({}), and arrays are defined by using square brackets ([]). A JSON object or array may contain several different types of values including numbers, booleans (true/false), strings, objects, arrays, or the keyword 'null' representing no value. For example, a simple JSON object might be defined as:

```
{
  "number": 1,
  "decimalNumber": 1.2,
  "boolean": true,
  "string": "Any string",
  "dateString": "2013-05-23 17:32:02 UTC",
  "object": {
    "key": "value"
  },
  "array": [
    "value1",
    "value2"
  ],
  "nullValue": null
}
```

Dates in MWS, for both input and output, use the pattern "`yyyy-MM-dd HH:mm:ss ZZZ`". For more details on that pattern, see Joda-Time DateTimeFormat. For a list of valid time zone IDs, see Joda-Time Available Time Zones.

For more information on JSON, see json.org.

The data format of MWS is defined as follows:

- Input for a POST or PUT must be in JSON format. Set the `Content-Type` header to `application/json`.
- Output is in JSON format and always consists of an object with zero or more key/value pairs.
- The output may also be "pretty-printed" or formatted for human viewing by sending a URL parameter. For more information, see **Global URL parameters** on page 1403.

Related topics

- **About the API** on page 1401

Global URL parameters

All URL parameters are optional.

Parameter	Valid values	Description
api-version	Integer	Requests a specific API version
pretty	*true*	Controls pretty printing of output
fields	Comma-separated string	Includes only specified fields in output
exclude-fields	Comma-separated string	Excludes specified fields from output
max	Integer	The maximum number of items to return
offset	Integer	The index of the first item to return

API version (api-version)

See **Requesting specific API versions** on page 1406 for information on this parameter and how it should be used.

Pretty (pretty)

By default, the output is easy for a machine to read but difficult for humans to read. The **pretty** parameter formats the output so that it is easier to read.

Field selection (fields)

The **fields** parameter will include *only* the specified fields in the output. For list queries, the field selection acts on the objects in **results** and not on the **totalCount** or **results** properties themselves.

The format of the **fields** parameter is a comma-separated list of properties that should be included, as in `id,state`. Using periods, sub-objects may also be specified, and fields of these objects may be included as well. This is done with the same syntax for both single sub-objects and lists of sub-objects, as in `id,requirements.requiredNodeCountMinimum,blockReason.message`.

Example 4-1: Example for a job query

```
Request
------------------------------

GET /rest/jobs?api-
version=3&fields=name,flags,requirements.taskCount,dates.createdDate
```

```
Response
------------------------------

{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
    "dates": {"createdDate": "2012-10-17 01:11:54 UTC"},
    "flags": ["GLOBALQUEUE"],
    "name": "Moab.24",
    "requirements": [{"taskCount": 1}]
  }]
}
```

Field exclusion (exclude-fields)

The **exclude-fields** parameter is the opposite of the **fields** parameter. All fields will be included in the output *except* those that are specified. For list queries, the field exclusion acts on the objects in **results** and not on the **totalCount** or **results** properties themselves.

The format of the **exclude-fields** parameter is a comma-separated list of properties that should be excluded from the output, as in `id,state`. Using periods, sub-objects may also be specified, and fields of these objects may be excluded as well. This is done with the same syntax for both single sub-objects and lists of sub-objects, as in `id,requirements.requiredNodeCountMinimum,blockReason.message`.

Example 4-2:

Suppose a query returns the following JSON:

```
Request with No Field Exclusion
------------------------------

GET /objects
```

```
Response
------------------------------

{
  "id": "1",
  "listOfStrings":   [
    "string1",
    "string2"
  ],
  "listOfObjects": [  {
    "item1": "value1",
    "item2": "value2"
  }],
  "singleObject":   {
    "id": "obj1",
    "field1": "value1"
  }
}
```

The same query with exclude-fields would return the following output:

```
Request with No Field Exclusion
-------------------------------

GET /objects?exclude-fields=id,listOfObjects.item2,singleObject.field1,listOfStrings
```

```
Response
-------------------------------

{
  "listOfObjects": [{"item1": "value1"}],
  "singleObject": {"id": "obj1"}
}
```

Sorting (sort)

Images on page 1514 and **Events** on page 1506 support sorting based on MongoDB syntax by using the sort parameter. To sort in ascending order, specify a `1` for the sorting field. To sort in descending order, specify a `-1`. Objects can also be sorted on nested fields by using dot notation to separate the sub-fields, such as `field.subfield1.subfield2`.

Related topics

- **About the API** on page 1401

Requesting specific API versions

Because of significant changes in the API introduced in release version 7.2.0, MWS possesses a versioned API. The api-version URL parameter may be used to change the requested API version for any call to MWS. The current valid API versions with their corresponding MWS versions are shown in the table below:

API version	MWS version	Documentation	Additional notes
2 (deprecated)	7.2.x	7.2.x documentation on http://docs.adaptivecomputing.com/	As of the 8.0.1 release, API version 2 is **officially deprecated** and will be removed from Moab Web Services in the next major release.
3	8.0	Contained within this document	--
latest	Latest	Contained within this document	When the `latest` API version is requested, it resolves to the latest API version of MWS, such as api-version=3 for MWS 8.0.1.

> If no API version is specified, the request is rejected. An API version must be specified with every call in Moab Web Services 8.0.1 and later.

Resources introduction on page 1424 and **Resources reference** on page 1760 contain information for the latest API version. For documentation of previous API versions, please see the table above.

Examples

```
GET http://localhost:8080/mws/rest/nodes?api-version=2
// Data returned uses API version 2

GET http://localhost:8080/mws/rest/nodes?api-version=latest
// Data returned uses API version 3
```

Related topics

- **About the API** on page 1401

Responses and return codes

Various HTTP responses and return codes are generated from MWS operations. These are documented below according to the operation that they are associated with.

Listing and showing resources

For any successful list or show operation `(GET)`, a `200 OK` response code is always returned. No additional headers beyond those typical of a HTTP response are given in the response.

The body of this response consists of the results of the list or show operation. For a list operation, the results are wrapped in metadata giving total and result counts. The result count represents the number of resource records returned in the current request, and the total count represents the number of all records available. These differ when querying or the `max` and `offset` parameters are used. The following is an example of a list operation response:

```
JSON List Response Body
------------------------------------

{
  "resultCount":1,
  "totalCount":5,
  "results":[
    {
      "id":"Moab.1",
      …
    }
  ]
}
```

For a show operation, the result is given as a single object:

```
JSON Show Response Body
------------------------------------

{
  "id":"Moab.1",
  …
}
```

Creating resources

A successful creation (POST) of a resource has two potential response codes:

- If the resource was created immediately, a 201 Created response code is returned.
- If the resource is still being created, a 202 Accepted response code is returned.

In either case, a `Location` header is added to the response with the full URL which can be used to get more information about the newly created resource or the task associated with creating the resource (if a `202` is returned).

Additionally, the body of the response will contain the unique identifier of the newly created resource or the unique identifier for the task associated with creating the resource (if a `202` is returned).

For example, during creation or submission of a job, a `201` response code is returned with the following response headers and body:

```
Job Creation Response Headers
------------------------------------

HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Location: /mws/rest/jobs/Moab.21
X-Moab-Status: Success
X-Moab-Code: 000
Content-Type: application/json;charset=utf-8
Content-Length: 16
Date: Wed, 21 Dec 2011 23:04:47 GMT
```

```
Job Creation Response Body
------------------------------------

{"id":"Moab.21"}
```

Modifying resources

For any successful resource modification operation `(PUT)`, a `200 OK` or `202` Accepted response code is returned. A `200` response code signifies that the modification was immediately completed. No additional headers are returned in this case. A `202` response code is used again to signify that the modification is not yet complete and additional actions are taking place. In this case, a `Location` header is also returned with the full URL of the resource describing the additional actions.

In the case of a `200` response code, the body of this response typically consists of an object with a single `messages` property containing a list of statuses or results of the modification(s). However, a few exceptions to this rule exist as documented in **Resources introduction** on page 1424. In the case of a `202` response code, the format is the same as for a `202` during a creation operation, in that the body consists of an object with the unique identifier for the task associated with the additional action(s).

For example, when modifying a job, several messages may be returned as follows with the associated `200` response code.

```
Job Modification Response Headers
------------------------------------

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Moab-Status: Success
X-Moab-Code: 000
X-Moab-Message:
Content-Type: application/json;charset=utf-8
Content-Length: …
Date: Thu, 22 Dec 2011 16:49:43 GMT
```

```
JSON Modify Response Body
------------------------------------

{
  "messages":[
    "gevent processed",
    "variables successfully modified"
  ]
}
```

Deleting resources

For any successful resource deletion operation (`DELETE`), a `200 OK` or `202 Accepted` response code is returned. A `200` response code signifies that the deletion was immediately completed. No additional headers are returned in this case. A `202` response code is used again to signify that the deletion is not yet complete and additional actions are taking place. In this case, a `Location` header is also returned with the full URL of the resource describing the additional actions.

In the case of a `200` response code, the body of this response is empty. In the case of a `202` response code, the format is the same as for a `202` during a creation operation, in that the body consists of an object with the unique identifier for the task associated with the additional action(s).

For example, when deleting a job, a `200` response code is returned with an empty body as shown below.

```
Job Deletion Response
------------------------------------

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Moab-Status: Success
X-Moab-Code: 000
X-Moab-Message:
Content-Type: application/json;charset=utf-8
Content-Length: 0
Date: Thu, 22 Dec 2011 16:49:43 GMT
```

Moab response headers

In addition to the typical HTTP headers and the Location header described above, several headers are returned if the operations directly interact with Moab. These headers are described in the following table:

Name	Description
X-Moab-Status	One of `Success`, `Warning`, or `Failure`. Describes the overall status of the Moab request.
X-Moab-Code	A three digit code specifying the exact error encountered, used only in debugging.
X-Moab-Message	An optional message returned by Moab during the request.

Related topics

- **About the API** on page 1401

Error messages

Below is an explanation of what error message format to expect when an HTTP status code other than 20x is returned. All error codes have a response code of 400 or greater.

- **400 Bad request** on page 1411
- **401 Unauthorized** on page 1411
- **403 Forbidden** on page 1411
- **404 Not found** on page 1411
- **405 Method Not Allowed** on page 1411
- **500 Internal server error** on page 1412

400 Bad request

This response code is returned when the request itself is at fault, such as when trying to modify a resource with an empty `PUT` request body or when trying to create a new resource with invalid parameters. The response body is as follows:

```
{
  "messages":[
    "Message describing error",
    "Possible prompt to take action"
  ]
}
```

401 Unauthorized

This response code is returned when authentication credentials are not supplied or are invalid. The response body is as follows:

```
{
  "messages":[
    "You must be authenticated to access this area"
  ]
}
```

403 Forbidden

This response code is returned when the credentials supplied are valid, but the permissions granted are insufficient for the operation. This occurs when using application accounts (see **Access control** on page 1398) with limited access.

```
{
  "messages":[
    "You are not authorized to access this area"
  ]
}
```

404 Not found

This response code is returned when the request specifies a resource that does not exist. The response body is as follows:

```
{
  "messages":[
    "The resource with id 'uniqueId' was not found"
  ]
}
```

405 Method Not Allowed

This response code is returned when a resource does not support the specified HTTP method as an operation. The response body is as follows:

```
{
  "messages":[
    "The specified HTTP method is not allowed for the requested resource"
  ]
}
```


500 Internal server error

This indicates that there was an internal server error while performing the request, or that an operation failed in an unexpected manner. These are the most serious errors returned by MWS. If additional information is needed, the MWS log may contain further error data. The response body is as follows:


```
{
  "messages":[
    "A problem occurred while processing the request",
    "A message describing the error"
  ]
}
```


Related topics

- **About the API** on page 1401

Pre and post-processing hooks

MWS provides functionality to intercept and modify data sent to and returned from web services for all available resources. This is done by creating hooks in Groovy files located in a sub-directory of the `MWS_HOME` directory (by default, `/opt/mws/hooks`).

> Please see **Reference** on page 1416 in this topic for the full reference for available hooks and methods available to them.

Configuring hooks

The directory of the `hooks` folder may be changed by providing a value for `mws.hooks.location` in the configuration file. If the directory starts with a path separator (ie `/path/to/hooks`), it will be treated as an absolute path. Otherwise, it will be used relative to the location of the MWS home directory (for more information, see **Configuring Moab Web Services** on page 1373).

For example, if the MWS home directory is set to /opt/mws, the hooks directory by default would be in /opt/mws/hooks. Changing the mws.hooks.location property to myhooks would result in the hooks directory being located at /opt/mws/myhooks. Due to the default location of the MWS home directory, the default directory of the hooks directory is /opt/mws/hooks.

On startup, if the hooks directory does not exist, it will be created with a simple README.txt file with instructions on how to create hooks, the objects available, and the hooks available. If the folder or file is unable to be created, a message will be printed on the log with the full location of a README file, copied into a temporary directory.

Defining hooks for a resource

Hooks are defined for resources by creating groovy class files in the hooks directory (MWS_HOME/hooks by default). Each groovy file must be named by the resource URL it is associated with and end in ".groovy". The following table shows some possible hook files that may be created. Notice that the virtual machines hook file is abbreviated as vms, just as the URL for virtual machines is /rest/vms. In most cases, the hook file names will exactly match the URLs. However, in cases of nested URLs—such as with "accounting/users"—the hook file name must replace slashes with periods. For example:

Resource	Hook filename
Jobs	jobs.groovy
Nodes	nodes.groovy
Virtual Machines	vms.groovy
Accounting Users	accounting.users.groovy
Accounting Funds Reports Statement	accounting.funds.reports.statement.groovy
Accounting Charge Rates	accounting.charge-rates.groovy
url	*url.groovy*

plugins.rm.groovy is a valid hook filename. It works for the following URL: /rest/plugins/<pluginID or all>/rm/<query or action> (for example, /rest/plugins/plugin1/rm/cluster-query).

A complete example of a hook file is as follows:

```
Complete Hook File
------------------------------------

// Example before hook
def beforeList = {
  // Perform actions here
  // Return true to allow the API call to execute normally
  return true
}

def beforeShow = {
  // Perform actions here
  // Render messages to the user with a 405 Method Not Allowed
  // HTTP response code
  renderMessages("Custom message here", 405)
  // Return false to stop normal execution of the API call
  return false
}

// Example after hook
def afterList = { o ->
  if (!isSuccess()) {
    // Handle error here
    return false
  }
  // Perform actions here
  return o
}
```

You must convert all actions or queries that are separated by dashes to a camel case. For example, the hooks called for "cluster-query" should be `beforeClusterQuery` and `afterClusterQuery`.

As the specific format for the hooks for `before` and `after` are different, each will be explained separately.

Before hooks

As shown above, before hooks require no arguments. They can directly act on several properties, objects, and methods as described in **Reference** on page 1416. The return value is one of the most important aspects of a `before` hook. If it is `false`, a `renderMessages`, `renderObject`, `renderList`, `render`, or `redirect` method *must* first be called. This signifies that the API call should be interrupted and the render or redirect action specified within the hook is to be completed immediately.

A return value of `true` signifies that the API call should continue normally. Parameters, session variables, request and response variables may all be modified within a `before` hook.

If no return value is explicitly given, the result of the last statement in the `before` hook to be executed will be returned. This may cause unexpected behavior if the last statement resolves to `false`.

For all methods available to `before` hooks as well as specific examples, see **beforeSave** on page 1416.

After hooks

`After` hooks are always passed one argument: the object or list that is to be rendered as JSON. This may be modified as desired, but note that the object or list value is either a JSONArray or JSONObject. Therefore, it may not be accessed and modified as a typical groovy Map.

Unlike `before` hooks, `after` hooks should not call the `render*` methods directly. This method will automatically be called on the resulting object or list returned. The `redirect` and `render` methods should also not be called at this point. Instead, if a custom object or list is desired to be used, the `serializeObject` and `serializeList` methods are available to create suitable results to return.

The return value of an `after` hook may be one of two possibilities:

- The potentially modified object or list passed as the first argument to the hook. In this case, this value will override the output object or list unless it is null.
- Null or false. In this case, the original, unmodified object or list will be used in the output.

> The return value of the `after` hook, if not null or false, *must* be the modified object passed into the hook or an object or list created with the `serialize*` methods.

For all methods available to after hooks as well as specific examples, see **afterSave** on page 1416.

Error handling

`After` hooks, unlike the `before` hooks, have the possibility of handling errors encountered during the course of the request. Handling errors is as simple as adding a one-line check to the hook as shown above or in the following code:

```
if (!isSuccess()) {
  // Handle error
  return false
}
```

It is recommended that each `after` hook contain at least these lines of code to prevent confusion on what the input object or list represents or should look like.

The `isSuccess()` function is false if and only if the HTTP response code is 400 or higher, such as a 404 Not Found, 400 Bad Request, or 500 Internal Server Error and the cause of the error state was not in the associated `before` hook. In other words, objects and lists rendered in the `before` hook with any HTTP response code will never run the associated `after` hook.

When handling errors, the passed in object will always contain a `messages` property containing a list of strings describing the error(s) encountered.

Defining common hooks

Sometimes it is beneficial to create hooks which are executed for all calls of a certain type, such as a `beforeList` hook that is executed during the course of listing any resource. These are possible using an `all.groovy` file. The format of this file is exactly the same as other hook files. The order of execution is as follows:

1. `Before` common hook executed.
2. `Before` resource-specific hook executed.
3. Normal API call executed.
4. `After` resource-specific hook executed.
5. `After` common hook executed.

Reference

This page gives specific examples and reference for implementing hooks in MWS.

Available hooks

The following table lists the available hooks for each resource with their associated HTTP method and description.

Name	HTTP method	Description
beforeList	GET	Runs before an API call that lists resources (for example, `GET /rest/jobs`).
afterList	GET	Runs after an API call that lists resources.
beforeShow	GET	Runs before an API call that returns a single resource (for example, `GET /rest/jobs/job.1`).
afterShow	GET	Runs after an API call that returns a single resource.
beforeSave	POST	Runs before an API call that saves a new resource (for example, `POST /rest/jobs`).
afterSave	POST	Runs after an API call that returns a single resource.
beforeUpdate	PUT	Runs before an API call that returns a single resource (for example, `PUT /rest/jobs/job.1`).
afterUpdate	PUT	Runs after an API call that returns a single resource.
beforeDelete	DELETE	Runs before an API call that returns a single resource (for example, `DELETE /rest/jobs/job.1`).
afterDelete	DELETE	Runs after an API call that returns a single resource.

> If a resource does not support a certain operation, any hooks for that operation will simply be ignored—such as **beforeSave** and **afterSave** hooks for the Node resource, where saving is not supported.

Available properties

The following table lists the properties, objects, and methods available in all hooks. Note that although it is possible to directly call the `render*` methods in the `after` hooks, it is not recommended.

Name	Type	Description
params	Map	Contains all URL parameters as well as the body of the request as parsed JSON.
request	HttpServletRequest	Contains properties of the HTTP request.
response	HttpServletResponse	Contains properties of the HTTP response which can be modified directly.
session	HttpSession	Contains the session parameters which can be modified directly.
flash	Map	Temporary storage that stores objects within the session for the next request only.
controllerName	String	The name of the controller responding to the request. Only available in `before` hooks.
actionName	String	The name of the action to be run on the controller. Only available in `before` hooks.
apiVersion	String	The API version for the current request (for example, 1 for 7.0 and 7.1, 2 for 7.2).

> The parsed JSON may be accessed in `before` hooks as a simple groovy Map with `params[controllerName]`.

In addition, several methods are available to the hooks. These are described in the following sections.

Redirect

The `redirect` method may be used to redirect the request to another API call or an arbitrary URL.

```
redirect(uri:'/rest/jobs')       // uri is used for internal redirection within MWS
redirect(url:'http://adaptivecomputing.com')    // url is used for external redirection
redirect(uri:'http://adaptivecomputing.com', params:[lang:'en'])       // params may be used
for URL parameters
```

> The `redirect` method will use the GET HTTP method for the resulting redirected request.

See the `redirect` method's documentation for more information.

Rendering objects, lists, or messages

There are several `render*` methods available to handle any case where objects or lists are desired to be rendered directly from the hook without continuing to the API call. Three different methods may be used depending on the desired output object type:

```
Render object
------------------------------------

// Object that should be rendered as JSON
def objectToRender = …
// HTTP response code (bad request)
def responseCode = 400
// Render a simple object
renderObject(objectToRender)
// Render a simple object with a custom response code
renderObject(objectToRender, responseCode)
```

```
Render list
------------------------------------

// List that should be rendered as JSON
def listToRender = …
// If the totalCount property differs from resultCount, use this value instead
def totalCount = …
// HTTP response code (bad request)
def responseCode = 400
// Render a simple list
//     Dynamically adds "resultCount" and "totalCount" properties based on the size of
the input list
renderList(listToRender)
// Render a simple list with a custom "totalCount"
renderList(listToRender, totalCount)
// Render a simple list without changing the "totalCount" but with a custom response
code
renderList(listToRender, null, responseCode)
// Render a simple list with a custom "totalCount" and response code
renderList(listToRender, totalCount, responseCode)
```

```
Render message(s)
------------------------------------

// Messages
def messageToRender = "Single message"
def messagesListToRender = ["Message 1", "Message 2"]
// HTTP response code (bad request)
def responseCode = 400
// Render messages as an object with a property of "messages" containing a list of the
messages passed in
renderMessages(messageToRender)
renderMessages(messageToRender, responseCode)
// Supports either a single String or list of Strings
renderMessages(messagesListToRender)
renderMessages(messagesListToRender, responseCode)
```

> It is not recommended to call any of these methods from an `after` hook.

Render

Less commonly used, the `render` method is also available directly. This may be used to render text directly, change the content-type of the output, and many other functions. See the `render` method's documentation for more information.

> It is not recommended to call this method from an `after` hook.

Serialize objects

The `serializeObject` and `serializeList` methods may be used to convert a custom object or list respectively into a format usable for returning in the `after` hooks. Simply pass in the object or list and a serialized version will be returned from the method.

```
def afterShow = {
	def objectToRender = …
	def serializedObject = serializeObject(objectToRender)
	return serializedObject
}
```

```
def afterShow = {
	def listToRender = [...]
	def serializedList = serializeList(listToRender)
	return serializedList
}
```

Error handling

Error handling is only available in `after` hooks by using the following check:

```
if (!isSuccess()) {
	// Handle error
	return … // False or modified object/list to render
}
```

Usage examples

Override an API call

The following hook would serve to override an entire API call, the list call in this case, and return a `messages` list containing a single element of "Action is not supported" and a HTTP response code of 405 (Method Not Allowed):

```
def beforeList = {
	renderMessages("Action is not supported", 405)
	return false
}
```

To be even more specific and disallow the deletion of virtual machines, the following may be used as the `vms.groovy` file:

```
def beforeDelete = {
       renderMessages("Virtual Machine deletion is not allowed", 405)
       return false
}
```

Add an additional property during job creation

To add an additional property to a job definition during creation, create a `beforeSave` hook in the `jobs.groovy` file as follows:

```
def beforeSave = {
       // params[controllerName] is equivalent to params["job"] or params.job
       params[controllerName].user = "myuser"
}
```

This would cause the created job to have a user of `myuser`.

Redirect based on URL parameter

To redirect an API call if a certain URL parameter exists, create a `beforeSave` hook in the `jobs.groovy` file as follows:

```
def beforeSave = {
       if (params.external) {
               redirect(url:'http://example.com/create-job')
               return false;    // Stop API call
       }
}
```

This would cause an API call of `PUT /rest/jobs?external=1` to redirect to `GET http://example.com/create-job`.

Remove a property from getting a single job

To remove a property from the output of getting a single job, create an `afterShow` hook in the `jobs.groovy` file as follows:

```
def afterShow = { o ->
       o.discard("group")
       return o
}
```

This will cause the resulting JSON to be missing the `group` property of the job resource. Note again that these calls must use the JSONArray and JSONObject classes as mentioned in **After hooks** on page 1415.

Filter list items

To filter the items in a list nodes request based on user provided query parameter in the URL, use the following in the `nodes.groovy` file. A sample request that would activate the filter is `http://localhost:8080/mws/rest/nodes?api-version=3&filter-power=On`.

```
def afterList = { o ->
        // Do not filter if the user did not ask for it
        if (!params['filter-power'])
                return o
        // o = {resultCount: x, totalCount: x, results:[...]}

        // Using a built-in groovy method findAll to return all
        //      list items that return true from the block
        def results = o.results.findAll { node ->
                // Includes the node only if the power equals the user input
                return params['filter-power'].equalsIgnoreCase(node.power)
        }

        // Sets the results on the return object and updates the counts
        o.element("results", results)
        o.element("resultCount", results.size())
        return o
}
```

To filter the items in a list nodes request based on values within the list itself, such as variable values, use the following in the `nodes.groovy` file.

```
def afterList = { o ->
        // o = {resultCount: x, totalCount: x, results:[...]}
        // Using a built-in groovy method findAll to return all
        //      list items that return true from the block
        def results = o.results.findAll { node ->
                // Includes the node only if the variable "included" is set to "true"
                return node.variables?.included=="true"
        }

        // Sets the results on the return object and updates the counts
        o.element("results", results)
        o.element("resultCount", results.size())
        return o
}
```

Related topics

- **About the API** on page 1401

Authentication

MWS uses Basic Authentication for all REST API requests. This means that a username and password must be provided for each call to resources. There are two types of accounts that can be granted access: **Users** and **Applications**.

- For instructions on how to set the credentials for the default **User** account, see **Setting up MWS security** on page 1388.
- For instructions on how to manage **Application** accounts, see **Access control** on page 1398.

To use Basic Authentication, each client request must contain a header that looks like this:

```
Authorization: Basic YWRhcHRpdmU6YzNVU3R1bkU=
```

The string after the word `Basic` is the base64 encoding of *username : password*. In the example above, `YWRhcHRpdmU6YzNVU3R1bkU=` is the base64 encoding of `adaptive:c3UStunE`. For more details, see section 2 of RFC 2617.

The username and password in the Basic Authentication header are encoded but not encrypted. Therefore, it is *strongly* recommended that MWS be run behind a proxy (like Apache) with SSL enabled. For more information, see **Setting up MWS security** on page 1388.

Related topics

- **About the API** on page 1401

System events

The broad category of system events may be broken down into two subcategories: events and notification conditions.

- **Events** on page 1422
- **Notification conditions** on page 1423

Events

Events on page 1506 are created by many components in the system, but most events originate from Moab Workload Manager and Moab Web Services. Events can be created via the MWS interface or by being placed on the message queue. The ZeroMQ™ message queue libraries were introduced in Moab and MWS 7.5.0. The message queue is critical to service lifecycle functionality (see "Service lifecycle" in the *Moab Cloud Suite Installation Guide*).

In a typical system, Moab will communicate events to MWS via a "private" message queue, and then MWS will replicate the events on the "public" message queue, or the message queue that is available to subscribers with the correct secret keys. In some cases, such as those related to the MWS service lifecycle, MWS uses events to determine activities or capabilities that are available.

A typical message on the message queue may look like the following (sent with a topic of `system.moab`):

```
Sample message on message queue
------------------------------------

{
       "body" : {
               "associatedObjects" : [
                       {
                               "id" : "Moab",
                               "type" : "scheduler"
                       }
               ],
               "code" : 16777619,
               "eventDate" : "2014-02-28T10:57:21.000-0700",
               "message" : "A scheduler iteration is ending.",
               "origin" : "MSysMainLoop.c, MSysMainLoop, line 959"
       },
       "messageId" : "843269550",
       "messageType" : "event",
       "senderId" : "mwm@mwm-server",
       "sentDate" : "2014-02-28T10:57:21.000-0700",
       "ttl" : 3000
}
```


Notification conditions

Notification conditions on page 1558 are related to an event, but differ in three distinct areas:

1. Notification conditions are a persistent condition of the system or a component rather than a single occurrence.
   - They are ongoing rather than reoccurring, which is why they are generated from NotificationConditions.
   - They may be observed many times, but the condition is always the same.
   - A good test for this is if something "is" wrong rather than something "went" wrong.
2. Notification conditions can be acted on to result in a resolved state, mean the administrator or user can and must take actions to "fix" the condition or problem.
3. Notification conditions contain state information based on administrator or user input, meaning that they contain information about the condition (similar to events), but also contain the "status" of the administrator's view of the notification, whether it is currently open, dismissed, or ignored.

In general, questions may be asked to ascertain whether an event or a notification condition is the right fit for an occurrence. These questions, along with some sample situations, are provided below.

- Is the occurrence the root cause of a potentially ongoing condition?
  - A VM migration failed because the VM's state was unknown. The root cause was that the state was unknown, not that the VM migration failed. Therefore, VM migration failed would be an event, while the unknown state would be a notification condition.
  - A VM service provision fails because there are no hypervisors that satisfy the requirements. This would be an event. Note that there may be a notification related to this failure, such as a service template requires a feature that does not exist on any

hypervisors in the system, but this would be distinctly detected and managed from the provision failure event.

- A request to MWS failed because the connection between Moab and MongoDB was misconfigured. The failed request may be represented as an event, but a notification condition should exists that the connection between Moab and MongoDB was down.

- Can an administrator or user affect the outcome of the occurrence?
  - The outcome of a VM migration failing is in the past and cannot be changed by the administrator. However, the outcome of a future VM migration may be changed when the administrator resolves the root problem (such as VM state is unknown).

Related topics

- **Events** on page 1506
- **Notifications** on page 1563
- **Notification conditions** on page 1558
- **Securing the connection with the message queue** on page 1395
- **Creating events and notifications** on page 1682 (for plugin development only)
- **Plugin event service** on page 1733

Resources

Resources introduction

The sections in this chapter show the MWS resources and the HTTP methods defined on them. The prefix for these resources depends on how the `mws.war` file is deployed. A typical prefix would be `http://localhost:8080/mws`. Using this example, one absolute resource URI would be `http://localhost:8080/mws/rest/jobs`.

This section only contains documentation for the latest API version. Please see the table in **Requesting specific API versions** on page 1406 for links to documentation for previous versions.

This chapter contains these sections:

Related topics

- **Resources reference** on page 1760

Access control lists (ACLs)

This topic describes behavior of the ACL Rules (Access Control List Rules) object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The **Fields: Access Control Lists (ACLs)** reference contains the type and description of all fields in the **ACL Rules** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported methods

ACLs are not directly manipulated through a single URL, but with sub-URLs of the other objects such as Virtual Containers and Reservations.

Resource	GET	PUT	POST	DELETE
/rest/reservations/<rsvId>/acl-rules/<aclId>	--	**Create or update ACL**	--	**Delete ACL**
/rest/vcs/<vcId>/acl-rules/<aclId>	--	**Create or update ACL**	--	**Delete ACL**

This topic contains these sections:

Getting ACLs

Although **ACL Rules** cannot be retrieved directly using the GET method on any of the `acl-rules` resources, **ACL Rules** are attached to supported objects when querying for them. Each supported object contains a field named `aclRules`, which is a collection of the **ACL Rules** defined on that object.

Supported objects

The following is a list of objects that will return **ACL Rules** when queried:

- **Reservations** on page 1624
- **Standing reservations** on page 1639

Creating or updating ACLs

The HTTP PUT method is used to create or update **ACL Rules**. The request body can contain one or more **ACL Rules**. If an **ACL Rule** with the same `type` and `value` exists, then it will be overwritten.

Quick reference

```
PUT http://localhost:8080/mws/rest/reservations/<rsvId>/acl-rules?api-version=3
```

Create or update ACL

URLs and parameters

```
PUT http://localhost:8080/mws/rest/reservations/<rsvId>/acl-rules?api-version=3
```

Parameter	Required	Type	Valid values	Description
objectId	Yes	String	--	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

The request body below shows all the fields that are available for the PUT method, along with some sample values.

```
JSON Request Body
------------------------------------

{"aclRules": [{
  "affinity": "POSITIVE",
  "comparator": "LEXIGRAPHIC_EQUAL",
  "type": "USER",
  "value": "ted"
}]}
```

Sample response

This message may not match the message returned from Moab exactly, but is given as an example of the structure of the response.

```
JSON Request Body
------------------------------------

{"messages":["Reservation 'rsv1' successfully modified"]}
```

Samples

Create or update multiple ACLs on a single object:

```
CPUT http://localhost:8080/mws/rest/reservations/system.21/acl-rules?api-version=3
------------------------------------

{"aclRules": [
    {
    "affinity": "POSITIVE",
    "comparator": "LESS_THAN_OR_EQUAL",
    "type": "DURATION",
    "value": "3600"
  },
    {
    "affinity": "POSITIVE",
    "comparator": "LEXIGRAPHIC_EQUAL",
    "type": "USER",
    "value": "ted"
  }
]}
```

Restrictions

- **ACL Rules** cannot be added to or updated on **Standing Reservations**.

Deleting ACLs

The HTTP DELETE method is used to remove **ACL Rules**.

Quick reference

ACL Rules cannot be removed from **Standing Reservations**.

```
DELETE http://localhost:8080/mws/rest/reservations/<rsvId>/acl-rules?api-
version=3/<aclId>
```

Delete ACL

URLs and parameters

```
DELETE http://localhost:8080/mws/rest/reservations/<objectId>/acl-rules?api-
version=3/<aclId>
```

Parameter	Required	Type	Valid values	Description
objectId	Yes	String	--	The unique identifier of the object from which to remove the **ACL Rule**.
aclId	Yes	String	--	A string representing the **ACL Rule**, with the format `type:value`.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

> This message may not match the message returned from Moab exactly, but is given as an example of the structure of the response.

```
JSON Response
-----------------------------------

{"messages":["Successfully modified reservation 'rsv1'"]}
```

Restrictions

- **ACL Rules** cannot be removed from **Standing Reservations**.

Related topics

- **Fields: Access Control Lists (ACLs)** on page 1761
- **Resources introduction** on page 1424

Accounting

Accounting Accounts

This section describes the services available through Moab Web Services for interacting with the **Account** object in Moab Accounting Manager. It contains the URLs, request bodies, and responses delivered to and from MWS as an intermediary for MAM.

> The **Fields: Accounts** reference contains the type and description of the default fields for the **Accounts** object.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/accounting/accounts	Get all accounts	--	--	--
/rest/accounting/accounts/<id>	Get single account	--	--	--

This topic contains these sections:

Getting accounts

The HTTP GET method is used to retrieve **Accounts** information.

Quick reference

```
GET http://localhost:8080/mws/rest/accounting/accounts?api-version=3
GET http://localhost:8080/mws/rest/accounting/accounts/<id>?api-version=3
```

Get all accounts

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/accounts?api-version=3&proxy-user=<user>
[&query=<query_conditions>][&fields=<fields_to_display>[&sort=<fields_to_sort>]|&show-
all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`
query	No	JSON	--	Results are restricted to those having the specified field values. The **query** parameter does not support the full Mongo query syntax. Only querying for a simple, non-nested JSON object is allowed.	`query={"organ-ization":"sciences"}`

Parameter	Required	Type	Valid values	Description	Example
fields	No	String	--	Comma-separated list of field names to display.	`fields=id,organization`
sort	No	JSON	--	Sort the results. Use `1` for ascending and `-1` for descending. Should be used in conjunction with the **fields** parameter.	`sort={"organization":1}`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/accounts?api-version=3&proxy-
user=amy&fields=id,organization&pretty=true
------------------------------------

{
  "totalCount": 2,
  "resultCount": 2,
  "results":   [
        {
      "organization": "sciences",
      "id": "biology"
    },
        {
      "organization": "sciences",
      "id": "chemistry"
    }
  ]
}
```

Get single account

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/accounts/<id>?api-version=3&proxy-
user=<user>[&fields=<fields_to_display>|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
id	Yes	String	--	The unique identifier of the object.	--
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`
fields	No	String	--	Comma-separated list of field names to display.	`fields=id,organization`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/accounts/chemistry?api-version=3&proxy-
user=amy&pretty=true
------------------------------------
```


```
{
  "id": "chemistry",
  "active": true,
  "organization": "",
  "description": "Chemistry Dept",
  "users":   [
        {
      "id": "amy",
      "active": true,
      "admin": false
    },
        {
      "id": "bob",
      "active": true,
      "admin": false
    },
        {
      "id": "dave",
      "active": true,
      "admin": false
    }
  ]
}
```


Related topics

- **Fields: Accounts** on page 1770
- **Resources introduction** on page 1424

Accounting Allocations

This section describes the services available through Moab Web Services for interacting with the **Allocation** object in Moab Accounting Manager. It contains the URLs, request bodies, and responses delivered to and from MWS as an intermediary for MAM.

> The **Fields: Allocations** reference contains the type and description of the default fields for the **Allocation** object.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/accounting/allocations	**Get all allocations**	--	--	--
/rest/accounting/allocations/<id>	**Get single allocation**	--	--	--

This topic contains these sections:

Getting allocations

The HTTP GET method is used to retrieve **Allocation** information.

Quick reference

```
GET http://localhost:8080/mws/rest/accounting/allocations?api-version=3
GET http://localhost:8080/mws/rest/accounting/allocations/<id>?api-version=3
```

Get all allocations

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/allocations?api-version=3&proxy-
user=<user>[&query=<query_conditions>][&fields=<fields_to_display>[&sort=<fields_to_
sort>]|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`
query	No	JSON	--	Results are restricted to those having the specified field values. The **query** parameter does not support the full Mongo query syntax. Only querying for a simple, non-nested JSON object is allowed.	`query={"active":true}`
fields	No	String	--	Comma-separated list of field names to display.	`fields=id,fund,amount`

Parameter	Required	Type	Valid values	Description	Example
sort	No	JSON	--	Sort the results. Use `1` for ascending and `-1` for descending. Should be used in conjunction with the **fields** parameter.	`sort={"fund":1}`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/allocations?api-version=3&proxy-
user=amy&pretty=true
------------------------------------

{
  "totalCount": 5,
  "resultCount": 5,
  "results":   [
        {
      "id": 1,
      "fund": 1,
      "startTime": "2013-07-12 22:16:33 UTC",
      "endTime": "infinity",
      "amount": 50000000,
      "creditLimit": 0,
      "initialDeposit": 50000000,
      "allocated": 50000000,
      "active": true,
      "description": ""
    },
        {
      "id": 3,
      "fund": 3,
      "startTime": "2013-07-12 22:16:33 UTC",
      "endTime": "infinity",
      "amount": 0,
      "creditLimit": 20000000,
      "initialDeposit": 0,
      "allocated": 0,
      "active": true,
      "description": ""
    },
        {
      "id": 2,
      "fund": 2,
      "startTime": "2013-07-12 22:16:33 UTC",
      "endTime": "infinity",
      "amount": 30000000,
      "creditLimit": 0,
      "initialDeposit": 30000000,
      "allocated": 30000000,
      "active": true,
      "description": ""
    }
  ]
}
```

Get single allocation

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/allocations/<id>?api-version=3&proxy-
user=<user>[&fields=<fields_to_display>|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
id	Yes	String	--	The unique identifier of the object.	--
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`
fields	No	String	--	Comma-separated list of field names to display.	`fields=id,fund,amount`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/allocations/1?api-version=3&proxy-
user=amy&pretty=true
------------------------------------

{
  "id": 1,
  "fund": 1,
  "startTime": "2013-07-12 22:16:33 UTC",
  "endTime": "infinity",
  "amount": 50000000,
  "creditLimit": 0,
  "initialDeposit": 50000000,
  "allocated": 50000000,
  "active": true,
}
```

Related topics

- **Fields: Allocations** on page 1773
- **Resources introduction** on page 1424

Accounting Charge rates

This section describes the services available through Moab Web Services for interacting with the **ChargeRate** object in Moab Accounting Manager. It contains the URLs, request bodies, and responses delivered to and from MWS as an intermediary for MAM.

The **Fields: Charge Rates** reference contains the type and description of the default fields for the **ChargeRates** object.

Resource	GET	PUT	POST	DELETE
/rest/accounting/charge-rates	Get all charge rates	--	--	--
/rest/accounting/charge-rates/<name>/<value>	Get single charge rate	--	--	--
/rest/accounting/charge-rates/<name>	Get single charge rate	--	--	--

This topic contains these sections:

Getting charge rates

The HTTP GET method is used to retrieve **ChargeRate** information.

Quick reference

```
GET http://localhost:8080/mws/rest/accounting/charge-rates?api-version=3
GET http://localhost:8080/mws/rest/accounting/charge-rates?api-version=3/<name>
[/<value>]
```

Get all charge rates

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/charge-rates?api-version=3&proxy-
user=<user>[&query=<query_conditions>][&fields=<fields_to_display>[&sort=<fields_to_
sort>]|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`

Parameter	Required	Type	Valid values	Description	Example
query	No	JSON	--	Results are restricted to those having the specified field values. The **query** parameter does not support the full Mongo query syntax. Only querying for a simple, non-nested JSON object is allowed.	`query={"name":"QualityOfService"}`
fields	No	String	--	Comma-separated list of field names to display.	`fields=id,organization`
sort	No	JSON	--	Sort the results. Use `1` for ascending and `-1` for descending. Should be used in conjunction with the **fields** parameter.	`sort={"organization":1}`

Parameter	Required	Type	Valid values	Description	Example
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/charge-rates?api-version=3&proxy-
user=moab&pretty=true
------------------------------------

{
  "totalCount": 4,
  "resultCount": 4,
  "results":   [
        {
      "name": "Processors",
      "value": "",
      "amount": "1/s",
      "description": "1 credit per processor-second"
    },
        {
      "name": "QualityOfService",
      "value": "high",
      "amount": "*2",
      "description": "Charge double for high QOS"
    },
        {
      "name": "QualityOfService",
      "value": "low",
      "amount": "*.5",
      "description": "Charge half for low QOS"
    },
        {
      "name": "QualityOfService",
      "value": "",
      "amount": "*1",
      "description": "No extra charge for \"normal\" QOSes"
    }
  ]
}
```

Get single charge rate

A regular charge rate is uniquely specified by both its name and its value. A default charge rate has a null value and is uniquely specified by only its name.

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/charge-rates?api-version=3/<name>
[/<value>]?proxy-user=<user>[&fields=<fields_to_display>|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
name	Yes	String	--	The name of the charge rate.	--
value	No	String	--	The value of the charge rate.	--
fields	No	String	--	Comma-separated list of field names to display.	`fields=name,value,amount`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/charge-rates/QualityOfService/high?api-
version=3&proxy-user=moab&pretty=true
------------------------------------

{
  "name": "QualityOfService",
  "value": "high",
  "amount": "*2",
  "description": "Charge double for high QOS"
}
```

Related topics

- **Fields: Charge Rates** on page 1777
- **Resources introduction** on page 1424

Accounting Funds

This section describes the services available through Moab Web Services for interacting with the **Fund** object in Moab Accounting Manager. It contains the URLs, request bodies, and responses delivered to and from MWS as an intermediary for MAM.

The **Fields: Funds**, **Fields: Fund Balances**, **Fields: Fund Statements**, and **Fields: Fund Statement Summary** reference sections contain the type and description of the default fields in the **Fund** object as well as related objects and reports given in the URLs below.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/accounting/funds	Get all funds	--	--	--
/rest/accounting/funds/<id>	Get single fund	--	--	--
/rest/accounting/funds/balances	Get all fund balances	--	--	--
/rest/accounting/funds/reports/statement	Get fund statement	--	--	--
/rest/accounting/funds/reports/statement/summary	Get fund statement summary	--	--	--

This topic contains these sections:

Getting funds

The HTTP GET method is used to retrieve **Fund** information.

Quick reference

```
GET http://localhost:8080/mws/rest/accounting/funds?api-version=3
GET http://localhost:8080/mws/rest/accounting/funds/<id>?api-version=3
GET http://localhost:8080/mws/rest/accounting/funds/balances?api-version=3
GET http://localhost:8080/mws/rest/accounting/funds/reports/statement?api-version=3
GET http://localhost:8080/mws/rest/accounting/funds/reports/statement/summary?api-
version=3
```

Get all funds

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/funds?api-version=3&proxy-user=<user>
[&active=true][&filter=<filter_options>[&filter-type=<filter_type>]][&query=<query_
conditions>][&fields=<fields_to_display>[&sort=<fields_to_sort>]|&show-all=
(true|false)]
```

Parameter	Required	Type	Description	Example
proxy-user	Yes	String	Perform action as defined MAM user.	`proxy-user=amy`
active	No	Boolean	Lists only active or non-active allocations of the fund. The fund amount becomes the sum of the active/inactive allocations.	`active=true`
filter	No	JSON	Query funds based on defined MAM filter.	`filter={"account":"chemistry"}`
filter-type	No	String	Query funds based on defined MAM filter type.	`filter-type=NonExclusive`

Parameter	Required	Type	Description	Example
query	No	JSON	Results are restricted to those having the specified field values. The **query** parameter does not support the full Mongo query syntax. Only querying for a simple, non-nested JSON object is allowed.	`query={"priority":"2","allocation.active":"false"}`
fields	No	String	Comma-separated list of field names to display.	`fields=id,name,amount`
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending. Should be used in conjunction with the **fields** parameter.	`sort={"id":1}`

Parameter	Required	Type	Description	Example
show-all	No	Boolean (`true` or `false`)	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/funds?api-version=3&proxy-
user=amy&fields=id,name,amount&pretty=true
------------------------------------

{
  "totalCount": 2,
  "resultCount": 2,
  "results":   [
        {
      "id": 1,
      "name": "biology",
      "amount": 50000000
    },
        {
      "id": 2,
      "name": "chemistry",
      "amount": 99727
    }
  ]
}
```

Get single fund

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/funds/<id>?api-version=3&proxy-
user=<user>[&active=(true|false)][&fields=<fields_to_display>|&show-all=(true|false)]
```

Parameter	Required	Type	Description	Example
id	Yes	String	The unique identifier of the object	--
proxy-user	Yes	String	Perform action as defined MAM user.	`proxy-user=amy`

Parameter	Required	Type	Description	Example
active	No	Boolean	Lists only active or non-active allocations of the fund. The fund amount becomes the sum of the active/inactive allocations.	`active=true`
fields	No	String	Comma-separated list of field names to display.	`fields=id,name,amount`
show-all	No	Boolean (`true` or `false`)	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/funds/1?api-version=3&proxy-
user=amy&pretty=true
------------------------------------

{
  "id": 1,
  "name": "biology",
  "priority": 0,
  "defaultDeposit": 50000000,
  "description": "",
  "amount": 50000000,
  "allocated": 50000000,
  "initialDeposit": 50000000,
  "creditLimit": 0,
  "allocations":   [
        {
      "id": 1,
      "startTime": "2013-08-21 16:57:53 UTC",
      "endTime": "infinity",
      "amount": 50000000,
      "creditLimit": 0,
      "initialDeposit": 50000000,
      "allocated": 50000000,
      "active": false,
      "description": ""
    }
  ],
  "fundConstraints": [  {
    "id": 1,
    "name": "Account",
    "value": "biology"
  }]
}
```

Get all fund balances

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/funds/balances?api-version=3&proxy-
user=<user>[&filter=<filter_options>][&filter-type=<filter_type>]
```

Parameter	Required	Type	Description	Example
proxy-user	Yes	String	Perform action as defined MAM user.	`proxy-user=amy`
filter	No	JSON	Query funds based on defined MAM filter.	`filter={"accoun-t":"chemistry"}`
filter-type	No	String	Query funds based on defined MAM filter type.	`filter-type-e=NonExclusive`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

The fund balances resource is an aggregation of fund data. For more information, see the **Fields: Fund Balances** on page 1779 reference section.

```
GET http://localhost:8080/mws/rest/accounting/funds/balances?api-version=3&proxy-
user=amy&pretty=true
------------------------------------

{
  "totalCount": 2,
  "resultCount": 2,
  "results": [
    {
      "id": 2,
      "name": 1204,
      "priority": 0,
      "description": "R&D for Manufacturing",
      "creationTime": "2012-02-02 09:34:42 UTC",
      "amount": 9060000,
      "deposited": 9060000,
      "creditLimit": 0,
      "reserved": 0,
      "allocations": [
        {
          "id": 2,
          "amount": 9060000,
          "creditLimit": 0,
          "deposited": 9060000
        }
      ],
      "fundConstraints": [
        {
          "id": 2,
          "name": "CostCenter",
          "value": 1204
        }
      ],
      "balance": 9060000,
      "available": 9060000,
      "allocated": 9060000,
      "used": 0,
      "percentRemaining": 100,
      "percentUsed": 0
    },
    {
      "id": 5,
      "name": "",
      "priority": 0,
      "description": "",
      "creationTime": "2012-04-03 09:25:47 UTC",
      "amount": 901290219001,
      "deposited": 901290219021,
      "creditLimit": 30,
      "reserved": 84018308897.68,
      "allocations": [
        {
          "id": 6,
          "amount": 901290219001,
          "creditLimit": 30,
          "deposited": 901290219021
        }
      ],
      "fundConstraints": [],
      "balance": 817271910103.32,
      "available": 817271910133.32,
      "allocated": 901290219051,
```

```
      "used": 20,
      "percentRemaining": 100,
      "percentUsed": 0
    }
  ]
}
```

Get fund statement

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/funds/reports/statement?api-
version=3&proxy-user=<user>[&filter=<filter_options>][&filter-type=<filter_type>]
[&start-time=<date_string>][&end-time=<date_string>][&context=<context>]
```

Parameter	Required	Type	Description	Example
proxy-user	Yes	String	Perform action as defined MAM user.	`proxy-user=amy`
filter	No	JSON	Query funds based on defined MAM filter.	`filter={"accoun-t":"chemistry"}`
filter-type	No	String	Query funds based on defined MAM filter type.	`filter-type-e=NonExclusive`
start-time	No	Date, -infinity, or now	Filter allocations and transaction after a start time.	`start-time=2012-04-03 15:24:39 UTC`
end-time	No	Date, -infinity, or now	Filter allocations and transactions before an end time.	`end-time=2012-04-03 15:24:39 UTC`
context	No	hpc or cloud	The context to use in Moab Accounting Manager. The **context** parameter overrides the default context set for MAM using the `mam.context` configuration parameter. For more information about this parameter, see **Configuration** on page 1750.	`context=hpc`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

The fund statement report provides a snapshot of the current funds. For more information, see **Fields: Fund Statements** on page 1797.

```
GET http://localhost:8080/mws/rest/accounting/funds/reports/statement?api-
version=3&proxy-user=amy&fields=startBalance,endBalance&pretty=true
------------------------------------

{
  "startBalance":1234.01,
  "endBalance":1000
}
```

Get fund statement summary

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/funds/reports/statement/summary?api-
version=3&proxy-user=<user>[&filter=<filter_options>][&filter-type=<filter_type>]
[&start-time=<date_string>][&end-time=<date_string>]
```

Parameter	Required	Type	Description	Example
proxy-user	Yes	String	Perform action as defined MAM user.	`proxy-user=amy`
filter	No	JSON	Query funds based on defined MAM filter.	`filter={"accoun-t":"chemistry"}`
filter-type	No	String	Query funds based on defined MAM filter type.	`filter-type-e=NonExclusive`
start-time	No	Date, -infin-ity, or now	Filter allocations and trans-action after a start time.	`start-time=2012-04-03 15:24:39 UTC`
end-time	No	Date, -infin-ity, or now	Filter allocations and trans-actions before an end time.	`end-time=2012-04-03 15:24:39 UTC`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

The fund statement summary is slightly different from the typical fund statement in that the transactions are provided as summaries grouped by object and action. For more information, see **Fields: Fund Statement Summary** on page 1786.

```
GET http://localhost:8080/mws/rest/accounting/funds/reports/statement/summary?api-
version=3&proxy-
user=amy&fields=totalCredits,totalDebits,transactions.action,transactions.amount,trans
actions.count&pretty=true
------------------------------------

{
  "totalCredits":200.02,
  "totalDebits":-100,
  "transactions":[ {
      "action":"Deposit",
      "amount":200.02,
      "count":2
    }, {
      "action":"Charge",
      "amount":-100,
      "count":1
    }
  ]
}
```

Related topics

- **Fields: Funds** on page 1807
- **Resources introduction** on page 1424

Accounting Liens

This section describes the services available through Moab Web Services for interacting with the Lien object in Moab Accounting Manager. It contains the URLs, request bodies, and responses delivered to and from MWS as an intermediary for MAM.

The **Fields: Liens** reference contains the type and description of the default fields for the **Liens** object.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/accounting/liens	Get all liens	--	--	--
/rest/accounting/liens/<id>	Get single lien	--	--	--

This topic contains these sections:

Getting liens

The HTTP GET method is used to retrieve **Lien** information.

Quick reference

```
GET http://localhost:8080/mws/rest/accounting/liens?api-version=3
GET http://localhost:8080/mws/rest/accounting/liens/<id>?api-version=3
```

Get all liens

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/liens?api-version=3&proxy-user=<user>
[&active=true][&filter=<filter_options>[&filter-type=<filter_type>]][&query=<query_
conditions>][&fields=<fields_to_display>[&sort=<fields_to_sort>]|&show-all=
(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`
active	No	Boolean	--	Lists only active or non-active liens.	`active=true`
filter	No	JSON	--	Query funds based on defined MAM filter.	`filter={"accoun-t":"chemistry"}`
filter-type	No	String	--	Query funds based on defined MAM filter type.	`filter-type=NonExclusive`

Parameter	Required	Type	Valid values	Description	Example
query	No	JSON	--	Results are restricted to those having the specified field values. The **query** parameter does not support the full Mongo query syntax. Only querying for a simple, non-nested JSON object is allowed.	`query={"allocations.fund":2}`
fields	No	String	--	Comma-separated list of field names to display.	`fields=id,instance,amount`
sort	No	JSON	--	Sort the results. Use `1` for ascending and `-1` for descending. Should be used in conjunction with the **fields** parameter.	`sort={"instance":1}`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/liens?api-version=3&proxy-
user=amy&filter={"account":"chemistry"}&fields=instance,amount&active=true&pretty=true
------------------------------------

{
  "totalCount": 2,
  "resultCount": 2,
  "results":   [
        {
      "instance": "job.1",
      "amount": 57600
    },
        {
      "instance": "job.2",
      "amount": 40762
    }
  ]
}
```

Get single lien

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/liens/<id>?api-version=3&proxy-
user=<user>[&active=(true|false)][&fields=<fields_to_display>|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
id	Yes	String	--	The unique identifier of the object	--
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`
active	No	Boolean	--	Lists only active or non-active liens.	`active=true`
fields	No	String	--	Comma-separated list of field names to display.	`fields=id,name,amount`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/liens/1?api-version=3&proxy-
user=amy&pretty=true
------------------------------------

{
  "id": 1,
  "instance": "job.1",
  "usageRecord": 1,
  "startTime": "2013-08-21 16:45:57 UTC",
  "endTime": "2013-08-21 17:45:57 UTC",
  "duration": 3600,
  "description": "",
  "amount": 57600,
  "allocations": [  {
    "id": 2,
    "fund": 2,
    "amount": 57600
  }]
}
```

Related topics

- **Fields: Liens** on page 1815
- **Resources introduction** on page 1424

Accounting Organizations

This section describes the services available through Moab Web Services for interacting with the **Organization** object in Moab Accounting Manager. It contains the URLs, request bodies, and responses delivered to and from MWS as an intermediary for MAM.

The **Fields: Organizations** reference contains the type and description of the default fields for the **Organization** object.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/accounting/organizations	Get all organizations	--	--	--
/rest/accounting/organizations/<id>	Get single organization	--	--	--

This topic contains these sections:

Getting organizations

The HTTP GET method is used to retrieve **Organizations** information.

Quick reference

```
GET http://localhost:8080/mws/rest/accounting/organizations?api-version=3
GET http://localhost:8080/mws/rest/accounting/organizations/<id>?api-version=3
```

Get all organizations

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/organizations?api-version=3&proxy-
user=<user>[&query=<query_conditions>][&fields=<fields_to_display>[&sort=<fields_to_
sort>]|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`
query	No	JSON	--	Results are restricted to those having the specified field values. The query parameter does not support the full Mongo query syntax. Only querying for a simple, non-nested JSON object is allowed.	`query={"deleted":-false}`
fields	No	String	--	Comma-separated list of field names to display.	`fields=id`
sort	No	JSON	--	Sort the results. Use `1` for ascending and `-1` for descending. Should be used in conjunction with the **fields** parameter.	`sort={"reques-tedId":-1}`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/organizations?api-version=3&proxy-
user=moab&fields=id,description&sort={"id":1}&pretty=true
------------------------------------

{
  "totalCount": 2,
  "resultCount": 2,
  "results":   [
        {
      "description": "Arts College",
      "id": "arts"
    },
        {
      "description": "Sciences College",
      "id": "sciences"
    }
  ]
}
```

Get single organization

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/organizations/<id>?api-version=3&proxy-
user=<user>[&fields=<fields_to_display>|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
id	Yes	String	--	The unique identifier of the object.	--
fields	No	String	--	Comma-separated list of field names to display.	`fields=id`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all-l=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/organizations/sciences?api-
version=3&proxy-user=moab&pretty=true
------------------------------------

{
  "description": "Sciences College",
  "id": "sciences"
}
```

Related topics

- **Fields: Organizations** on page 1819
- **Resources introduction** on page 1424

Accounting Quotes

This section describes the services available through Moab Web Services for interacting with the **Quote** object in Moab Accounting Manager. It contains the URLs, request bodies, and responses delivered to and from MWS as an intermediary for MAM.

> The **Fields: Quotes** reference contains the type and description of the default fields for the **Quotes** object.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/accounting/quotes	**Get all quotes**	--	--	--
/rest/accounting/quotes/<id>	**Get single quote**	--	--	--

This topic contains these sections:

Getting quotes

The HTTP GET method is used to retrieve **Quote** information.

Quick reference

```
GET http://localhost:8080/mws/rest/accounting/quotes?api-version=3
GET http://localhost:8080/mws/rest/accounting/quotes/<id>?api-version=3
```

Get all quotes

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/quotes?api-version=3&proxy-user=<user>
[&active=true][&filter=<filter_options>[&filter-type=<filter_type>]][&query=<query_
conditions>][&fields=<fields_to_display>[&sort=<fields_to_sort>]|&show-all=
(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
proxy-user	Yes	String	--	Perform actions as defined MAM user.	`proxy-user=amy`
active	No	Boolean	`true` or `false`	Lists only active or non-active quotes.	`active=true`
filter	No	JSON	--	Query funds based on defined MAM filter.	`filter={"accoun-t":"chemistry"}`
filter-type	No	String	--	Query funds based on defined MAM filter type.	`filter-type=NonExclusive`
query	No	JSON	--	Results are restricted to those having the specified field values. The **query** parameter does not support the full Mongo query syntax. Only querying for a simple, non-nested JSON object is allowed.	`query={"instance":"-job.1"}`
fields	No	String	--	Comma-separated list of field names to display.	`fields=id,instance,amount`

Parameter	Required	Type	Valid values	Description	Example
sort	No	JSON	--	Sort the results. Use `1` for ascending and `-1` for descending. Should be used in conjunction with the **fields** parameter.	`sort={"instance":1}`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/quotes?api-version=3&proxy-
user=amy&filter={"account":"chemistry"}
&fields=usageRecord,amount&active=true&pretty=true
------------------------------------

{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
    "usageRecord": 1,
    "amount": 57600
  }]
}
```

Get single quote

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/quotes/<id>?api-version=3&proxy-
user=<user>[&active=(true|false)][&fields=<fields_to_display>|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
id	Yes	String	--	The unique identifier of the object.	--

Parameter	Required	Type	Valid values	Description	Example
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`
active	No	Boolean	`true` or `false`	Lists only active or non-active quotes.	`active=true`
fields	No	String	--	Comma-separated list of field names to display.	`fields=id,name,amount`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/quotes/1?api-version=3&proxy-
user=amy&pretty=true
------------------------------------

{
  "id": 1,
  "amount": 57600,
  "pinned": true,
  "instance": "",
  "usageRecord": 1,
  "startTime": "2013-08-21 16:45:57 UTC",
  "endTime": "2013-08-21 17:57:57 UTC",
  "duration": 3600,
  "description": "",
  "chargeRates": [  {
    "name": "Processors",
    "value": "",
    "amount": "1/s"
  }]
}
```

Related topics

- **Fields: Quotes** on page 1821
- **Resources introduction** on page 1424

Accounting Transactions

This section describes the services available through Moab Web Services for interacting with the **Transaction** object in Moab Accounting Manager. It contains the URLs, request bodies, and responses delivered to and from MWS as an intermediary for MAM.

The **Fields: Transactions** reference contains the type and description of the default fields for the **Transaction** object.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/accounting/transactions	Get all transactions	--	--	--
/rest/accounting/transactions/<id>	Get single transaction	--	--	--

This topic contains these sections:

Getting transactions

The HTTP GET method is used to retrieve **Transaction** information.

Quick reference

```
GET http://localhost:8080/mws/rest/accounting/transactions?api-version=3
GET http://localhost:8080/mws/rest/accounting/transactions/<id>?api-version=3
```

Get all transactions

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/transactions?api-version=3&proxy-
user=<user>[&query=<query_conditions>][&fields=<fields_to_display>[&sort=<fields_to_
sort>]|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`

Parameter	Required	Type	Valid values	Description	Example
query	No	JSON	--	Results are restricted to those having the specified field values. The **query** parameter does not support the full Mongo query syntax. Only querying for a simple, non-nested JSON object is allowed.	`query={"action":"Charge","account":"chemistry"}`
fields	No	String	--	Comma-separated list of field names to display.	`fields=id`

Parameter	Required	Type	Valid values	Description	Example
sort	No	JSON	--	Sort the results. Use `1` for ascending and `-1` for descending. Should be used in conjunction with the **Accounting Transactions** parameter.	`sort={"id":1}`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/transactions?api-version=3&proxy-
user=moab&query={"instance":"job.1"}&fields=object,action,instance,amount&pretty=true
------------------------------------

{
  "totalCount": 310,
  "resultCount": 3,
  "results":   [
        {
      "object": "UsageRecord",
      "action": "Reserve",
      "instance": "job.1",
      "amount": 57600
    },
        {
      "object": "UsageRecord",
      "action": "Charge",
      "instance": "job.1",
      "amount": 11520
    },
        {
      "object": "UsageRecord",
      "action": "Refund",
      "instance": "job.1",
      "amount": 11520
    }
  ]
}
```

Get single transaction

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/transactions/<id>?api-version=3&proxy-
user=<user>[&fields=<fields_to_display>|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
id	Yes	String	--	The unique identifier of the object.	--
fields	No	String	--	Comma-separated list of field names to display.	`fields=id`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all-l=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/transactions/1?api-version=3&proxy-
user=moab&pretty=true
------------------------------------
```


```
{
  "id": 1,
  "object": "Organization",
  "action": "Create",
  "actor": "scottmo",
  "key": "sciences",
  "child": "",
  "count": 1,
  "instance": "",
  "amount": "",
  "delta": "",
  "user": "",
  "account": "",
  "machine": "",
  "fund": "",
  "allocation": "",
  "usageRecord": "",
  "duration": "",
  "description": ""
}
```


Related topics

- **Fields: Transactions** on page 1826
- **Resources introduction** on page 1424

Accounting Usage records

This section describes the services available through Moab Web Services for interacting with the **Usage Record** object in Moab Accounting Manager. It contains the URLs, request bodies, and responses delivered to and from MWS as an intermediary for MAM.

The **Fields: Usage Records** reference section contains the type and description of all fields in the **Usage Record** object.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/accounting/usage-records	**Get all usage records**	--	--	--
/rest/accounting/usage-records/<id>	**Get single usage record**	--	--	--
/rest/accounting/usage-records/quote	--	--	**Obtain a quote for resource usage**	--

This topic contains these sections:

Getting usage records

The HTTP GET method is used to retrieve **Usage Record** information.

Quick reference

```
GET http://localhost:8080/mws/rest/accounting/usage-records?api-version=3
GET http://localhost:8080/mws/rest/accounting/usage-records/<id>?api-version=3
POST http://localhost:8080/mws/rest/accounting/usage-records/quote?api-version=3
```

Get all usage records

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/usage-records?api-version=3&proxy-
user=<user>[&query=<query_conditions>][&fields=<fields_to_display>[&sort=<fields_to_
sort>]|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`

Parameter	Required	Type	Valid values	Description	Example
query	No	JSON	--	Results are restricted to those having the specified field values. The **query** parameter does not support the full Mongo query syntax. Only querying for a simple, non-nested JSON object is allowed.	`query={"account":"query"}`
fields	No	String	--	Comma-separated list of field names to display.	`fields=id,instance,charge,user,account`

Parameter	Required	Type	Valid values	Description	Example
sort	No	JSON	--	Sort the results. Use `1` for ascending and `-1` for descending. Should be used in conjunction with the **fields** parameter.	`sort={"user":1}`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/usage-records?api-version=3&proxy-
user=amy&fields=id,instance,charge,user,account&pretty=true
------------------------------------

{
  "totalCount": 2,
  "resultCount": 2,
  "results":   [
        {
      "id": 1,
      "instance": "job.1",
      "charge": 31,
      "user": "amy",
      "account": "chemistry"
    },
        {
      "id": 2,
      "instance": "job.2",
      "charge": 30,
      "user": "amy",
      "account": "biology"
    }
}
```

Get single usage record

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/usage-records/<id>?api-version=3&proxy-
user=<user>[&fields=<fields_to_display>|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
id	Yes	String	--	The unique identifier of the object.	`code`
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`
fields	No	String	--	Comma-separated list of field names to display.	`fields=id,instance,charge,user,account`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/usage-records/1?api-version=3&proxy-
user=amy&pretty=true
------------------------------------
```


```
{
  "id": 1,
  "type": "Job",
  "instance": "job.1",
  "charge": 31,
  "stage": "Charge",
  "user": "amy",
  "group": "faculty",
  "account": "chemistry",
  "organization": "sciences",
  "qualityOfService": "",
  "machine": "colony",
  "nodes": "",
  "processors": 16,
  "memory": "",
  "disk": "",
  "network": "",
  "duration": 720,
  "startTime": "",
  "endTime": "",
  "description": ""
}
```


Obtain a quote for resource usage

URLs and parameters

```
POST http://localhost:8080/mws/rest/accounting/usage-records/quote?api-
version=3&object-type=<object>&proxy-user=<user>&charge-duration=<seconds>
```

Parameter	Required	Type	Valid values	Description	Example
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`
charge-duration	Yes	Integer	--	The quote duration of the job in seconds.	`charge-duration=6400`

Parameter	Required	Type	Valid values	Description	Example
object-type	Yes	String	--	The object to quote. It can be job or service.	`object-type=job`
itemize	No	Boolean	`true` or `false`	Returns the composite charge information in the response data.	`itemize=true`
rate	No	JSONArray	--	Uses the specified charge rates in the quote. The specified rates override the standard and quote rates. If the guarantee field is set to true, these charge rates will be saved and used when this quote is referenced in a charge action.	`rate=[{"type":"VBR","name":"Memory","rate":1},{"type":"VBR","name":"Processors","rate":1}]`

Parameter	Required	Type	Valid values	Description	Example
guarantee	No	Boolean	`true` or `false`	Guarantees the quote and returns a quote id to secure the current charge rates. This results in the creation of a quote record and a permanent usage record. This parameter is mutually exclusive with the cost-only parameter.	`guarantee=true`

Parameter	Required	Type	Valid values	Description	Example
grace-duration	No	Integer	--	The guaranteed quote grace period in seconds. If the quote duration is specified but not the quote end time, the quote endtime will be calculated as the quote start time plus the quote duration plus the grace duration.	`grace-duration=6400`
cost-only	No	Integer	--	Returns the cost, ignoring all balance and validity checks. This parameter is mutually exclusive with the guarantee parameter.	`cost-only=true`

Parameter	Required	Type	Valid values	Description	Example
description	No	String	--	The guaranteed quote description.	`description="ABC Coupon Rate"`
start-time	No	Date	--	The guaranteed quote start time in the format `yyyy-MM-dd HH:mm:ss z`, `-Infinity`, `Infinity`, or `Now`.	`start-time="2012-04-09 13:49:40 UTC"`
end-time	No	Date	--	The guaranteed quote end time in the format `yyyy-MM-dd HH:mm:ss z`, `-Infinity`, `Infinity`, or `Now`.	`end-time="2012-04-09 14:49:40 UTC"`

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

The request body below shows all of the fields in a job that could affect the quote.

```
POST http://localhost:8080/mws/rest/accounting/usage-records/quote?api-
version=3&object-type=job&charge-duration=300
------------------------------------

{
  "id": "Moab.1",
  "user": "amy",
  "group": "group",
  "rmName": "machine1",
  "templateList": [
  "genericVm"
  ],
  "account": "biology",
  "qosRequested": "QOS1",
  "variables": {
    "imageName": "centos5.5-stateless",
    "topLevelServiceId": "myService.1",
    "serviceId": "vmService.1",
    "vmid": "VmService.1",
    "pmid": "VmService.1"
  },
  "requirements": [
  {
      "requiredProcessorsPerTask": 2,
      "genericResources": {
         "gold": 100,
         "os": 500
      },
      "requiredNodeCountMinimum": 1,
      "requiredMemoryPerTask": 1024,
      "requiredClass": "batch"
  }
  ]
}
```

The request body below shows all of the fields in a service that affect the quote in a default MAM installation.

```
POST http://localhost:8080/mws/rest/accounting/usage-records/quote?api-
version=3&object-type=service&charge-duration=300
------------------------------------

{
     "name":"service.1",
     "user": "amy",
     "account": "chemistry"
     "attributes":{
        "moab":{
           "job":{
              "resources":{
                 "procs":1,
                 "mem":2048,
                 "OS":500,
                 "gold":100
              },
              "variables":{
                 "Var1": 1524
              },
              "image":"centos5.5-stateless",
              "template":"genericVM",
           }
        }
     }
  }
```

Sample response

- If the quote is not guaranteed:

```
JSON response
------------------------------------

{
    "instance": "Moab.1",
    "amount": 600
}
```

- If the quote is guaranteed:

```
JSON response
------------------------------------

{
    "id": 1,
    "usageRecord": 2,
    "instance": "Moab.1",
    "amount": 600
}
```

- If the quote is guaranteed and itemized:

```
JSON response
------------------------------------

{
    "details":   [
        {
      "name": "Processors",
      "value": "2",
      "duration": 300,
      "rate": 1,
      "scalingFactor": 1,
      "amount": 600,
      "details": "2 [Processors] * 1 [ChargeRate{VBR}{Processors}] * 300
[Duration]"
    },
        {
      "name": "Memory",
      "value": "1024",
      "duration": 300,
      "rate": 1,
      "scalingFactor": 1,
      "amount": 307200,
      "details": "1024 [Memory] * 1 [ChargeRate{VBR}{Memory}] * 300 [Duration]"
    }
  ],
  "id": 20,
  "instance": "Moab.1",
  "usageRecord": 20,
  "amount": 307800
}
```

- If the quote is on a service:

```
JSON response
------------------------------------

{
  "services":   [
        {
      "details":       [
                {
          "name": "Processors",
          "value": "22",
          "duration": 30,
          "rate": 1,
          "scalingFactor": 1,
          "amount": 660,
          "details": "22 [Processors] * 1 [ChargeRate{VBR}{Processors}] * 30
[Duration]"
        },
                {
          "name": "Memory",
          "value": "32343242",
          "duration": 30,
          "rate": 1,
          "scalingFactor": 1,
          "amount": 970297260,
          "details": "32343242 [Memory] * 1 [ChargeRate{VBR}{Memory}] * 30
[Duration]"
        }
      ],
      "id": 120,
      "instance": "myVmWorkflow",
      "usageRecord": 157,
      "amount": 970297920
    },
        {
      "details": [      {
        "name": "Storage",
        "value": "2500",
        "duration": 30,
        "rate": 1.157E-7,
        "scalingFactor": 1,
        "amount": 0,
        "details": "2500 [Storage] * 1.157e-07 [ChargeRate{VBR}{Storage}] * 30
[Duration]"
      }],
      "id": 122,
      "instance": "myExtraStorageWorkflow",
      "usageRecord": 159,
      "amount": 0
    },
        {
      "details":       [
                {
          "name": "Processors",
          "value": "0",
          "duration": 30,
          "rate": 1,
          "scalingFactor": 1,
          "amount": 0,
          "details": "0 [Processors] * 1 [ChargeRate{VBR}{Processors}] * 30
[Duration]"
        },
                {
```

```
            "name": "Memory",
            "value": "0",
            "duration": 30,
            "rate": 1,
            "scalingFactor": 1,
            "amount": 0,
            "details": "0 [Memory] * 1 [ChargeRate{VBR}{Memory}] * 30 [Duration]"
          }
        ],
        "id": 123,
        "instance": "myPmWorkflow",
        "usageRecord": 160,
        "amount": 0
      }
    ],
    "amount": 970297920
}
```


Restrictions

The `details` field is only available with MAM version 7.1.0 or later.

Related topics

- **Fields: Usage Records** on page 1830
- **Resources introduction** on page 1424

Accounting Users

This section describes the services available through Moab Web Services for interacting with the User object in Moab Accounting Manager. It contains the URLs, request bodies, and responses delivered to and from MWS as an intermediary for MAM.

The **Fields: Users** reference contains the type and description of all fields in the **User** object.

Supported methods

Resource	GET	PUT	POST	DELETE
rest/accounting/users	Get all users	--	--	--
rest/accounting/users/<id>	Get single user	--	--	--

This topic contains these sections:

Getting users

The HTTP GET method is used to retrieve **User** information.

Quick reference

```
GET http://localhost:8080/mws/rest/accounting/users?api-version=3
GET http://localhost:8080/mws/rest/accounting/users/<id>?api-version=3
```

Get all users

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/users?api-version=3&proxy-user=<user>
[&query=<query_conditions>][&fields=<fields_to_display>[&sort=<fields_to_sort>]|&show-
all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
proxy-user	Yes	String	--	Perform action as defined MAM user.	`proxy-user=amy`
query	No	JSON	--	Results are restricted to those having the specified field values. The **query** parameter does not support the full Mongo query syntax. Only querying for a simple, non-nested JSON object is allowed.	`query={"active":true}`

Parameter	Required	Type	Valid values	Description	Example
fields	No	String	--	Comma-separated list of field names to display.	`fields=name,defaultAccount`
sort	No	JSON	--	Sort the results. Use `1` for ascending and `-1` for descending. Should be used in conjunction with the **fields** parameter.	`sort={"defaultAccount":1}`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/users?api-version=3&proxy-
user=moab&query={"active":true}&pretty=true
------------------------------------

{
  "totalCount": 6,
  "resultCount": 4,
  "results":   [
        {
      "active": true,
      "commonName": "",
      "phoneNumber": "",
      "emailAddress": "",
      "defaultAccount": "",
      "description": "Accounting Admin",
      "id": "scottmo"
    },
        {
      "active": true,
      "commonName": "Amy Miller",
      "phoneNumber": "(801) 555-1437",
      "emailAddress": "amy@hpc.com",
      "defaultAccount": "chemistry",
      "description": "",
      "id": "amy"
    },
        {
      "active": true,
      "commonName": "Robert Taylor",
      "phoneNumber": "(801) 555-1474",
      "emailAddress": "bob@hpc.com",
      "defaultAccount": "biology",
      "description": "",
      "id": "bob"
    },
        {
      "active": true,
      "commonName": "David Jones",
      "phoneNumber": "(801) 555-1436",
      "emailAddress": "dave@hpc.com",
      "defaultAccount": "film",
      "description": "",
      "id": "dave"
    }
  ]
}
```

Get single user

URLs and parameters

```
GET http://localhost:8080/mws/rest/accounting/users/<id>?api-version=3&proxy-
user=<user>[&fields=<fields_to_display>|&show-all=(true|false)]
```

Parameter	Required	Type	Valid values	Description	Example
id	Yes	String	--	The unique identifier of the object	--
fields	No	String	--	Comma-separated list of field names to display.	`fields=name,defaultAccount`
show-all	No	Boolean	`true` or `false`	`true` shows all fields including metadata and hidden fields. Default is `false`.	`show-all=true`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/accounting/users/amy?api-version=3&proxy-
user=moab&pretty=true
------------------------------------

{
  "active": true,
  "commonName": "Amy Miller",
  "phoneNumber": "(801) 555-1437",
  "emailAddress": "amy@hpc.com",
  "defaultAccount": "chemistry",
  "description": "",
  "id": "amy"
}
```

Related topics

- **Fields: Users** on page 1834
- **Resources introduction** on page 1424

Credentials

This section describes behavior of the **Credential** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The Credential API is new with *API version 2*. The supported methods table below requires each resource to be accessed with a URL parameter of `api-version=3`.

For more information, see **Requesting specific API versions** on page 1406.

The **Fields: Credentials** reference contains the type and description of all fields in the **Credential** object.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/credentials/accounts	**Get all account credentials** **Get single account credential**	**Modify account credentials on page 1497**	--	--
/rest/credentials/classes	**Get all class credentials** **Get single class credential**	**Modify class credentials on page 1497**	--	--
/rest/credentials/groups	**Get all group credentials** **Get single group credential**	**Modify group credentials on page 1498**	--	--
/rest/credentials/qoses	**Get all QoS credentials** **Get single QoS credential**	**Modify QoS credentials on page 1498**	--	--
/rest/credentials/users	**Get all user credentials** **Get single user credential**	**Modify User credentials on page 1498**	--	--

This topic contains these sections:

Getting credentials

The HTTP GET method is used to retrieve **Resource Type** information.

Quick reference

```
GET http://localhost:8080/mws/rest/credentials/accounts[/<name>]?api-version=3
GET http://localhost:8080/mws/rest/credentials/classes[/<name>]?api-version=3
GET http://localhost:8080/mws/rest/credentials/groups[/<name>]?api-version=3
GET http://localhost:8080/mws/rest/credentials/qoses[/<name>]?api-version=3
GET http://localhost:8080/mws/rest/credentials/users[/<name>]?api-version=3
```

Get all account credentials

URLs and parameters

```
GET http://localhost:8080/mws/rest/credentials/accounts?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/credentials/accounts?api-version=3
------------------------------------

{
  "totalCount": 1,
  "resultCount": 1,
  "results":   [
    {
      "name": "Administration",
      "account_access_list": ["Administration"],
      "default_account": "Administration",
      "qos_access_list":   [
        "qos1",
        "qos2"
      ],
      "default_qos": "qos1",
      "partition_access_list":   [
        "partition1",
        "SHARED"
      ],
      "default_partition": "partition1",
      "target_type": "CEILING",
      "target": 18.43,
      "priority": 53,
      "max_job_duration_in_seconds": 234,
      "max_idle_jobs": 42,
      "max_jobs": 523,
      "max_processors": 4,
      "max_processor_seconds": 525,
      "max_nodes": 75,
      "reservation": "system.1",
      "user_access_list": ["adaptive"]
    }
   ]
}
```

Get single account credential

URLs and parameters

```
GET http://localhost:8080/mws/rest/credentials/accounts/<name>?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/credentials/accounts/Administration?api-version=3
------------------------------------

{
  "name": "Administration",
  "account_access_list": ["Administration"],
  "default_account": "Administration",
  "qos_access_list":   [
    "qos1",
    "qos2"
  ],
  "default_qos": "qos1",
  "partition_access_list":   [
    "partition1",
    "SHARED"
  ],
  "default_partition": "partition1",
  "target_type": "CEILING",
  "target": 18.43,
  "priority": 53,
  "max_job_duration_in_seconds": 234,
  "max_idle_jobs": 42,
  "max_jobs": 523,
  "max_processors": 4,
  "max_processor_seconds": 525,
  "max_nodes": 75,
  "reservation": "system.1",
  "user_access_list": ["adaptive"]
}
```

Get all class credentials

URLs and parameters

```
GET http://localhost:8080/mws/rest/credentials/classes?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/credentials/classes?api-version=3
------------------------------------

{
  "totalCount": 1,
  "resultCount": 1,
  "results":   [
    {
      "name": "highprio",
      "account_access_list": ["Administration"],
      "default_account": "Administration",
      "qos_access_list":   [
        "qos1",
        "qos2"
      ],
      "default_qos": "qos1",
      "partition_access_list":   [
        "partition1",
        "SHARED"
      ],
      "default_partition": "partition1",
      "target_type": "CEILING",
      "target": 18.43,
      "priority": 53,
      "max_job_duration_in_seconds": 234,
      "max_idle_jobs": 42,
      "max_jobs": 523,
      "max_processors": 4,
      "max_processor_seconds": 525,
      "max_nodes": 75,
      "reservation": "system.1",
      "user_access_list": ["adaptive"]
    }
  ]
}
```

Get single class credential

URLs and parameters

```
GET http://localhost:8080/mws/rest/credentials/classes/<name>?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/credentials/classes/highprio?api-version=3
------------------------------------

{
  "name": "highprio",
  "account_access_list": ["Administration"],
  "default_account": "Administration",
  "qos_access_list":   [
    "qos1",
    "qos2"
  ],
  "default_qos": "qos1",
  "partition_access_list":   [
    "partition1",
    "SHARED"
  ],
  "default_partition": "partition1",
  "target_type": "CEILING",
  "target": 18.43,
  "priority": 53,
  "max_job_duration_in_seconds": 234,
  "max_idle_jobs": 42,
  "max_jobs": 523,
  "max_processors": 4,
  "max_processor_seconds": 525,
  "max_nodes": 75,
  "reservation": "system.1",
  "user_access_list": ["adaptive"]
}
```

Get all group credentials

URLs and parameters

```
GET http://localhost:8080/mws/rest/credentials/groups/<name>?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/credentials/groups?api-version=3
------------------------------------

{
  "totalCount": 1,
  "resultCount": 1,
  "results":   [
    {
  "name": "students",
  "account_access_list": ["Administration"],
  "default_account": "Administration",
  "qos_access_list":   [
    "qos1",
    "qos2"
  ],
  "default_qos": "qos1",
  "partition_access_list":   [
    "partition1",
    "SHARED"
  ],
  "default_partition": "partition1",
  "target_type": "CEILING",
  "target": 18.43,
  "priority": 53,
  "max_job_duration_in_seconds": 234,
  "max_idle_jobs": 42,
  "max_jobs": 523,
  "max_processors": 4,
  "max_processor_seconds": 525,
  "max_nodes": 75,
  "reservation": "system.1",
  "user_access_list": ["adaptive"]
}
```

Get single group credential

URLs and parameters

```
GET http://localhost:8080/mws/rest/credentials/groups/<name>?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/credentials/groups/students?api-version=3
------------------------------------

{
  "name": "students",
  "account_access_list": ["Administration"],
  "default_account": "Administration",
  "qos_access_list":   [
    "qos1",
    "qos2"
  ],
  "default_qos": "qos1",
  "partition_access_list":   [
    "partition1",
    "SHARED"
  ],
  "default_partition": "partition1",
  "target_type": "CEILING",
  "target": 18.43,
  "priority": 53,
  "max_job_duration_in_seconds": 234,
  "max_idle_jobs": 42,
  "max_jobs": 523,
  "max_processors": 4,
  "max_processor_seconds": 525,
  "max_nodes": 75,
  "reservation": "system.1",
  "user_access_list": ["adaptive"]
}
```

Get all QoS credentials

URLs and parameters

```
GET http://localhost:8080/mws/rest/credentials/qoses?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/credentials/qoses?api-version=3
------------------------------------

{
  "totalCount": 1,
  "resultCount": 1,
  "results":   [
    {
      "name": "special",
      "account_access_list": ["Administration"],
      "default_account": "Administration",
      "qos_access_list":   [
        "qos1",
        "qos2"
      ],
      "default_qos": "qos1",
      "partition_access_list":   [
        "partition1",
        "SHARED"
      ],
      "default_partition": "partition1",
      "target_type": "CEILING",
      "target": 18.43,
      "priority": 53,
      "max_job_duration_in_seconds": 234,
      "max_idle_jobs": 42,
      "max_jobs": 523,
      "max_processors": 4,
      "max_processor_seconds": 525,
      "max_nodes": 75,
      "reservation": "system.1",
      "user_access_list": ["adaptive"],
      "flags": [
         "DEADLINE",
         "RESERVEALWAYS",
         "DEDICATED"
      ]
      "queue_time_weight": 30,
      "expansion_factor_weight": 40,
      "quality_of_service_priority": 20
    }
   ]
}
```

Get single QoS credential

URLs and parameters

```
GET http://localhost:8080/mws/rest/credentials/qoses/<name>?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/credentials/qoses/special?api-version=3
------------------------------------

{
  "name": "special",
  "account_access_list": ["Administration"],
  "default_account": "Administration",
  "qos_access_list":   [
    "qos1",
    "qos2"
  ],
  "default_qos": "qos1",
  "partition_access_list":   [
    "partition1",
    "SHARED"
  ],
  "default_partition": "partition1",
  "target_type": "CEILING",
  "target": 18.43,
  "priority": 53,
  "max_job_duration_in_seconds": 234,
  "max_idle_jobs": 42,
  "max_jobs": 523,
  "max_processors": 4,
  "max_processor_seconds": 525,
  "max_nodes": 75,
  "reservation": "system.1",
  "user_access_list": ["adaptive"]
  "flags": [
    "DEADLINE",
    "RESERVEALWAYS",
    "DEDICATED"
   ]
   "queue_time_weight": 30,
   "expansion_factor_weight": 40,
   "quality_of_service_priority": 20
}
```

Get all user credentials

URLs and parameters

```
GET http://localhost:8080/mws/rest/credentials/users?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/credentials/users?api-version=3
------------------------------------

{
  "totalCount": 1,
  "resultCount": 1,
  "results":   [
    {
      "name": "root",
      "account_access_list": ["Administration"],
      "default_account": "Administration",
      "qos_access_list":   [
        "qos1",
        "qos2"
      ],
      "default_qos": "qos1",
      "partition_access_list":   [
        "partition1",
        "SHARED"
      ],
      "default_partition": "partition1",
      "target_type": "CEILING",
      "target": 18.43,
      "priority": 53,
      "max_job_duration_in_seconds": 234,
      "max_idle_jobs": 42,
      "max_jobs": 523,
      "max_processors": 4,
      "max_processor_seconds": 525,
      "max_nodes": 75,
      "email": "root@root.com"
    }
   ]
}
```

Get single user credential

URLs and parameters

```
GET http://localhost:8080/mws/rest/credentials/users/<name>?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/credentials/users/root?api-version=3
------------------------------------

{
  "name": "root",
  "account_access_list": ["Administration"],
  "default_account": "Administration",
  "qos_access_list":   [
    "qos1",
    "qos2"
  ],
  "default_qos": "qos1",
  "partition_access_list":   [
    "partition1",
    "SHARED"
  ],
  "default_partition": "partition1",
  "target_type": "CEILING",
  "target": 18.43,
  "priority": 53,
  "max_job_duration_in_seconds": 234,
  "max_idle_jobs": 42,
  "max_jobs": 523,
  "max_processors": 4,
  "max_processor_seconds": 525,
  "max_nodes": 75,
  "email": "root@root.com"
}
```

Modifying credentials

The HTTP PUT method is used to modify **credentials**.

Quick reference

```
PUT http://localhost:8080/mws/rest/credentials/accounts/<name>?api-version=3[&change-
mode=<add|remove|set>]
PUT http://localhost:8080/mws/rest/credentials/classes/<name>?api-version=3[&change-
mode=<add|remove|set>]
PUT http://localhost:8080/mws/rest/credentials/groups/<name>?api-version=3[&change-
mode=<add|remove|set>]
PUT http://localhost:8080/mws/rest/credentials/qoses/<name>?api-version=3[&change-
mode=<add|remove|set>]
PUT http://localhost:8080/mws/rest/credentials/users/<name>?api-version=3[&change-
mode=<add|remove|set>]
```

URL parameters

URL parameters for modifying a credential.

Credentials parameter	Required	Type	Valid values	Description
change-mode	No	String	set (default) add remove	If **set**, replace existing list with the given one. If **add**, add the given field(s) to the existing list. If **remove**, remove the given field(s) from the existing list.

Moab Workload Manager will automatically add SHARED and the value of default_partition to the partition_access_list.

Modify account credentials

URLs and parameters

```
PUT http://localhost:8080/mws/rest/credentials/accounts/<name>?api-version=3[&change-
mode=<add|remove|set>]
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample body

```
PUT http://localhost:8080/mws/rest/credentials/accounts/biology?api-version=3&change-
mode=add
------------------------------------

{
  "qos_access_list":   [
    "qos3",
    "qos4"
  ],
  "max_job_duration_in_seconds": 234
}
```

Modify class credentials

URLs and parameters

```
PUT http://localhost:8080/mws/rest/credentials/classes/<name>?api-version=3[&change-
mode=<add|remove|set>]
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample body

```
PUT http://localhost:8080/mws/rest/credentials/classes/highprio?api-version=3
------------------------------------

{
  "max_idle_jobs": 50,
  "max_jobs": 300
}
```

Modify group credentials

URLs and parameters

```
PUT http://localhost:8080/mws/rest/credentials/groups/<name>?api-version=3[&change-
mode=<add|remove|set>]
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample body

```
PUT http://localhost:8080/mws/rest/credentials/groups/students?api-version=3&change-
mode=set
------------------------------------

{
  "reservation": "system.2",
  "user_access_list": ["tom"]
}
```

Modify QoS credentials

URLs and parameters

```
PUT http://localhost:8080/mws/rest/credentials/qoses/<name>?api-version=3[&change-
mode=<add|remove|set>]
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample body

```
PUT http://localhost:8080/mws/rest/credentials/qoses/special?api-version=3
------------------------------------

{
  "max_processors": 5,
  "max_processor_seconds": 500
}
```

Modify User credentials

URLs and parameters

```
PUT http://localhost:8080/mws/rest/credentials/users/<name>?api-version=3[&change-
mode=<add|remove|set>]
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample body

```
PUT http://localhost:8080/mws/rest/credentials/users/tom?api-version=3
------------------------------------

{
  "email": "tom@root.com"
}
```

Related topics

- **Fields: Credentials** on page 1836
- **Resources introduction** on page 1424

Diagnostics

This section describes additional REST calls that are available for performing diagnostics on Moab Web Services.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/diag/about	**Get version information**	--	--	--
/rest/diag/auth	**Diagnose authentication**	--	--	--
/rest/diag/health/summary	**Get health summary**	--	--	--
/rest/diag/health/detail	**Get health detail**	--	--	--

/rest/diag/ldap is deprecated. All information that was available in that resource is now available in **/rest/diag/health/detail**.

This topic contains these sections:

Get version information

The HTTP GET method is used to retrieve version and build information.

Quick reference

```
GET http://localhost:8080/mws/rest/diag/about?api-version=3
```

URLs and parameters

```
GET http://localhost:8080/mws/rest/diag/about?api-version=3
```

Sample response

The response contains the application suite, version, build date, and revision.


```
{
  "suite": "CLOUD",
  "version": "7.2.2",
  "buildDate": "2013.03.15_13.12.45",
  "revision": "302238e24e327f4aa45ab4c91834216a7fc19d63"
}
```


Diagnose authentication

The HTTP GET method is used to test for proper authentication. This resource is designed to be used as a simple validation of credentials and gives no output besides the response code.

Quick reference

```
GET http://localhost:8080/mws/rest/diag/auth?api-version=3
```

URLs and parameters

```
GET http://localhost:8080/mws/rest/diag/auth?api-version=3
```

Sample response

> A successful result is indicated by the `200` response code while a failure is indicated by a `401` response code.

```
{}
```

Connection health information

The HTTP GET method is used to retrieve health or status information for connections to external systems or software. There are two available resources for health, one that returns simple summary information and another that returns detailed information.

Quick reference

```
GET http://localhost:8080/mws/rest/diag/health/summary?api-version=3
GET http://localhost:8080/mws/rest/diag/health/detail?api-version=3
```

Get health summary

URLs and parameters

```
GET http://localhost:8080/mws/rest/diag/health/summary?api-version=3
```

If the MongoDB connection is down, authenticated resources are not available. While this resource does not possess much detail beyond that of simple connection information, it is still useful as it does not require authentication and therefore can be used to determine connection problems with MongoDB.

Sample response

The response contains the connection health for Moab Workload Manager (MWM), Moab Accounting Manager (MAM), MongoDB, LDAP, ZeroMQ, PAM. A `true` response value indicates that the connection is healthy and available, and a `false` response indicates that the connection is currently down. Likewise, the `mongoConnected` property for Moab signifies the state of the Moab to MongoDB connection. The possible values of this state are `UP`, `DOWN`, `NOT_CONFIGURED` (when the MongoDB server is not configured in Moab), `NOT_SUPPORTED` (when Moab is not compiled with MongoDB support), and `UNKNOWN` (when MWS cannot communicate with Moab).

```
{
  "mam": {"connected": true},
  "mongo": {"connected": true},
  "mwm":   {
    "connected": true,
    "mongoConnected": "UP",
    "zmqConnected": true
  },
  "ldap": {"connected": true},
  "pam": {"connected": true},
  "zmq": {"connected": true},
}
```

Get health detail

URLs and parameters

```
GET http://localhost:8080/mws/rest/diag/health/detail?api-version=3
```

If the MongoDB connection is down, authenticated resources such as this are not available. In this case, using the **Get health summary** instead may be required.

Sample response

The response contains the connection health and information for Moab Workload Manager (MWM), Moab Accounting Manager (MAM), MongoDB, LDAP, ZeroMQ, PAM. A `"connected": true` response value indicates that the connection is healthy and available, and a `false` response indicates that the connection is currently down. Likewise, the `mongoConnected` property for Moab signifies the state of the Moab to MongoDB connection. The possible values of this state are `UP`, `DOWN`, `NOT_CONFIGURED` (when the MongoDB server is not configured in Moab), `NOT_SUPPORTED` (when Moab is not compiled with MongoDB support), and `UNKNOWN` (when MWS cannot communicate with Moab). A message is also present for all down connections except Moab to MongoDB giving a reason for the error state.

```
{
  "mam":   {
    "connected": true,
    "adminUser": "root",
    "host": "mamhost",
    "port": 7741,
    "version": "7.5",
    "message": null
  },
  "mongo":   {
    "connected": true,
    "host": "127.0.0.1",
    "port": 27017,
    "replicaSet": null,
    "databaseName": "mws",
    "username": {},
    "version": "2.4.8",
    "message": null
  },
  "mwm":   {
    "connected": true,
    "adminUser": "root",
    "host": "localhost",
    "port": 42559,
    "version": "7.5",
    "licensedFeatures":     [
      "green",
      "provision",
      "vm"
    ],
    "state": "RUNNING",
    "mongo":     {
      "connected": "UP",
      "credentialsSet": false,
      "host": "myhost",
      "port": 27017
    },
    "zmq":     {
      "connected": true,
      "encryptionStatus": "OFF",
      "port": 5563
    },
    "message": null
  },
  "ldap":   {
    "connected": true,
    "message": null,
    "server": "openldapnis.ac",
    "port": 389,
    "baseDNs": ["dc=testldap,dc=ac"],
    "bindUser": "cn=admin,dc=testldap,dc=ac",
    "directoryType": "OpenLDAP Using InetOrgPerson Schema",
    "securityType": "NONE",
    "userObjectClass": "inetOrgPerson",
    "groupObjectClass": "groupOfNames",
    "ouObjectClass": "organizationalUnit",
    "userMembershipAttribute": null,
    "groupMembershipAttribute": "member",
    "userNameAttribute": "uid"
  },
  "pam":   {
    "connected": true,
```

```
    "authenticationModule": "system-auth",
    "message": "PAM is configured in MWS."
  },
  "zmq":   {
    "connected": true,
    "version": "3.2.3",
    "message": null,
    "mwmSubscriber":     {
      "connected": true,
      "address": "localhost",
      "port": 5563,
      "message": null
    },
    "mwsSubscriber":     {
      "connected": true,
      "address": "localhost",
      "port": 5564,
      "message": null
    },
    "publisher":     {
      "connected": true,
      "address": "*",
      "port": 5564,
      "message": null
    }
  },
}
```


Related topics

- **Resources introduction** on page 1424

Distinct

The **Distinct** resource enables clients to retrieve distinct (unique) values from another MWS resource. For example, a client can request the list of all `featuresReported` across all nodes like this:

```
GET http://localhost:8080/mws/rest/distinct/nodes/featuresReported/?api-version=3
```

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/distinct/<resource>/<field>	Get distinct values	--	--	--

This topic contains these sections:

Get distinct values

The HTTP GET method is used to retrieve **distinct** values from another MWS resource.

URLs and parameters

```
GET http://localhost:8080/mws/rest/distinct/<resource>/<field>?api-version=3
```

Parameter	Required	Type	Valid values	Example
resource	Yes	String	The MWS resource to query.	`nodes`
field	Yes	String	The field for which to return the distinct values.	`featuresReported`
query	No	JSON	Determines the subset of objects from which to retrieve the distinct values.	`query={"states.-powerState": "On"}`

The **Distinct** resource has no access control of its own. Rather, it depends on the access control of the MWS resource being queried.

For example, for a client to run a query like `/rest/distinct/nodes/featuresReported`, it must have `GET` rights on the `Nodes` resource. For more information, see **Access control** on page 1398.

Examples

Example 4-3: Get all `featuresReported` across all nodes

```
http://localhost:8080/mws/rest/distinct/nodes/featuresReported?api-version=3
------------------------------------

{
  "totalCount": 1,
  "resultCount": 1,
  "results": ["vlan1"]
}
```

Example 4-4: Get all available operating system images across all nodes that are powered on

```
http://localhost:8080/mws/rest/distinct/nodes/operatingSystem.imagesAvailable?api-
version=3&query={"states.powerState": "On"}
------------------------------------

{
  "totalCount": 2,
  "resultCount": 2,
  "results":   [
    "linux",
    "windows"
  ]
}
```

Related topics

- **Resources introduction** on page 1424

Events

This section describes the URLs, request bodies, and responses delivered to and from Moab Web Services for handling events.

The Event API is new with API version 3. The supported methods table below requires each resource to be accessed with a URL parameter of `api-version=3` in order to behave as documented.

For more information, see **Requesting specific API versions** on page 1406.

The **Fields: Events** reference contains the type and description of all fields in the **Event** object. It also contains details regarding which fields are valid during POST actions.

Important changes

- The following fields have been renamed in API version 3:

Name in version 1 & 2	Name in version 3
`eventTime`	`eventDate`
`sourceComponent`	`origin`
`errorMessage.message`	`message`
`relatedObjects`	`associatedObjects`

- The following fields have been removed in API version 3.

  MWS will no longer report these fields, even if there are existing events in the database with these fields.

  - `eventCategory`
  - `status`
  - `facility`
  - `initiatedBy`
  - `primaryObject` (Primary objects are now reported in `associatedObjects`.)
  - `errorMessage.originator`
  - `errorMessage.errorCode`
  - `details`

- The following fields are new in API version 3 (see **Fields: Events** on page 1837):
  - `arguments`
  - `code`

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/events	Get all events	--	Create event	--
/rest/events/<id>	Get single event	--	--	--

This topic contains these sections:

Getting events

The HTTP GET method is used to retrieve **Event** information. Queries for all objects and a single object are available.

Quick reference

```
GET http://localhost:8080/mws/rest/events?api-version=3[&query={"field":"value"}&sort=
{"field":<1|-1>}]
GET http://localhost:8080/mws/rest/events/<id>?api-version=3
```

Get all events

URLs and parameters

```
GET http://localhost:8080/mws/rest/events?api-version=3[&query={"field":"value"}&sort=
{"field":<1|-1>}]
```

Parameter	Required	Type	Valid values	Example
query	No	JSON	Query for specific results. It is possible to query events by one or more fields based on MongoDB query syntax.	`query={"sever-ity":"ERROR"}`

Parameter	Required	Type	Valid values	Example
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	`sort={"id":-1}`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/events?api-version=3
------------------------------------

{
   "totalCount":2,
   "resultCount":2,
   "results":[
      {
         "arguments":[

         ],
         "associatedObjects":[
            {
               "type":"VM",
               "id":"vm1"
            }
         ],
         "tenant":
            {
               "id":"1234567890abcdef12345678",
               "name":"Research"
            },
         "code":234881023,
         "eventDate":"2013-06-10 17:13:31 UTC",
         "eventType":"VM Provision",
         "message":null,
         "origin":"CSA Plugin",
         "severity":"INFO",
         "id":"51b6093bc4aa708a5bebb6ae"
      },
      {
         "arguments":[
            "51b608ddc4aa708a5bebb684"
         ],
         "associatedObjects":[
            {
               "type":"Service",
               "id":"51b608ddc4aa708a5bebb684"
            }
         ],
         "tenant":
            {
               "id":"1234567890abcdef12345678",
               "name":"Research"
            },
         "code":33554944,
         "eventDate":"2013-06-10 17:11:59 UTC",
         "eventType":"Service Create",
         "message":"The service '51b608ddc4aa708a5bebb684' was created",
         "origin":"MWS/ServiceEvents/CREATE_1ID",
         "severity":"INFO",
         "id":"51b608dfc4aa708a5bebb686"
      }
   ]
}
```

Querying events

It is possible to query events by one or more fields based on MongoDB query syntax. The following contains examples of simple and complex event queries and event queries by date.

Simple queries:

- To see only events that are of type "Service Create":

```
http://localhost:8080/mws/rest/events?api-version=3&query={"eventType":"Service
Create"}
```

- To see only events of type "Service Create" with the severity of "INFO":

```
http://localhost:8080/mws/rest/events?api-version=3&query={"eventType":"Service
Create","severity":"INFO"}
```

- To see only events with a code of 33554946

```
http://localhost:8080/mws/rest/events?api-version=3&query={code:33554946}
```

More complex queries:

- You can query on embedded JSON objects within the event JSON. For example, to see events associated with service 51b608ddc4aa708a5bebb684:

```
http://localhost:8080/mws/rest/events?api-version=3&query=
{"associatedObjects.id":"51b608ddc4aa708a5bebb684"}
```

- To see only events that are NOT associated with service 51b608ddc4aa708a5bebb684:

```
http://localhost:8080/mws/rest/events?api-version=3&query=
{"associatedObjects.id":{"$ne":"51b608ddc4aa708a5bebb684"}}
```

- When the field values of the desired events are a finite set, you can use the $in operator. For example, to see events that have a severity of either WARN or ERROR:

```
http://localhost:8080/mws/rest/events?api-version=3&query={"severity":{"$in":
["ERROR","WARN"]}}
```

Querying events by date

- To see events created before January 27, 2012 at 12:08 a.m. UTC:

```
http://localhost:8080/mws/rest/events?api-version=3&query={"eventDate":
{"$lt":"2012-01-27 12:08:00 UTC"}}
```

- To see events created before or on January 27, 2012 at 12:08 a.m. UTC:

```
http://localhost:8080/mws/rest/events?api-version=3&query={"eventDate":
{"$lte":"2012-01-27 12:08:00 UTC"}}
```

- To see all events created after January 27, 2012 at 12:04 a.m. UTC:

```
http://localhost:8080/mws/rest/events?api-version=3&query={"eventDate":
{"$gt":"2012-01-27 12:04:00 UTC"}}
```

- To see all events created after or on January 27, 2012 at 12:04 a.m. UTC:

```
http://localhost:8080/mws/rest/events?api-version=3&query={"eventDate":
{"$gte":"2012-01-27 12:04:00 UTC"}}
```

- To see events created between 12:04 a.m. and 12:08 a.m. UTC inclusive:

```
http://localhost:8080/mws/rest/events?api-version=3&query={"eventDate":
{"$gte":"2012-01-27 12:04:00 UTC","$lte":"2012-01-27 12:08:00 UTC"}}
```

- To see events created between 12:04 a.m. and 12:08 a.m. UTC inclusive that have a severity of ERROR:

```
http://localhost:8080/mws/rest/events?api-version=3&query=
{"severity":"ERROR","eventDate":{"$gte":"2012-01-27 12:04:00 UTC","$lte":"2012-
01-27 12:08:00 UTC"}}
```

Sorting

See the sorting section of **Global URL parameters** on page 1403.

Limiting the number of results

- If you want to limit the number of results of events, you can use the `max` parameter. For example, to see only 10 "VM Provision" events:

```
http://localhost:8080/mws/rest/events?api-version=3&query={"eventType":"VM
Provision"}&sort={"eventDate":1}&max=10
```

- To see "VM Provision" events 51-60 when sorted by eventDate in descending order, you can combine `max` with `offset`, as follows:

```
http://localhost:8080/mws/rest/events?api-version=3&query={"eventType":"VM
Provision"}&sort={"eventDate":-1}&max=10&offset=50
```

Get single event

URLs and parameters

```
GET http://localhost:8080/mws/rest/events/<id>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/events/51b608dfc4aa708a5bebb686?api-version=3
------------------------------------

{
  "arguments": ["51b608ddc4aa708a5bebb684"],
  "associatedObjects": [  {
    "type": "Service",
    "id": "51b608ddc4aa708a5bebb684"
  }],
  "tenant":
    {
      "id":"1234567890abcdef12345678",
      "name":"Research"
    },
  "code": 33554944,
  "eventDate": "2013-06-10 17:11:59 UTC",
  "eventType": "Service Create",
  "message": "The service '51b608ddc4aa708a5bebb684' was created",
  "origin": "MWS/ServiceEvents/CREATE_1ID",
  "severity": "INFO",
  "id": "51b608dfc4aa708a5bebb686"
}
```

Creating events

The HTTP POST method is used to create an **Event**.

Quick reference

```
POST http://localhost:8080/mws/rest/events?api-version=3
```

Create event

URLs and parameters

```
POST http://localhost:8080/mws/rest/events?api-version=3
```

Request body

```
POST http://localhost:8080/mws/rest/events?api-version=3 Content-Type:application/json
------------------------------------

{
      "arguments": ["vm1"],
      "associatedObjects": [      {
        "type": "VM",
        "id": "vm1"
      }],
      "code": 234881023,
      "eventDate": "2013-06-10 17:13:31 UTC",
      "eventType": "VM Provision",
      "message": "The virtual machine \"vm1\" was provisioned",
      "origin": "CSA Plugin",
      "severity": "INFO"
    }
```

An event's `tenant` is automatically inherited from the `associatedObjects`.

Sample response

If the request was successful, the response will be an object with an **id** property containing the ID of the newly created events. On failure, the response is an error message.

```
JSON response
------------------------------------

{"arguments":["vm1"],"associatedObjects":[{"_
id":"vm1","id":"vm1","type":"VM","version":0}],"code":234881023,"eventDate":"2013-06-
10 17:13:31 UTC","eventType":"VM
Provision","id":"51b62046c4aa708a5bebc018","message":"The virtual machine vm1 was
provisioned","origin":"CSA Plugin","severity":"INFO","version":0}
```

Below is an example of `events.log` output for a successful event request:

```
2013-06-10T11:13:31.000-06:00 severity="INFO" code="0x0dffffff" type="VM Provision"
origin="CSA Plugin" associatedObject.0.type="VM" associatedObject.0.id="vm1"
arguments=["vm1"] message="The virtual machine \"vm1\" was provisioned"
```

Note that " (double quote) characters in the input have been replaced by \" characters in the output. (For other character restrictions, see **Restrictions** on page 1513.)

Restrictions

Special characters—such as newline, carriage return, and " (double quote) characters—are encoded in the output of `events.log` to make `events.log` easy to parse with scripts and third party tools. For example, if the input XML contains:

```
<ErrorMessage>RM says, "Cannot provision vm21"</ErrorMessage>
```

Then the following will be output to `events.log`:

```
error.message="RM says, \"Cannot provision vm21\""
```

(Notice that " has been replaced with \".)

This table contains the most common encodings. (For more information, see escape sequences for Java Strings.)

Character	Escape sequence
" (double quote)	\"
\ (backslash)	\\
newline	\n

Character	Escape sequence
carriage return	\r
tab	\t

Other restrictions include:

- **origin**, **eventType**, **associatedObject.id**, and **associatedObject.type** cannot contain single quotes (') or double quotes (").

Related topics

- **Resources introduction** on page 1424
- **Notifications** on page 1563
- **Fields: Notifications** on page 2028
- **Notification conditions** on page 1558
- **Fields: Notification Conditions** on page 2024
- **Fields: Events** on page 1837
- **System events** on page 1422
- **Creating events and notifications** on page 1682
- **Plugin event service** on page 1733
- **Handling events** on page 1689
- **Securing the connection with the message queue** on page 1395

Images

This section describes behavior of the **Image** object in Moab Web Services. An image resource is used to track the different types of operating systems and hypervisors available in the data center. It also tracks which virtual machines are available on the hypervisors. This section describes the URLs, request bodies, and responses delivered to and from MWS.

> The **Fields: Images** reference contains the type and description of all fields in the **Image** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/images	**Get all images**	--	**Create single image**	--

Resource	GET	PUT	POST	DELETE
/rest/images/<id>	Get single image	Modify single image	--	Delete single image
/rest/images/<name>	Get single image	Modify single image	--	Delete single image

This topic contains these sections:

Getting images

The HTTP GET method is used to retrieve **Image** information. You can query all objects or a single object.

Quick reference

```
GET http://localhost:8080/mws/rest/images?api-version=3[&query={"field":"value"}&sort=
{"field":<1|-1>}]
GET http://localhost:8080/mws/rest/images/<id>?api-version=3
GET http://localhost:8080/mws/rest/images/<name>?api-version=3
```

Get all images

URLs and parameters

```
GET http://localhost:8080/mws/rest/images?api-version=3[&query={"field":"value"}&sort=
{"field":<1|-1>}]
```

Parameter	Required	Type	Description	Example
query	No	JSON	Queries for specific results. It is possible to query images by one or more fields based on MongoDB query syntax.	`query={"type":"stateful","osType":"linux"}`
sort	No	JSON	Sort the results. Use 1 for ascending and -1 for descending.	`sort={"name":-1}`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/images?api-version=3&fields=id,name
------------------------------------

{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
    "id": "4fa197e68ca30fc605dd1cf0",
    "name": "centos5-stateful"
  }]
}
```

Sorting and querying

See the sorting and querying sections of **Global URL parameters** on page 1403.

Get single image

URLs and parameters

```
GET http://localhost:8080/mws/rest/images/<id>?api-version=3
GET http://localhost:8080/mws/rest/images/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the Image.
name	Yes	String	--	The name of the Image.

You must specify either **id** or **name**, but you do not have to specify both.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

Virtual machine image example:

```
GET http://localhost:8080/mws/rest/images/centos5-compute-stateful?api-version=3
------------------------------------

{
  "active":true,
  "extensions":{
    "xcat":{
      "os":"centos",
      "architecture":"x86_64",
      "profile":"compute"
    }
  },
  "features":[],
  "hypervisor":false,
  "hypervisorType": null,
  "id":"4fa197e68ca30fc605dd1cf0",
  "name":"centos5-compute-stateful",
  "osType":"linux",
  "supportsPhysicalMachine":false,
  "supportsVirtualMachine":true,
  "templateName":null,
  "type":"stateful",
  "version":0,
  "virtualizedImages":[]
}
```

Hypervisor image example:

```
GET http://localhost:8080/mws/rest/images/esxi-4.1-stateful?api-version=3
------------------------------------

{
  "active":true,
  "extensions":{
    "xcat":{
      "hvGroupName":"hvGroup",
      "vmGroupName":"vmGroup",
      "os":"esxi-4.1",
      "architecture":"x86_64",
      "profile":"hv"
    }
  },
  "features":[],
  "hypervisor":true,
  "hypervisorType":"ESX",
  "id":"4fa197e68ca30fc605dd1cf0",
  "name":"centos5-compute-stateful",
  "osType":"linux",
  "supportsPhysicalMachine":true,
  "supportsVirtualMachine":false,
  "templateName":null,
  "type":"stateful",
  "version":0,
  "virtualizedImages":[]
}
```

The `version` field contains the current version of the database entry and does not reflect the version of the operating system. For more information, see **Modify single image** on page 1521.

Creating images

The HTTP POST method is used to submit **Images**.

Quick reference

```
POST http://localhost:8080/mws/rest/images?api-version=3
```

Create single image

URLs and parameters

```
POST http://localhost:8080/mws/rest/images?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

Three fields are required to submit an image: **name**, **hypervisor**, and **osType**. Each image must also support provisioning to either a physical machine or a virtual machine by using the **supportsPhysicalMachine** or **supportsVirtualMachine** fields.

The **name** field must contain only letters, digits, periods, dashes, and underscores.

The array of virtualized images are themselves objects that contain image IDs or names. For more information on available fields and types, see **Fields: Images** on page 1843.

The following is an example of the most basic image that can be created:

```
POST http://localhost:8080/mws/rest/images?api-version=3
------------------------------------

{
  "name": "centos5-stateful",
  "osType": "linux",
  "hypervisor": false,
  "supportsVirtualMachine":true
}
```

Note that this example does not provide any information for a provisioning manager (such as xCAT) to actually provision the machine. In order to provide this, you must add an entry to the **extensions** field that contains provisioning manager-specific information. Each key in the extensions field corresponds to the provisioning manager, and certain properties are required based on this key. For example, the xCAT extension key must be named `xcat` and must contain certain fields. These extension keys are documented in **Fields: Images** on page 1843. See the following examples of creating images with xCAT-specific provisioning information below.

Sample response

If the request was successful, the response body is the new image that was created exactly as shown in **Get single image**. On failure, the response is an error message.

Samples

The **virtualizedImages** field only accepts input when the image is a hypervisor and expects an array of image IDs *or* names, as shown in the following example:

```
Example payload of hypervisor with 2 vms
------------------------------------

{
  "hypervisor":true,
  "name":"esx5-stateful",
  "osType":"linux",
  "supportsPhysicalMachine":true,
  "type":"stateful",
  "hypervisorType":"ESX",
  "virtualizedImages":   [
    {"id": "4fa197e68ca30fc605dd1cf0"},
    {"name": "centos5-stateful"}
  ]
}
```

The following example shows how to create an image that utilizes a cloned template for a virtual machine. (Note that the **type** must be set to `linkedclone` in order to set the **templateName** field.)

```
VM Utilizing a Cloned Template
------------------------------------

{
  "active": true,
  "hypervisor": false,
  "name": "centos5-compute-stateful",
  "osType": "linux",
  "type": "linkedclone",
  "supportsVirtualMachine":true,
  "templateName":"centos5-compute"
}
```

The following are samples of a virtual machine and a hypervisor image that can be provisioned with xCAT:

```
xCAT Virtual Machine Image
------------------------------------

{
  "active": true,
  "features": [],
  "hypervisor": false,
  "name": "centos5-compute-stateful",
  "osType": "linux",
  "type": "stateful",
  "supportsVirtualMachine":true,
  "extensions": {
    "xcat": {
      "os": "centos",
      "architecture": "x86_64",
      "profile": "compute"
    }
  }
}
```

```
xCAT Hypervisor Image
-----------------------------------

{
  "active": true,
  "features": [],
  "hypervisor": true,
  "name": "esxi5-base-stateless",
  "osType": "linux",
  "virtualizedImages":   [
    {"name": "centos5-compute-stateless"}
  ],
  "type": "stateless",
  "hypervisorType": "ESX",
  "supportsPhysicalMachine":true,
  "extensions": {
    "xcat": {
      "os": "esxi5",
      "architecture": "x86_64",
      "profile": "base",
      "hvType": "esx",
      "hvGroupName": "esx5hv",
      "vmGroupName": "esx5vm"
    }
  }
}
```

Modifying images

The HTTP PUT method is used to modify **Images**.

Quick reference

```
PUT http://localhost:8080/mws/rest/images/<id>?api-version=3
PUT http://localhost:8080/mws/rest/images/<name>?api-version=3
```

Modify single image

URLs and parameters

```
PUT http://localhost:8080/mws/rest/images/<id>?api-version=3
PUT http://localhost:8080/mws/rest/images/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the Image.
name	Yes	String	--	The name of the Image.

You must specify either **id** or **name**, but you do not have to specify both. The **name** field must contain only letters, digits, periods, dashes, and underscores.

See **Global URL parameters** on page 1403 for available URL parameters.

Example request

```
PUT http://locahost/mws/rest/images/centos5-stateful?api-version=3
------------------------------------

{
  "name": "centos5-stateful",
  "type": "stateful",
  "hypervisor": false,
  "osType": "linux",
  "virtualizedImages": []
}
```

The **version** field contains the current version of the database entry and does not reflect the version of the operating system. This field cannot be updated directly. However, if **version** is included in the modify request, it will be used to verify that another client did not update the object in between the time the data was retrieved and the modify request was delivered.

Sample response

If the request was successful, the response body is the modified image as shown in **Get single image**. On failure, the response is an error message.

Deleting images

The HTTP DELETE method is used to delete **Images**.

Quick reference

```
DELETE http://localhost:8080/mws/rest/images/<id>?api-version=3
DELETE http://localhost:8080/mws/rest/images/<name>?api-version=3
```

Delete single image

URLs and parameters

```
DELETE http://localhost:8080/mws/rest/images/<id>?api-version=3
DELETE http://localhost:8080/mws/rest/images/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the Image.
name	Yes	String	--	The name of the Image.

Only one of **id** or **name** are required.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON Response
------------------------------------

{}
```

Related topics

- **Fields: Images** on page 1843
- **Resources introduction** on page 1424

Job arrays

This section describes behavior of the **Job Array** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The **Fields: Job Arrays** reference section contains the type and description of all fields in the **Job Array** object.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/job-arrays	--	--	**Submit job array**	--

This topic contains these sections:

Submitting job arrays

The HTTP POST method is used to submit **Job Arrays**.

Quick reference

```
POST http://localhost:8080/mws/rest/job-arrays?api-version=3[&proxy-user=<username>]
```

While the `Job Array` resource only gives access to create job arrays, job arrays are retrieved using the operations in **Getting job information** on page 1526.

Restrictions

All restrictions present for **Submitting jobs** are present for job arrays. In addition, job arrays are *only* supported if the `ENABLEJOBARRAYS` parameter is set to `TRUE` in the `moab.cfg` file. For example:

```
ENABLEJOBARRAYS          TRUE
```

Submit job array

URLs and parameters

```
POST http://localhost:8080/mws/rest/job-arrays?api-version=3[&proxy-user=<username>]
```

Parameter	Required	Type	Valid values	Description
proxy-user	No	String	--	Perform this action as this user.

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

To submit a job array, only two fields are required: `jobPrototype` and one of `indexValues` or `indexRanges`. Both index ranges and values may be specified if desired.

The request body below shows all the fields that are available during job array submission, although the `jobPrototype` shown is a simple example and does not utilize all fields of a job submission.

The `jobPrototype` field has the same properties as a typical job submission. Consequently the api-version of the job array will apply to the `jobPrototype` like it does when you submit jobs, so the api-version in the call must match the api-version of the job. Examples of this can be seen in **Submitting jobs** on page 1533.

```
JSON request body
------------------------------------

{
  "name": "myarray",
  "indexRanges": [  {
    "startIndex": 11,
    "endIndex": 25,
    "increment": 2
  }],
  "indexValues":   [ 2, 4, 6, 8, 10 ],
  "slotLimit": 2,
  "cancellationPolicy":   {
    "firstJob": "FAILURE",
    "anyJob": "SUCCESS"
  },
  "jobPrototype":   {
    "commandFile": "/tmp/test.sh",
    "initialWorkingDirectory": "/tmp",
    "requirements": [{"taskCount": 4}]
  }
}
```

Sample response

The response of this task is the same as submitting a job (see **Submit job** on page 1534).

Related topics

- **Fields: Job Arrays** on page 1851
- **Resources introduction** on page 1424
- **Jobs** on page 1525
- **Job templates** on page 1547

Jobs

This section describes behavior of the **Job** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The Job API is new with *API version 2*. The supported methods table below requires each resource to be accessed with a URL parameter of `api-version=3` in order to behave as documented.

For more information, see **Requesting specific API versions** on page 1406.

The **Fields: Jobs** reference contains the type and description of all fields in the **Job** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/jobs	Get all jobs	--	Submit job	--
/rest/jobs/<name>	Get single job	Modify job attributes	--	Cancel job
/rest/jobs/<name>/<modifyAction>	--	Perform actions on job	--	--

This topic contains these sections:

Getting job information

The HTTP GET method is used to retrieve **Job** information.

Quick reference

```
GET http://localhost:8080/mws/rest/jobs/<name>?api-version=3
```

Get all jobs

URLs and parameters

```
GET http://localhost:8080/mws/rest/jobs?api-version=3
```

Parameter	Required	Type	Description	Example
query	No	JSON	Queries for specific results. It is possible to query by one or more fields based on MongoDB query syntax.	`query={"isAct-ive":true}`
sort	No	JSON	Sort the results. Use 1 for ascending and -1 for descending.	`sort={"name":-1}`

See **Global URL parameters** on page 1403 for available URL parameters.

How to get all jobs

```
GET http://localhost:8080/mws/rest/jobs?api-version=3&fields=name,flags&max=3
------------------------------------

{
  "totalCount": 8,
  "resultCount": 3,
  "results":   [
        {
      "flags": ["GLOBALQUEUE"],
      "name": "Moab.1"
    },
        {
      "flags": ["GLOBALQUEUE"],
      "name": "Moab.2"
    },
        {
      "flags": ["GLOBALQUEUE"],
      "name": "Moab.4"
    }
  ]
}
```

How to get a subset of jobs

```
Get active jobs
------------------------------------

http://localhost:8080/mws/rest/jobs?api-version=3&query={"isActive":true}
```

```
Get completed jobs
------------------------------------

http://localhost:8080/mws/rest/jobs?api-version=3&query={"isActive":false}
```

```
Get jobs owned by a particular user
------------------------------------

http://localhost:8080/mws/rest/jobs?api-version=3&query={"credentials.user":"fred"}
```

Known issues

Some jobs are not returned if `DisplayFlags UseBlocking` is set in the `moab.cfg` file.

Get single job

URLs and parameters

```
GET http://localhost:8080/mws/rest/jobs/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
name	Yes	String	--	The name of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

> The `attributes` field is only applicable in API version 2 and later, and the `MOAB_TENANT` field only applies if the job is attached to a tenant.

Sample response

```
JSON response
------------------------------------

{
  "arrayIndex": null,
  "arrayMasterName": null,
  "attributes": [],
  "blocks": [  {
    "category": "jobBlock",
    "message": null,
    "type": null
  }],
  "bypassCount": 0,
  "cancelCount": 0,
  "commandFile": "/tmp/test.sh",
  "commandLineArgs": null,
  "completionCode": null,
  "cpuTime": 0,
  "credentials":   {
    "account": null,
    "group": "adaptive",
    "jobClass": null,
    "qos": "NONE",
    "qosRequested": null,
    "user": "adaptive"
  },
  "customName": null,
  "dates":   {
    "completedDate": null,
    "createdDate": "2012-10-11 17:58:16 UTC",
    "deadlineDate": "2037-10-24 12:26:40 UTC",
    "dispatchedDate": null,
    "earliestRequestedStartDate": null,
    "earliestStartDate": "2012-10-11 17:58:18 UTC",
    "eligibleDate": "2012-10-11 17:59:19 UTC",
    "lastCanceledDate": null,
    "lastChargedDate": null,
    "lastPreemptedDate": null,
    "lastUpdatedDate": "2012-10-11 17:59:19 UTC",
    "startDate": null,
    "submitDate": "2012-10-11 17:58:16 UTC",
    "terminationDate": "2037-10-24 12:26:40 UTC"
  },
  "deferCount": 0,
  "dependencies": [],
  "description": null,
  "duration": 8639999,
  "durationActive": 0,
  "durationQueued": 31,
  "durationRemaining": 0,
  "durationSuspended": 0,
  "emailNotifyAddresses": [],
  "emailNotifyTypes": [],
  "environmentRequested": false,
  "environmentVariables": {},
  "epilogScript": null,
  "flags": ["GLOBALQUEUE"],
  "holdDate": null,
  "holdReason": null,
  "holds": [],
  "initialWorkingDirectory": "/tmp",
  "isActive": true,
```

```
  "jobGroup": null,
  "masterNode": null,
  "memorySecondsDedicated": 0,
  "memorySecondsUtilized": 0,
  "messages": [],
  "migrateCount": 0,
  "minimumPreemptTime": 0,
  "mwmName": "Moab",
  "name": "Moab.15",
  "nodesExcluded": [],
  "nodesRequested": [],
  "nodesRequestedPolicy": null,
  "partitionAccessList":   [
    "msm",
    "SHARED"
  ],
  "partitionAccessListRequested":   [
    "msm",
    "SHARED"
  ],
  "preemptCount": 0,
  "priorities":   {
    "run": 0,
    "start": 1,
    "system": 0,
    "user": 0
  },
  "processorSecondsDedicated": 0,
  "processorSecondsLimit": 0,
  "processorSecondsUtilized": 0,
  "prologScript": null,
  "queueStatus": "blocked",
  "rejectPolicies": [],
  "requirements": [  {
    "architecture": null,
    "attributes":{
      "matlab":   [
             {
            "restriction":"must",
            "comparator": "<=",
            "value": "7.1",
            "displayValue": null
         }
      ],
      "MOAB_TENANT":   [   {
            "value": "1234567890aabbccddeeff00",
            "displayValue": "ResearchGroup"
      }],
      "soffice":   [
             {
            "restriction":"must",
            "comparator": "%=",
            "value": "3.1",
            "displayValue": null
         }
      ]
    },
    "features": [],
    "index": 0,
    "featuresRequested": [],
    "featuresRequestedMode": "AND",
    "featuresExcluded": [],
```

```
    "featuresExcludedMode": "AND",
    "metrics": {},
    "nodeAccessPolicy": null,
    "nodeAllocationPolicy": null,
    "nodeCount": 0,
    "nodes": [],
    "nodeSet": null,
    "image": null,
    "reservation": null,
    "resourcesPerTask":     {
      "processors":       {
        "dedicated": 1,
        "utilized": 0
      },
      "memory":       {
        "dedicated": 0,
        "utilized": 0
      },
      "disk":       {
        "dedicated": 0,
        "utilized": null
      },
      "swap":       {
        "dedicated": 0,
        "utilized": null
      }
    },
    "taskCount": 4,
    "tasksPerNode": 0
  }],
  "reservationRequested": null,
  "resourceFailPolicy": null,
  "resourceManagerExtension": null,
  "resourceManagers": [  {
    "isDestination": false,
    "isSource": true,
    "jobName": "Moab.15",
    "name": "internal"
  }],
  "rmStandardErrorFilePath": null,
  "rmStandardOutputFilePath": null,
  "standardErrorFilePath": null,
  "standardOutputFilePath": null,
  "startCount": 0,
  "states":   {
    "state": "Idle",
    "stateExpected": "Idle",
    "stateLastUpdatedDate": null,
    "subState": null
  },
  "submitHost": "0:0:0:0:0:0:0:1",
  "systemJobAction": null,
  "systemJobType": null,
  "targetedJobAction": null,
  "targetedJobName": null,
  "templates": [{"name": "DEFAULT"}],
  "triggers": [],
  "variables": {},
  "virtualContainers": [],
  "virtualMachines": [],
  "vmUsagePolicy": null
}
```

Job arrays

- If a job is the master of a job array, the response will have some additional fields set as shown in the following example. The **name** field is chosen by the Moab, and the **customName** field comes from the **Fields: Job Arrays** `name` field.

```
Job array master
------------------------------------

{
  "name": "Moab.5",
  "customName": "myarray",
  "flags":   [
    "ARRAYMASTER",
    "GLOBALQUEUE",
    "CANCELONFIRSTFAILURE",
    "CANCELONANYSUCCESS"
  ]
}
```

- If a job is a sub-job of an array, the response will have other fields set as shown in the following example.

```
Array sub-job
------------------------------------

{
  "name": "Moab.5[21]",
  "customName": "myarray",
  "arrayIndex": 21,
  "arrayMasterName": "Moab.5",
  "flags":   [
    "ARRAYJOB",
    "GLOBALQUEUE",
    "CANCELONFIRSTFAILURE",
    "CANCELONANYSUCCESS"
  ]
}
```

Submitting jobs

The HTTP POST method is used to submit **Jobs**.

Quick reference

```
POST http://localhost:8080/mws/rest/jobs?api-version=3[&proxy-user=<username>]
```

Restrictions

- No more than one virtual container can be specified in the request. The virtual container must already exist.
- The `credentials.user` and `credentials.group` properties are used to submit a job as the specified user belonging to the specified group.

- Job `variables` have the following restrictions:
  - Variable names cannot contain equals (=), semicolon (;), colon (:), plus (+), question mark (?), caret (^), backslash (\), or white space.
  - Variable values cannot contain semicolon (;), colon (:), plus (+), or caret (^).
- When submitting jobs, the only supported `hold` type is `User`.
- The proxy-user parameter is ignored unless you set `ENABLEPROXY=TRUE` in the `moab.cfg` file. For example:

  ```
  ADMINCFG[1]            USERS=root,ted ENABLEPROXY=TRUE
  ```

- When the JOBNODEMATCHPOLICY is set to EXACTNODE in Moab, avoid using requirements.tasksPerNode. Use requirements.resourcesPerTask.processors.dedicated instead.

Submit job

URLs and parameters

```
POST http://localhost:8080/mws/rest/jobs?api-version=3[&proxy-user=<username>]
```

Parameter	Required	Type	Valid values	Description
proxy-user	No	String	--	Perform the action as this user.

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

```
JSON request body (specified host list)
------------------------------------

{
  "attributes":   [
    "attr1",
    "attr2"
  ],
  "commandFile": "/tmp/test.sh",
  "commandScript": "c2xlZXAgNjAK",
  "commandLineArguments": "-x -v",
  "credentials":   {
    "account": "account",
    "group": "group",
    "jobClass": "BATCH",
    "qosRequested": "QOS1",
    "user": "saadmin"
  },
  "customName": "custom name for job",
  "dates":   {
    "earliestRequestedStartDate": "2012-11-08 13:18:47 UTC",
    "deadlineDate": "2014-02-17 14:00:00 UTC"
  },
  "dependencies":   [
        {
      "type": "set",
      "name": "vc1.varA"
    },
        {
      "type": "set",
      "name": "vc2.varB"
    },
        {
      "type": "set",
      "name": "vc3.varC"
    }
  ],
  "duration": 600,
  "emailNotifyAddresses":   [
    "user3@ac.com",
    "user4@ac.com"
  ],
  "emailNotifyTypes":   [
    "JobStart",
    "JobEnd"
  ],
  "environmentRequested": true,
  "environmentVariables":   {
    "var1": "val1",
    "var2": "val2"
  },
  "epilogScript": "/tmp/epilog.sh",
  "flags":   [
    "RESTARTABLE",
    "SUSPENDABLE"
  ],
  "holds": ["User"],
  "initialWorkingDirectory": "/tmp",
  "jobGroup": "job_group",
  "nodesExcluded":   [
    {"name": "node07"},
    {"name": "node08"}
```

```
  ],
  "nodesRequested":   [
    {"name": "node01"},
    {"name": "node02"}
  ],
  "nodesRequestedPolicy": "SUBSET",
  "partitionAccessListRequested":   [
    "p1",
    "p2"
  ],
  "priorities": {"user": 5},
  "prologScript": "/tmp/prolog.sh",
  "requirements": [  {
    "architecture": "x86_64",
    "attributes":{
      "matlab":   [
             {
            "restriction":"must",
            "comparator": "<=",
            "value": "7.1"
         }
      ],
      "soffice":   [
             {
            "restriction":"must",
            "comparator": "%=",
            "value": "3.1"
         }
      ]
    },
    "featuresRequested":     [
      "a",
      "b",
      "c"
    ],
    "featuresRequestedMode": "OR",
    "featuresExcluded":     [
      "d",
      "e",
      "f"
    ],
    "featuresExcludedMode": "AND",
    "nodeAccessPolicy": "SINGLEJOB",
    "nodeAllocationPolicy": "PRIORITY",
    "nodeCount": 6,
    "nodeSet":"FIRSTOF:FEATURE:vlan2",
    "image": "linux",
    "resourcesPerTask":     {
      "disk": {"dedicated": 1024},
      "memory": {"dedicated": 512},
      "processors": {"dedicated": 2},
      "swap": {"dedicated": 4096},
      "matlab": {"dedicated": 6},
      "intellij": {"dedicated": 2}
    },
    "taskCount": 4,
    "tasksPerNode": 14
  }],
  "reservationRequested": {"name": "rsv.1"},
  "resourceFailPolicy": "RETRY",
  "resourceManagerExtension": "x=PROC=4",
  "standardErrorFilePath": "/tmp/error",
```

```
  "standardOutputFilePath": "/tmp/out",
  "submitHost": "admin-node",
  "templates":   [
    {"name": "template1"},
    {"name": "template2"}
  ],
  "variables":   {
    "var1": "val1",
    "var2": "val2"
  },
  "virtualContainers": [{"name": "vc1"}],
  "vmUsagePolicy": "CREATEVM"
}
```

Sample response

The response of this task is one of three possibilities:

- An object with a single **messages** property containing a list of error messages on failure.

```
{"messages":["Could not create job - invalid requirements"]}
```

- An object with a **name** property containing the name of the newly created job.

```
{"name":"Moab.1"}
```

- An object with a **name** property and a **virtualContainers** list containing the name of the newly created virtual container.

```
{ "name": "Moab.1", "virtualContainers": [{"name": "vc1"}] }
```

> The virtual container will only be reported when a *new* virtual container has been created by Moab for the job.

Examples of job submission

This section includes some sample job submission requests.

Example 4-5: Submit job to run on node2 and node3

```
POST http://localhost:8080/mws/rest/jobs?api-version=3
------------------------------------

{
  "commandFile": "/tmp/test.sh",
  "credentials":   {
    "group": "adaptive",
    "user": "adaptive"
  },
  "initialWorkingDirectory": "/tmp",
  "nodesRequested":   [
    {"name": "node2"},
    {"name": "node3"}
  ]
}
```

Example 4-6: Submit job that requires 20 processors

```
POST http://localhost:8080/mws/rest/jobs?api-version=3
------------------------------------

{
  "commandFile": "/tmp/test.sh",
  "credentials":   {
    "group": "adaptive",
    "user": "adaptive"
  },
  "initialWorkingDirectory": "/tmp",
  "requirements": [{"taskCount": 20}]
}
```

Example 4-7: Submit job to run after a certain time

```
POST http://localhost:8080/mws/rest/jobs?api-version=3
------------------------------------

{
  "commandFile": "/tmp/test.sh",
  "credentials":   {
    "group": "adaptive",
    "user": "adaptive"
  },
  "dates": {"earliestRequestedStartDate": "2012-10-11 18:36:35 UTC"},
  "initialWorkingDirectory": "/tmp",
  "requirements": [{"taskCount": 20}]
}
```

Example 4-8: Submit job based on `msub` example

Given this msub command:

```
msub -l nodes=3:ppn=2,walltime=1:00:00,pmem=100 script2.pbs.cmd
```

Here is an equivalent MWS request:

```
POST http://localhost:8080/mws/rest/jobs?api-version=3
------------------------------------

{
  "duration": 3600,
  "commandFile": "/home/adaptive/script2.pbs.cmd",
  "credentials":   {
    "group": "adaptive",
    "user": "adaptive"
  },
  "initialWorkingDirectory": "/home/adaptive",
  "requirements": [  {
    "resourcesPerTask": {"memory": {"dedicated": 100}},
    "taskCount": 6,
    "tasksPerNode": 2
  }]
}
```

> To emulate what msub does, make `commandFile` an absolute path, and add `credentials.user`, `credentials.group`, and `initialWorkingDirectory`.
>
> As shown above, `nodes=3:ppn=2` is equivalent to setting `taskCount` to 6 and `tasksPerNode` to 2.

Example 4-9: Submit a job array

For information on how to submit a job array, see **Submitting job arrays** on page 1523.

Modifying jobs

The HTTP PUT method is used to modify **Jobs**.

Quick reference

```
PUT http://localhost:8080/mws/rest/jobs/<name>[/<modifyAction>]?api-version=3[&proxy-
user=<username>]
```

Restrictions

The **proxy-user** parameter is ignored unless you set `ENABLEPROXY=TRUE` in the `moab.cfg` file. For example:

```
ADMINCFG[1]            USERS=root,ted ENABLEPROXY=TRUE
```

Modify job attributes

URLs and parameters

```
PUT http://localhost:8080/mws/rest/jobs/<name>?api-version=3[&proxy-user=<username>]
[&change-mode=set]
```

Parameter	Required	Type	Valid values	Description
name	Yes	String	--	The name of the object.
proxy-user	No	String	--	Perform the action as this user.

See **Global URL parameters** on page 1403 for available URL parameters.

Additional URL parameters

Parameter	Required	Valid values	Description
change-mode	No	set (default) add remove	If **set**, replace all fields with the fields specified. If **add**, add the specified fields to existing fields. If **remove**, remove the specified fields from existing fields.

Request body

The request body below shows all the fields that are available when modifying a job, along with some sample values.

```
JSON request body
------------------------------------

{
  "credentials":   {
    "account": "account",
    "jobClass": "BATCH",
    "qosRequested": "QOS1"
  },
  "customName": "custom name for job",
  "dates": {"earliestRequestedStartDate": "2012-11-08 13:18:47 UTC"},
  "duration": 600,
  "flags":   [
    "RESTARTABLE",
    "SUSPENDABLE"
  ],
  "holds": ["User"],
  "messages":   [
    {"message": "Message one"},
    {"message": "Message two"}
  ],
  "nodesRequested": [
       {
         "name": "n015"},
       {
      "name": "n016"
       },
       {
      "name": "n017"
       },
       {
      "name": "n018"
       }
  ],
  "partitionAccessListRequested":   [
    "p1",
    "p2"
  ],
  "priorities": {
      "system": 3,
      "user": 5
  },
  "requirements": [
    {
      "features": [
        "vlan1",
        "vlan2"
      ],
      "resourcesPerTask": [
        "matlab": {
          "dedicated": 1
        },
        "tape": {
          "dedicated": 2
        }
      ]
    }
  ],
  "reservationRequested": {"name": "rsv.1"},
  "variables":   {
    "var1": "val1",
    "var2": "val2"
```

```
  }
}
```

Sample response

These messages may not match the messages returned from Moab exactly, but are given as an example of the structure of the response.

Not all messages are shown for the above request body.


```
JSON response
------------------------------------

{"messages": [
  "Account modified successfully",
  "Messages modified successfully",
  "Variables modified successfully"
]}
```


Restrictions

- Old messages are not removed from jobs; only new messages are added.
- Job `variables` have the restrictions documented in **Submitting jobs** on page 1533.
- Although the client can modify `features` and `resourcesPerTask`, Moab only considers these elements when they appear in the first element of the `requirements` array. If the requirements array contains two or more elements, all elements but the first are silently ignored.

Generic resources

Jobs can require configurable, site-specific consumable resources called generic resources. For example, some jobs may require a matlab license. Only one job at a time may legally consume this license. Matlab is not a standard resource and may only be available on some sites. Nevertheless Moab allows this to be configured and tracked as is explained in **Managing Consumable Generic Resources** on page 573.

You must specify generic resources in the `requirements.resourcesPerTask` portion of the JSON document. Any resource in `requirement.resourcesPerTask` that is not a standard resource is considered a generic resource. Standard resources include disk, memory, processors, and swap. Assume a job has the following in `requirement.resourcesPerTask`:

```
{
  "resourcesPerTask":{
    "processors":{
      "dedicated":4,
      "utilized":0
    },
    "memory":{
      "dedicated":2048,
      "utilized":0
    },
    "disk":{
      "dedicated":4096,
      "utilized":0
    },
    "swap":{
      "dedicated":1024,
      "utilized":0
    },
    "tape":{
      "dedicated":1,
      "utilized":0
    },
    "matlab":{
      "dedicated":2,
      "utilized":0
    }
  }
}
```


The standard resources the job requires are:

- 4 processors
- 2048 MB of memory
- 4096 MB of disk
- 1024 MB of swap

The generic resources the job requires are

- 1 tape
- 2 matlab

To modify a job so that it requires 1 matlab license, run the following:

```
PUT http://localhost:8080/mws/rest/jobs/Moab.2?api-version=3
{
  "requirements":[
    {
      "resourcesPerTask":{
        "matlab":{
          "dedicated":1
        }
      }
    }
  ]
}
```

Perform actions on job

URLs and parameters

```
PUT http://localhost:8080/mws/rest/jobs/<name>/<modifyAction>?api-version=3[&proxy-
user=<username>]
```

Parameter	Required	Type	Valid values	Description
name	Yes	String	--	The name of the object.
modifyAction	Yes	String	cancel checkpoint execute hold requeue rerun resume suspend unhold	If **cancel**, attempts to cancel the job (equivalent to deleting a job). If **checkpoint**, attempts to checkpoint the job. Note that the OS must support checkpointing for this to work. If **execute**, executes the job (if possible). If **hold**, attempts to hold the job using the holds set in the request body. If **requeue**, attempts to requeue the job. If **rerun**, attempts to rerun the job. If **resume**, attempts to resume the job. If **suspend**, attempts to suspend the job. If **unhold**, attempts to release the holds set in the request body.
proxy-user	No	String	--	Perform the action as this user.

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

Request bodies are only required for holding or unholding jobs. All other actions do not require request bodies of any kind.

```
JSON request body to add holds to a job
------------------------------------

{"holds": ["User"]}
```

```
JSON request body to remove holds from a job
------------------------------------

{"holds": ["User"]}
```

If no holds are specified when unholding a job, all holds will be removed. This is equivalent to specifying `holds` as a list with a single element of `All`.

Sample response

This message may not match the message returned from Moab exactly, but is given as an example of the structure of the response.

```
JSON response
------------------------------------
{"messages": ["Job modified successfully"]}
```

Deleting (canceling) jobs

The HTTP DELETE method is used to cancel **Jobs**.

Quick reference

```
DELETE http://localhost:8080/mws/rest/jobs/<name>?api-version=3[&proxy-
user=<username>]
```

Restrictions

The **proxy-user** parameter is ignored unless you set `ENABLEPROXY=TRUE` in the `moab.cfg` file. For example:

```
ADMINCFG[1]           USERS=root,ted ENABLEPROXY=TRUE
```

Cancel job

URLs and parameters

```
DELETE http://localhost:8080/mws/rest/jobs/<name>?api-version=3[&proxy-
user=<username>]
```

Parameter	Required	Type	Valid values	Description
name	Yes	String	--	The name of the object.
proxy-user	No	String	--	Perform the action as this user.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response for successful DELETE
------------------------------------
{}
```

Additional information about the DELETE can be found in the HTTP response header `X-MWS-Message`.

Related topics

- **Fields: Jobs** on page 1914
- **Resources introduction** on page 1424
- **Job arrays** on page 1523
- **Job templates** on page 1547

Job templates

This section describes behavior of the **Job Template** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The **Fields: Job Templates** reference section contains the type and description of all fields in the **Job Template** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/job-templates	Get all job templates	--	--	--
/rest/job-templates/<id>	Get single job template	--	--	--

This topic contains these sections:

- **Getting job templates** on page 1547
  - **Get all job templates** on page 1548
  - **Get single job template** on page 1548

Getting job templates

The HTTP GET method is used to retrieve **Job Template** information. Queries for all objects and a single object are available.

Quick reference

```
GET http://localhost:8080/mws/rest/job-templates/<id>?api-version=3
```

Get all job templates

URLs and parameters

```
GET http://localhost:8080/mws/rest/job-templates?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/job-templates?api-version=3&fields=id
------------------------------------

{
  "totalCount": 14,
  "resultCount": 14,
  "results":   [
    {"id": "DEFAULT"},
    {"id": "genericVM"},
    {"id": "genericVM-setup"},
    {"id": "genericVM-destroy"},
    {"id": "genericVM-migrate"},
    {"id": "genericPM"},
    {"id": "genericPM-setup"},
    {"id": "genericPM-destroy"},
    {"id": "OSStorage"},
    {"id": "OSStorage-setup"},
    {"id": "OSStorage-destroy"},
    {"id": "extraStorage"},
    {"id": "extraStorage-setup"},
    {"id": "extraStorage-destroy"}
  ]
}
```

Get single job template

URLs and parameters

```
GET http://localhost:8080/mws/rest/job-templates/<id>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response


```
JSON response
-----------------------------------

{
  "account": "account",
  "args": "arg1 arg2",
  "commandFile": "/tmp/script",
  "description": "description",
  "genericSystemJob": true,
  "id": "genericVM",
  "inheritResources": false,
  "jobDependencies": [  {
    "name": "genericVM-setup",
    "type": "JOBSUCCESSFULCOMPLETE"
  }],
  "jobFlags": ["VMTRACKING"],
  "jobTemplateFlags": ["SELECT"],
  "jobTemplateRequirements": [  {
    "architecture": "x86_64",
    "diskRequirement": 500,
    "genericResources": {"tape": 3},
    "nodeAccessPolicy": "SINGLEJOB",
    "operatingSystem": "Ubuntu 10.04.3",
    "requiredDiskPerTask": 200,
    "requiredFeatures": ["dvd"],
    "requiredMemoryPerTask": 1024,
    "requiredProcessorsPerTask": 2,
    "requiredSwapPerTask": 512,
    "taskCount": 4
  }],
  "priority": 20,
  "qos": "qos",
  "queue": "queue",
  "durationRequested": 600,
  "select": true,
  "trigger": null,
  "version": 0,
  "vmUsagePolicy": "REQUIREPM"
}
```


Related topics

- **Fields: Job Templates** on page 1974
- **Resources introduction** on page 1424
- **Jobs** on page 1525
- **Job arrays** on page 1523

Metric types

This section describes behavior of the **Metric Type** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The **Fields: Metric Types** reference section contains the type and description of all fields in the **Metric Type** object.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/metric-types	Get all metric types	--	--	--

This topic contains these sections:

Getting metric types

The HTTP GET method is used to retrieve **Metric Type** information.

Quick reference

```
GET http://localhost:8080/mws/rest/metric-types?api-version=3
```

Get all metric types

URLs and parameters

```
GET http://localhost:8080/mws/rest/metric-types?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/metric-types?api-version=3&fields=id
------------------------------------

{
  "totalCount": 9,
  "resultCount": 9,
  "results":    [
    {"id": "vmcount"},
    {"id": "watts"},
    {"id": "pwatts"},
    {"id": "temp"},
    {"id": "cpu"},
    {"id": "mem"},
    {"id": "io"},
    {"id": "ccores"},
    {"id": "threads"}
  ]
}
```

Related topics

- **Fields: Metric Types** on page 2003
- **Resources introduction** on page 1424

Nodes

This section describes behavior of the **Node** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The Node API is new with *API version 2*. The supported methods table below requires each resource to be accessed with a URL parameter of `api-version=3` in order to behave as documented.

For more information, see **Requesting specific API versions** on page 1406.

The **Fields: Nodes** reference contains the type and description of all fields in the **Node** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/nodes	Get all nodes	--	--	--
/rest/nodes/<name>	Get single node	Modify node	--	--

This topic contains these sections:

Getting nodes

The HTTP GET method is used to retrieve **Node** information.

Quick reference

```
GET http://localhost:8080/mws/rest/nodes/<name>?api-version=3
```

Get all nodes

URLs and parameters

```
GET http://localhost:8080/mws/rest/nodes?api-version=3
```

Parameter	Required	Type	Description	Example
query	No	JSON	Queries for specific results. It is possible to query by one or more fields based on MongoDB query syntax.	`query={"type":"-compute"}`
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	`sort={"name":-1}`

See **Global URL parameters** on page 1403 for available URL parameters.

This query will not return the `DEFAULT` or `GLOBAL` nodes from Moab. However, the **Get single node** task may be used to retrieve them individually if desired.

Sample response

```
GET http://localhost:8080/mws/rest/nodes?api-version=3&fields=name
------------------------------------

{
  "totalCount": 3,
  "resultCount": 3,
  "results":    [
    {"name": "node1"},
    {"name": "node2"},
    {"name": "node3"}
  ]
}
```

Get single node

URLs and parameters

```
GET http://localhost:8080/mws/rest/nodes/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
name	Yes	String	--	The name of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

The `attributes` field is only applicable in API version 2 and later, and the `MOAB_TENANT` field only applies if the node is attached to a tenant.

Sample response

```
JSON response
------------------------------------

{
  "name": "l26.csa",
  "architecture": null,
  "classes": ["class1"],
  "attributes":   {
    "MOAB_TENANT":   {
      "value": "1234567890abcdef12345678",
      "displayValue": "ResearchGroup"
    },
    "MOAB_DATACENTER":   {
      "value": "vcenter-datacenter-401",
      "displayValue": "vcenter-vcenter - adaptive data center"
    },
    "vcenter-vcenter-adaptive data center-compute nodes":     {
      "value": null,
      "displayValue": null
    }
  },
  "featuresCustom": ["feature1", "feature2"],
```

```
  "featuresReported": ["vcenter-vcenter-adaptive data center-compute nodes"],
  "index": 26,
  "ipAddress": "10.0.8.76",
  "isHypervisor": true,
  "lastUpdatedDate": "2013-05-24 20:18:11 UTC",
  "migrationDisabled": false,
  "partition": "mws",
  "processorSpeed": null,
  "profilingEnabled": false,
  "rack": null,
  "resourceManagerMessages":   {
    "torque": null,
    "mws": null
  },
  "slot": null,
  "type": "compute",
  "messages": [  {
    "count": 11,
    "createdDate": "2012-10-24 04:06:04 UTC",
    "expireDate": "2037-10-24 12:26:40 UTC",
    "message": "This is a message"
  }],
  "metrics":   {
    "vmcount": 0,
    "cpuUtilization": 0.275,
    "cpuLoad": 0.01115
  },
  "variables":   {
    "VCENTER_DATASTORE_LOCAL1": "datastore-415",
    "VCENTER_DATASTORE_REMOTE1": "datastore-448"
  },
  "states":   {
    "powerState": "On",
    "powerStateExpected": null,
    "state": "Idle",
    "stateExpected": "Idle",
    "stateLastUpdatedDate": "2013-05-24 09:33:45 UTC",
    "subState": null,
    "subStateLast": null,
    "subStateLastUpdatedDate": null
  },
  "operatingSystem":   {
    "hypervisorType": "esx",
    "image": "vcenter-vcenter-esx-5.0",
    "imageExpected": null,
    "imageLastUpdatedDate": null,
    "imagesAvailable": [],
    "virtualMachineImages":     [
      "win2008",
      "centos6"
    ]
  },
  "resources":   {
    "processors":     {
      "configured": 4,
      "real": 4,
      "dedicated": 0,
      "available": 4,
      "utilized": -1
    },
    "memory":     {
      "configured": 10239,
```

```
        "real": 10239,
        "dedicated": 0,
        "available": 9227,
        "utilized": 0
      },
      "disk":     {
        "configured": 0,
        "real": 0,
        "dedicated": 0,
        "available": 0,
        "utilized": 0
      },
      "swap":     {
        "configured": 0,
        "real": 0,
        "dedicated": 0,
        "available": 0,
        "utilized": 0
      }
    },
    "resourceManagers": [  {
      "name": "mws",
      "isMaster": true,
      "stateReported": "Active"
    }],
    "jobs": [],
    "reservations":   [
          {
        "name": "system.5",
        "type": "user"
      },
          {
        "name": "system.17",
        "type": "user"
      }
    ],
    "virtualContainers": [],
    "virtualMachines": [],
    "triggers": []
  }
```


Modifying nodes

The HTTP PUT method is used to modify **Nodes**.

Quick reference

```
PUT http://localhost:8080/mws/rest/nodes/<name>?api-version=3[&proxy-user=<username>]
```

Restrictions

The **proxy-user** parameter is ignored unless you set `ENABLEPROXY=TRUE` in the `moab.cfg` file. For example:

```
ADMINCFG[1]           USERS=root,ted ENABLEPROXY=TRUE
```

Modify node

URLs and parameters

```
PUT http://localhost:8080/mws/rest/nodes/<name>?api-version=3[&proxy-user=<username>]
[&change-mode=set]
```

Parameter	Required	Type	Valid values	Description
name	Yes	String	--	The name of the object.
proxy-user	No	String	--	Perform the action as this user.

See **Global URL parameters** on page 1403 for available URL parameters.

Additional URL parameters

Parameter	Required	Valid values	Description
change-mode	No	set (default) add remove	If **set**, replace all features with the features specified. If **add**, add the specified features to existing features. If **remove**, remove the specified features from existing features.

Request body

The request body below shows all the fields that are available when modifying a node, along with some sample values.

```
Sample JSON request body to modify a node
------------------------------------

{
  "featuresCustom": ["feature1", "feature2"],
  "messages":   [
    {"message": "Message one"},
    {"message": "Message two"}
  ],
  "metrics": {"pwatts": 211},
  "operatingSystem": {"image": "esx4.1"},
  "partition": "part1",
  "states":   {
    "powerState": "On",
    "state": "Running"
  },
  "variables":   {
    "key": "value",
    "arbitrary text key": "more value"
  }
}
```

Sample response

This message may not match the message returned from Moab exactly, but is given as an example of the structure of the response.


```
JSON response
------------------------------------

{"messages":[
  "Successfully modified os to 'linux'",
  "Successfully powered node off"
]}
```


Related topics

- **Fields: Nodes** on page 2004
- **Resources introduction** on page 1424

Notification conditions

This section describes behavior of the **Notification Conditions** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The Notification Conditions API is new with *API version 3*, and is not available with older API versions. The supported methods table below requires each resource to be accessed with a URL parameter of `api-version=3`.

For more information, see **Requesting specific API versions** on page 1406.

The **Fields: Notification Conditions** reference contains the type and description of all fields in the **Notification Conditions** object.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/notification-conditions	Get all notification conditions	Update notification condition	--	--
/rest/notification-conditions/<id>	Get single notification condition	--	--	--

This topic contains these sections:

Getting notification conditions

The HTTP GET method is used to retrieve **Notification Condition** information.

Quick reference

```
GET http://localhost:8080/mws/rest/notification-conditions?api-version=3
GET http://localhost:8080/mws/rest/notification-conditions/<id>?api-version=3
```

Get all notification conditions

URLs and parameters

```
GET http://localhost:8080/mws/rest/notification-conditions?api-version=3[&query=
{"escalationLevel":"ADMIN"}][&sort={"observedDate":-1}]
```

Parameter	Required	Type	Description	Example
query	No	JSON	Query for specific results. It is possible to query notifications by one or more fields based on MongoDB query syntax.	`query={"escal-ationLevel":"ADMIN"}`
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	`sort={"observedDate":-1}`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/notification-conditions?api-version=3&query=
{"escalationLevel":"ADMIN"}&sort={"observedDate":-1}
------------------------------------

 {
    "totalCount": 2,
    "resultCount": 2,
    "results": [
        {
            "createdDate": "2013-09-10 23:13:33 UTC",
            "details": {
                "pluginType": "NodeUtilizationReport",
                "pluginId": "node-report"
            },
            "escalationLevel": "ADMIN",
            "expirationDate": null,
            "expirationDuration": null,
            "message": "The node 'testnode' has not been updated since the last poll,
which is likely due to a misconfiguration.",
            "objectId": "testnode",
            "objectType": "Node",
            "observedDate": "2013-09-10 23:13:33 UTC",
            "origin": "MWS/plugins/NodeUtilizationReport/node-report",
            "tenant": {
                 "id":"1234567890abcdef12345678",
                 "name":"Research"
            },
            "id": "522fa79de4b0cafeaec6f83e"
        },
        {
            "createdDate": "2013-09-11 17:19:35 UTC",
            "details": {
                "pluginType": "VCenter",
                "pluginId": "vcenter42"
            },
            "escalationLevel": "ADMIN",
            "expirationDate": null,
            "expirationDuration": null,
            "message": "The node 'node1' does not have vcenter tools installed,
therefore the state is unknown and migrations may not work correctly",
            "objectId": null,
            "objectType": "System",
            "observedDate": "2013-09-11 17:19:35 UTC",
            "origin": "MWS/plugins/VCenter/vcenter42",
            "tenant": {
                 "id":"1234567890abcdef12345678",
                 "name":"Research"
            },
            "id": "5230a627e4b0d51bef490e86"
        }
    ]
}
```

A notification's `tenant` is automatically inherited from the `objectId` and `objectType` fields. If no object is associated with the notification condition, the notification is visible to all tenants.

Get single notification condition

URLs and parameters

```
GET http://localhost:8080/mws/rest/notification-conditions/<id>?api-version=3
```

Parameter	Required	Type	Description
id	Yes	String	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/notification-
conditions/521a1f18e4b0e3f9031f47f5?api-version=3
------------------------------------

{
    "createdDate": "2013-09-10 23:13:33 UTC",
    "details": {
        "pluginType": "NodeUtilizationReport",
        "pluginId": "node-report"
    },
    "escalationLevel": "ADMIN",
    "expirationDate": null,
    "expirationDuration": null,
    "message": "The node 'testnode' has not been updated since the last poll, which is
likely due to a misconfiguration.",
    "objectId": "testnode",
    "objectType": "Node",
    "observedDate": "2013-09-10 23:13:33 UTC",
    "origin": "MWS/plugins/NodeUtilizationReport/node-report",
    "tenant": {
        "id":"1234567890abcdef12345678",
        "name":"Research"
    },
    "id": "522fa79de4b0cafeaec6f83e"
}
```

A notification's `tenant` is automatically inherited from the `objectId` and `objectType` fields. If no object is associated with the notification condition, the notification is visible to all tenants.

Updating notification conditions

The HTTP PUT method is used to update **Notification Condition** information. The PUT operation is idempotent, meaning that is used for both creating new notification conditions and updating existing ones. If the `escalationLevel`, `origin`, `message`, `objectType`, and `objectId` fields match an existing notification condition, it will be updated. Otherwise, a new condition will be created.

Quick reference

```
PUT http://localhost:8080/mws/rest/notification-conditions?api-version=3
```

Update notification condition

URLs and parameters

```
PUT http://localhost:8080/mws/rest/notification-conditions?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

The request body below shows some fields that are available when updating a notification condition, along with some sample values.

```
Sample JSON request body to update a notification condition
------------------------------------

{
    "details": {
        "pluginType": "NodeTester",
        "pluginId": "my-tester1"
    },
    "escalationLevel": "ADMIN",
    "expirationDuration": 30,
    "message": "Node 'node2' is powered off, please check your hardware.",
    "objectId": "node2",
    "objectType": "Node",
    "origin": "NodeTester/my-tester1/Test.groovy:141"
}
```

Sample response

```
JSON response
------------------------------------

{
    "createdDate": "2013-09-10 23:13:33 UTC",
    "details": {
        "pluginType": "NodeTester",
        "pluginId": "my-tester1"
    },
    "escalationLevel": "ADMIN",
    "expirationDate": "2013-09-10 23:14:03 UTC",
    "expirationDuration": 30,
    "observedDate": "2013-09-10 23:13:33 UTC",
    "message": "Node 'node2' is powered off, please check your hardware.",
    "objectId": "node2",
    "objectType": "Node",
    "origin": "NodeTester/my-tester1/Test.groovy:141",
    "tenant": {
        "id":"1234567890abcdef12345678",
        "name":"Research"
    },
    "id": "5230a627e4b0d51bef490e86"
}
```

Related topics

- **Resources introduction** on page 1424
- **Events** on page 1506

- **Fields: Events** on page 1837
- **Notifications** on page 1563
- **Fields: Notifications** on page 2028
- **Fields: Notification Conditions** on page 2024
- **Creating events and notifications** on page 1682
- **Plugin event service** on page 1733
- **Handling events** on page 1689
- **System events** on page 1422
- **Securing the connection with the message queue** on page 1395

Notifications

This section describes behavior of the **Notifications** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The Notifications API is new with *API version 3*, and is not available with older API versions. The supported methods table below requires each resource to be accessed with a URL parameter of `api-version=3`.

For more information, see **Requesting specific API versions** on page 1406.

The **Fields: Notifications** reference contains the type and description of all fields in the **Notifications** object.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/notifications/	Get all notifications	--	--	--
/rest/notifications/<id>	Get single notification	--	--	--
/rest/notifications/ignore	--	Ignore all notifications	--	--
/rest/notifications/<id>/ignore	--	Ignore single notification	--	--
/rest/notifications/unignore	--	Unignore all notifications	--	--

Resource	GET	PUT	POST	DELETE
/rest/notifications/<id>/unignore	--	Unignore single notification	--	--
/rest/notifications/dismiss	--	Dismiss all notifications	--	--
/rest/notifications/<id>/dismiss	--	Dismiss single notification	--	--

This topic contains these sections:

Getting notifications

The HTTP GET method is used to retrieve **Notification** information.

Quick reference

```
GET http://localhost:8080/mws/rest/notifications?api-version=3
GET http://localhost:8080/mws/rest/notifications/<id>?api-version=3
```

Get all notifications

URLs and parameters

```
GET http://localhost:8080/mws/rest/notifications?api-version=3[&proxy-user=<username>]
[&query={"ignoredDate":null,"dismissedDate":null}][&sort={"observedDate":-1}]
```

Para-meter	Requ-ired	Ty-pe	Description	Example
proxy-user	No	Str-ing	Perform the action as this user. Notifications cannot be created directly. Instead, they are automatically created for the current user or `proxy-user` specified in the request from non-expired notification conditions (see **Notification conditions** on page 1558). This is true no matter the `query` specified.	--
query	No	JS-ON	Query for specific results. It is possible to query notifications by one or more fields based on MongoDB query syntax. However, typically you will want to query on `{"ignoredDate":null,"dismissedDate":null}`.	`query={"ignoredDate":null,"dis-missedDate":null}`
sort	No	JS-ON	Sort the results. Use `1` for ascending and `-1` for descending.	`sort={"observedDate":-1}`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/notifications?api-version=3&proxy-
user=<username>&query={"ignoredDate":null,"dismissedDate":null}][&sort=
{"observedDate":-1}
------------------------------------

{
    "totalCount": 2,
    "resultCount": 2,
    "results": [
        {
            "conditionId": "521bdea1e4b019cd33e29c86",
            "createdDate": "2013-08-26 23:02:56 UTC",
            "details": {},
            "dismissedDate": null,
            "ignoredDate": null,
            "message": "A health check failed for the 'ZeroMQ Message Queue'
connection, please see the MWS health details page for more information.",
            "objectId": "zmq",
            "objectType": "Health",
            "observedDate": "2013-09-05 17:57:00 UTC",
            "origin": "MWS/HealthNotificationJob",
            "user": "admin",
            "id": "5230ed82e4b065347016d62f"
        },
        {
            "conditionId": "521a1f18e4b0e3f9031f47f5",
            "createdDate": "2013-08-25 15:13:28 UTC",
            "details": {},
            "dismissedDate": null,
            "ignoredDate": null,
            "message": "A health check failed for the 'LDAP' connection, please see
the MWS health details page for more information.",
            "objectId": "ldap",
            "objectType": "Health",
            "observedDate": "2013-08-30 18:11:15 UTC",
            "origin": "MWS/HealthNotificationJob",
            "user": "admin",
            "id": "5230ed82e4b065347016d60d"
        }
    ]
}
```

Get single notification

URLs and parameters

```
GET http://localhost:8080/mws/rest/notifications/<id>?api-version=3
```

Parameter	Required	Type	Description
id	Yes	String	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/notifications/5230ed82e4b065347016d60d?api-
version=3
------------------------------------

{
    "conditionId": "521a1f18e4b0e3f9031f47f5",
    "createdDate": "2013-08-25 15:13:28 UTC",
    "details": {},
    "dismissedDate": null,
    "ignoredDate": null,
    "message": "A health check failed for the 'LDAP' connection, please see the MWS
health details page for more information.",
    "objectId": "ldap",
    "objectType": "Health",
    "observedDate": "2013-08-30 18:11:15 UTC",
    "origin": "MWS/HealthNotificationJob",
    "user": "admin",
    "id": "5230ed82e4b065347016d60d"
}
```

Ignoring notifications

The HTTP PUT method is used to ignore **Notifications**.

Quick reference

```
PUT http://localhost:8080/mws/rest/notifications/ignore?api-version=3
PUT http://localhost:8080/mws/rest/notifications/<id>/ignore?api-version=3
```

Ignore all notifications

URLs and parameters

```
PUT http://localhost:8080/mws/rest/notifications/ignore?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response
------------------------------------

{"messages":["Updated 10 Notification objects"]}
```

Ignore single notification

URLs and parameters

```
PUT http://localhost:8080/mws/rest/notifications/5230ed82e4b065347016d60d/ignore?api-
version=3
```

Parameter	Required	Type	Description
id	Yes	String	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
PUT http://localhost:8080/mws/rest/notifications/5230ed82e4b065347016d60d/ignore?api-
version=3
------------------------------------

{
    "conditionId": "521a1f18e4b0e3f9031f47f5",
    "createdDate": "2013-08-25 15:13:28 UTC",
    "details": {},
    "dismissedDate": null,
    "ignoredDate": "2013-09-17 15:34:36 UTC",
    "message": "A health check failed for the 'LDAP' connection, please see the MWS
health details page for more information.",
    "objectId": "ldap",
    "objectType": "Health",
    "observedDate": "2013-08-30 18:11:15 UTC",
    "origin": "MWS/HealthNotificationJob",
    "user": "admin",
    "id": "5230ed82e4b065347016d60d"
}
```

Unignoring notifications

The HTTP PUT method is used to unignore **Notifications**.

Quick reference

```
PUT http://localhost:8080/mws/rest/notifications/unignore?api-version=3
PUT http://localhost:8080/mws/rest/notifications/<id>/unignore?api-version=3
```

Unignore all notifications

URLs and parameters

```
PUT http://localhost:8080/mws/rest/notifications/unignore?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response
------------------------------------

{"messages":["Updated 10 Notification objects"]}
```

Unignore single notification

```
PUT
http://localhost:8080/mws/rest/notifications/5230ed82e4b065347016d60d/unignore?api-
version=3
```

Parameter	Required	Type	Description
id	Yes	String	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
PUT
http://localhost:8080/mws/rest/notifications/5230ed82e4b065347016d60d/unignore?api-
version=3
------------------------------------

{
    "conditionId": "521a1f18e4b0e3f9031f47f5",
    "createdDate": "2013-08-25 15:13:28 UTC",
    "details": {},
    "dismissedDate": "null",
    "ignoredDate": null,
    "message": "A health check failed for the 'LDAP' connection, please see the MWS
health details page for more information.",
    "objectId": "ldap",
    "objectType": "Health",
    "observedDate": "2013-08-30 18:11:15 UTC",
    "origin": "MWS/HealthNotificationJob",
    "user": "admin",
    "id": "5230ed82e4b065347016d60d"
}
```

Dismissing notifications

The HTTP PUT method is used to dismiss **Notifications**.

Quick reference

```
PUT http://localhost:8080/mws/rest/notifications/dismiss?api-version=3
PUT http://localhost:8080/mws/rest/notifications/<id>/dismiss?api-version=3
```

Dismiss all notifications

URLs and parameters

```
PUT http://localhost:8080/mws/rest/notifications/dismiss?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response
------------------------------------

{"messages":["Updated 10 Notification objects"]}
```

Dismiss single notification

URLs and parameters

```
PUT http://localhost:8080/mws/rest/notifications/5230ed82e4b065347016d60d/dismiss?api-
version=3
```

Parameter	Required	Type	Description
id	Yes	String	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
PUT http://localhost:8080/mws/rest/notifications/5230ed82e4b065347016d60d/dismiss?api-
version=3
------------------------------------

{
    "conditionId": "521a1f18e4b0e3f9031f47f5",
    "createdDate": "2013-08-25 15:13:28 UTC",
    "details": {},
    "dismissedDate": "2013-09-17 15:34:36 UTC",
    "ignoredDate": null,
    "message": "A health check failed for the 'LDAP' connection, please see the MWS
health details page for more information.",
    "objectId": "ldap",
    "objectType": "Health",
    "observedDate": "2013-08-30 18:11:15 UTC",
    "origin": "MWS/HealthNotificationJob",
    "user": "admin",
    "id": "5230ed82e4b065347016d60d"
}
```

Related topics

- **Resources introduction** on page 1424
- **Events** on page 1506
- **Fields: Events** on page 1837
- **Notifications** on page 1563
- **Fields: Notifications** on page 2028
- **Creating events and notifications** on page 1682
- **Plugin event service** on page 1733
- **Handling events** on page 1689

Permissions

This section describes behavior of the **Permissions** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The **Fields: User's Permissions** reference section contains the type and description of fields that all **Permissions** have in common.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/permissions	Get all permissions	--	Create single permission	--
/rest/permissions/<id>	Get single permission	--	--	Delete single permission
/rest/permissions/users/<id>	Get a user's permissions	--	--	--
/rest/permissions/users	Get a current user's permissions	--	--	--

This topic contains these sections:

Getting permissions

The HTTP GET method is used to retrieve **Permission** information. You can query all objects or a single object.

Quick reference

```
GET http://localhost:8080/mws/rest/permissions?api-version=3
GET http://localhost:8080/mws/rest/permissions/<id>?api-version=3
```

Get all permissions

URLs and parameters

```
GET http://localhost:8080/mws/rest/permissions?api-version=3[&query={"field":"value"}
&sort={"field":<1|-1>}]
```

Parameter	Required	Type	Description	Example
query	No	JSON	Queries for specific results. It is possible to query permissions by one or more fields based on MongoDB query syntax.	`query={"type":"CUSTOM"}`
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	`sort={"name":-1}`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/permissions?api-
version=3&fields=resource,action,description
------------------------------------

{
       "totalCount": 1,
       "resultCount": 1,
       "results": [{
               "resource" : "chart",
               "action" : "read",
               "description" : "The permission to view all charts."
               } ]
}
```

Sorting and querying

See the sorting and querying sections of **Global URL parameters** on page 1403.

Get single permission

URLs and parameters

```
GET http://localhost:8080/mws/rest/permissions/<id>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the permission.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/permissions/<id>?api-version=3
------------------------------------

{
       "action" : "create",
       "description" : "The permission to create all charts.",
       "id" : "50296335e4b0011b0f8394ec",
       "label" : "Create Chart",
       "resource" : "chart",
       "resourceFilter" : null,
       "type" : "custom",
       "scope" : "NONE",
       "version" : 0
}
```

For permissions with type `"domain"`, scope must be `GLOBAL` or `TENANT`. All other permissions should have scope `NONE`.

Get a user's permissions

URLs and parameters

```
GET http://localhost:8080/mws/rest/permissions/users/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
name	Yes	String	--	The name of the user.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/permissions/users/bob?api-version=3
------------------------------------

[
          {
         "action": "read",
         "description": "The permission to read all charts",
         "id": "5033b842e4b09cc61bedb818",
         "label": "",
         "resource": "chart",
         "resourceFilter": null,
         "type": "custom",
         "scope": "NONE",
         "version": 1
       },
          {
         "action": "read",
         "description": "The permission to read all pages",
         "id": "5033b8a5e4b09cc61bedb82d",
         "label": "",
         "resource": "page",
         "resourceFilter": null,
         "type": "custom",
         "scope": "NONE",
         "version": 1
       },
          {
         "action": "update",
         "description": "The permission to update all pages",
         "id": "5033b8a5e4b09cc61bedb82f",
         "label": "",
         "resource": "page",
         "resourceFilter": null,
         "type": "custom",
         "scope": "NONE",
         "version": 1
       }
]
```

Get a current user's permissions

URLs and parameters

```
GET http://localhost/mws/rest/permissions/users/?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost/mws/rest/permissions/users/?api-version=3
------------------------------------

[
          {
          "action": "read",
          "description": "The permission to read all charts",
          "id": "5033b842e4b09cc61bedb818",
          "label": "",
          "resource": "chart",
          "resourceFilter": null,
          "type": "custom",
          "scope": "NONE",
          "version": 1
        },
          {
          "action": "read",
          "description": "The permission to read all pages",
          "id": "5033b8a5e4b09cc61bedb82d",
          "label": "",
          "resource": "page",
          "resourceFilter": null,
          "type": "custom",
          "scope": "NONE",
          "version": 1
        },
          {
          "action": "update",
          "description": "The permission to update all pages",
          "id": "5033b8a5e4b09cc61bedb82f",
          "label": "",
          "resource": "page",
          "resourceFilter": null,
          "type": "custom",
          "scope": "NONE",
          "version": 1
        }
]
```

Creating permissions

The HTTP POST method is used to create **Permissions**.

Quick reference

```
POST http://localhost:8080/mws/rest/permissions?api-version=3
```

Create single permission

URLs and parameters

```
POST http://localhost:8080/mws/rest/permissions?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

> The **resource**, **action**, and **type** are required on each permission.
>
> Api permissions are permissions with the type 'api' and are the only permissions enforced by MWS.
>
> Api permissions must map to a valid resource. For example, "services" is valid because there is a resource `/mws/rest/services`.
>
> Api permissions must have `create`, `read`, `update`, or `delete` as the action.

The following is an example request body to create a permission:

```
POST http://localhost:8080/mws/rest/permissions?api-version=3
------------------------------------

{
                        "resource" : "Chart",
                        "action" : "read",
                        "type" : "custom",
                        "scope" : "NONE",
                        "label" : "Read all charts",
                        "description" : "The permissions to view all charts."
}
```

Sample response

If the request was successful, the response body is the new permission that was created exactly as shown in **Get single permission**. On failure, the response is an error message.

Deleting permissions

The HTTP DELETE method is used to delete **Permissions**.

Quick reference

```
DELETE http://localhost:8080/mws/rest/permissions/<id>?api-version=3
```

Delete single permission

URLs and parameters

```
DELETE http://localhost:8080/mws/rest/permission/<id>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the permission.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response
------------------------------------

{}
```

Related topics

- **Fields: User's Permissions** on page 2186
- **Resources introduction** on page 1424

Plugins

This section describes behavior of the **Plugins** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

> The **Fields: Plugins** reference contains the type and description of all fields in the **Plugin** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/plugins	**Get all plugins**	--	**Create plugin**	--
/rest/plugins/reporting-jobs/<jobName>?api-version=3	**Get all plugins reporting object**	--	--	--
/rest/plugins/reporting-nodes/<nodeName>?api-version=3	**Get all plugins reporting object**	--	--	--
/rest/plugins/reporting-vms/<vmName>?api-version=3	**Get all plugins reporting object**	--	--	--
/rest/plugins/<id>	**Get single plugin**	**Modify plugin**	--	**Delete plugin**
/rest/plugins/<id>/poll	--	--	**Trigger plugin poll**	--

Resource	GET	PUT	POST	DELETE
/rest/plugins/<id>/services/<serviceName>	Access a plugin web service	Access a plugin web service	Access a plugin web service	Access a plugin web service

This topic contains these sections:

Getting plugins

The HTTP GET method is used to retrieve **Plugin** information. Queries for all objects, a single object, and query by reported object are available.

Quick reference

```
GET http://localhost:8080/mws/rest/plugins?api-version=3
GET http://localhost:8080/mws/rest/plugins/<id>?api-version=3
GET http://localhost:8080/mws/rest/plugins/reporting-jobs/<jobName>?api-version=3
GET http://localhost:8080/mws/rest/plugins/reporting-nodes/<nodeName>?api-version=3
GET http://localhost:8080/mws/rest/plugins/reporting-vms/<vmName>?api-version=3
```

Get all plugins

URLs and parameters

```
GET http://localhost:8080/mws/rest/plugins?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/plugins?api-version=3&fields=id
------------------------------------

{
  "totalCount": 3,
  "resultCount": 3,
  "results":   [
    {"id": "plugin1"},
    {"id": "plugin2"},
    {"id": "plugin3"}
  ]
}
```

Get all plugins reporting object

URLs and parameters

```
GET http://localhost:8080/mws/rest/plugins/reporting-jobs/<jobName>?api-version=3
GET http://localhost:8080/mws/rest/plugins/reporting-nodes/<nodeName>?api-version=3
GET http://localhost:8080/mws/rest/plugins/reporting-vms/<vmName>?api-version=3
```

Parameter	Required	Type	Valid values	Description
jobName	Yes	String	--	The name of the job to query by.
nodeName	Yes	String	--	The name of the node to query by.
vmName	Yes	String	--	The name of the VM to query by.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

This built-in query returns the same information as **Get all plugins**, but filters the items to only plugins that are currently reporting the specified job, node, or VM (see **Reporting state data** on page 1676). The list is sorted ascending by the `precedence` field. In other words, the most authoritative plugin for the report is listed first. For more information, see **Data consolidation** on page 1655.

```
GET http://localhost:8080/mws/rest/plugins/reporting-nodes/node1?api-
version=3&fields=id
------------------------------------

{
  "totalCount": 3,
  "resultCount": 3,
  "results":   [
    {"id": "plugin1"},
    {"id": "plugin2"},
    {"id": "plugin3"}
  ]
}
```

Get single plugin

URLs and parameters

```
GET http://localhost:8080/mws/rest/plugins/<id>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response
------------------------------------

{
  "id":"plugin1",
  "pluginType":"Native",
  "pollInterval":30,
  "autoStart":true,
  "config":{
    "getJobs":"exec:///opt/moab/tools/workload.query.pl"
  },
  "state":"STARTED",
  "nextPollDate":"2011-12-02 17:28:52 UTC",
  "lastPollDate":"2011-12-02 17:28:22 UTC"
}
```

Creating plugins

The HTTP POST method is used to create **Plugins**.

Quick reference

```
POST http://localhost:8080/mws/rest/plugins?api-version=3
```

Create plugin

URLs and parameters

```
POST http://localhost:8080/mws/rest/plugins?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

When creating a plugin, the `id` and `pluginType` fields are required. The request body below shows all fields that are available when creating a plugin, along with some sample values.

```
JSON request body
------------------------------------

{
  "id":"plugin1",
  "pluginType":"Native",
  "pollInterval":30,
  "autoStart":true,
  "config":{
    "getJobs":"exec:///opt/moab/tools/workload.query.pl"
  }
}
```

Sample response

```
JSON response for successful POST
------------------------------------

{"id": "plugin1"}
```

Restrictions

While it is possible to create a plugin with arbitrary nested configuration, such as:

```
…
"config":{
  "nestedObject":{
    "property1":"value1",
    "property2":"value2"
  },
  "nestedList:["listItem1", "listItem2"]
}
```

It is *not* recommended, because the user interface (see **Plugin management** on page 1720) does not support editing or viewing any configuration data values other than strings.

Modifying plugins

The HTTP PUT method is used to modify **Plugins**. Additionally, the POST method may be used to trigger an immediate poll of a **Plugin**.

Quick reference

```
PUT http://localhost:8080/mws/rest/plugins/<id>?api-version=3
POST http://localhost:8080/mws/rest/plugins/<id>/poll?api-version=3
```

Modify plugin

URLs and parameters

```
PUT http://localhost:8080/mws/rest/plugins/<id>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

The request body below shows all the fields that are available when modifying a **Plugin**, along with some sample values.

```
JSON request body for plugin modification
------------------------------------

{
  "state":"STARTED",
  "pollInterval":30,
  "autoStart":true,
  "config":{
    "getJobs":"exec:///opt/moab/tools/workload.query.pl"
  },
  "state":"STARTED"
}
```

Sample response

```
JSON response
------------------------------------

{"messages":["Plugin plugin1 updated", "Started Plugin 'plugin1'"]}
```

Trigger plugin poll

URLs and parameters

```
POST http://localhost:8080/mws/rest/plugins/<id>/poll?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Trigger poll

This resource call will trigger an immediate poll of the specified plugin. It is equivalent to the same operation on **Monitoring and lifecycle controls** on page 1723.

Request body

No request body is required.

Sample response

```
JSON response
------------------------------------

{"messages":["Polled Plugin with ID 'myPlugin'"]}
```

Deleting plugins

The HTTP DELETE method is used to delete **Plugins**.

Quick reference

```
DELETE http://localhost:8080/mws/rest/plugins/<id>?api-version=3
```

Delete plugin

URLs and parameters

```
DELETE http://localhost:8080/mws/rest/plugins/<id>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response for successful DELETE
------------------------------------

{}
```

Additional information about a successful DELETE can be found in the HTTP response header `X-MWS-Message`.

```
JSON response for an unsuccessful DELETE
------------------------------------

{"messages":["Plugin plugin1 could not be deleted", "Error message describing the
problem"]}
```

Accessing Plugin Web Services

All HTTP methods can be used to access **Plugin Web Services**. However, some services only support specific methods. Check the specific plugin type documentation for more information.

Quick reference

```
GET http://localhost:8080/mws/rest/plugins/<id>/services/<serviceName>
[/<objectId>]?api-version=3
POST http://localhost:8080/mws/rest/plugins/<id>/services/<serviceName>
[/<objectId>]?api-version=3
PUT http://localhost:8080/mws/rest/plugins/<id>/services/<serviceName>
[/<objectId>]?api-version=3
DELETE http://localhost:8080/mws/rest/plugins/<id>/services/<serviceName>
[/<objectId>]?api-version=3
```

Access a plugin web service

URLs and parameters

```
GET http://localhost:8080/mws/rest/plugins/<id>/services/<serviceName>
[/<objectId>]?api-version=3
POST http://localhost:8080/mws/rest/plugins/<id>/services/<serviceName>
[/<objectId>]?api-version=3
PUT http://localhost:8080/mws/rest/plugins/<id>/services/<serviceName>
[/<objectId>]?api-version=3
DELETE http://localhost:8080/mws/rest/plugins/<id>/services/<serviceName>
[/<objectId>]?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.
objectId	No	String	--	An arbitrary ID parameter that will be passed to the web service.
serviceName	Yes	String	--	The name of the web service, either in CamelCase or hyphenated.

See **Global URL parameters** on page 1403 for available URL parameters.

Web service IDs

Translation is done to map CamelCase web service names to hyphenated names in the URL. For example, a web service method named `notifyEvent` on a plugin with a name of `notifications` may be called with the following URLs:

```
// CamelCase
/rest/plugins/notifications/services/notifyEvent

// Hyphenated
/rest/plugins/notifications/services/notify-event
```

HTTP method and request body

Because plugin custom web services do not need to distinguish which HTTP method is used (see **Custom web services** on page 1654), it is recommended to use GET and POST when making requests to access web services unless documented otherwise. The request body and output may vary for each web service called. See **Plugin types** on page 1585 for the requested plugin for available web services, request parameters, and expected output.

Related topics

- **Fields: Plugins** on page 2030
- **Resources introduction** on page 1424
- **Plugin types** on page 1585

Plugin types

This section describes behavior of the **Plugin Type** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

> The **Fields: Plugin Types** reference section contains the type and description of all fields in the **Plugin Type** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/plugin-types	Get all plugin types	Creating or updating plugin types	--	--
/rest/plugin-types/<id>	Get single plugin type	--	--	--

This topic contains these sections:

Getting plugin types

The HTTP GET method is used to retrieve **Plugin Type** information. Queries for all objects and a single object are available.

Quick reference

```
GET http://localhost:8080/mws/rest/plugin-types/<id>?api-version=3
```

Get all plugin types

URLs and parameters

```
GET http://localhost:8080/mws/rest/plugin-types?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/plugin-types?api-version=3&fields=id
------------------------------------

{
  "totalCount": 2,
  "resultCount": 2,
  "results":   [
    {"id": "vCenter"},
    {"id": "Native"}
  ]
}
```

Get single plugin type

URLs and parameters

```
GET http://localhost:8080/mws/rest/plugin-types/<id>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response
------------------------------------

{
  "author": "Adaptive Computing Enterprises, Inc.",
  "commonsVersion": "0.9.3 > *",
  "description": "Polls a VMware® vCenter™ Server for information on the hypervisors
and virtual machines it manages.",
  "documentationLink": "",
  "email": "",
  "eventComponent": 1,
  "realizedEventComponent": 513,
  "id": "VCenter",
  "initialPlugins": { },
  "instances": [
    {"id":"vcenter"}
  ],
  "issueManagementLink": "",
  "license": "APACHE",
  "mwsVersion": "7.1.2 > *",
  "pollMethod": true,
  "scmLink": "",
  "title": "VCenter",
  "version": "1.0",
  "webServices": [ ],
  "website": "http://www.adaptivecomputing.com"
}
```

Creating or updating plugin types

The HTTP PUT method is used to create or update **Plugin Types**. The `Content-Type` HTTP header is used to determine if the request contains a single class file as plaintext or the binary data of a JAR file. Each request is explained in the following sections.

Quick reference

```
PUT http://localhost:8080/mws/rest/plugin-types?api-version=3[&reload-plugins=false]
```

There is a known issue with dynamically updating plugin types with typed field injection. For more information, see **Add or update plugin types** on page 1717.

Update plugin type (file)

URLs and parameters

```
PUT http://localhost:8080/mws/rest/plugin-types?api-version=3[&reload-plugins=false]
```

Parameter	Required	Type	Valid values	Description
reload-plugins	No	String	`true` or `false`	Reloads all plugins of this type on successful update. Defaults to `true`.

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

This function is idempotent, meaning it will create the **Plugin Type** if it does not exist or update it if it does. The request body is the actual contents of the class file to upload. This web service is an exception to most as it *requires* a content type of `application/x-groovy` or `text/plain`.

If the `application/x-groovy` or `text/plain` content types are not used in the request, it will be interpreted as JSON, resulting in a failure.

```
Plaintext upload
------------------------------------

package test

import com.adaptc.mws.plugins.*

class UploadPlugin {
       static author = "Adaptive Computing"
       static description = "A sample plugin class"
       String id

       public void configure() throws InvalidPluginConfigurationException {
               def myConfig = config
               def errors = []
               if (!myConfig.arbitraryKey)
                       errors << "Missing arbitraryKey!"
               if (errors)
                       throw new InvalidPluginConfigurationException(errors)
       }

       public def customService(Map params) {
               return params
       }
}
```

If using the curl library to perform plugin type uploading, the equivalent of the command-line option `--data-binary` must be used to send the request body. Otherwise compilation errors may be encountered when uploading the plugin type.

Sample response

The response of this task is the same as the **Get all plugin types** task. The reason that the return of this task is a list is to accommodate the possibility of uploading multiple plugin types in a single JAR file as explained in the next section.

Update plugin type (JAR)

URLs and parameters

```
PUT http://localhost:8080/mws/rest/plugin-types?api-version=3&jar-
filename=<filename.jar>[&reload-plugins=false]
```

Parameter	Required	Type	Valid values	Description
jar-filename	Yes	String	--	The filename of the JAR file that is being uploaded.
reload-plugins	No	String	`true` or `false`	Reloads all plugins of this type on successful update. Defaults to `true`.

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

This function is idempotent, meaning it will create the **Plugin Types** if they do not exist or update them if they do. The request body is the binary contents of the JAR file to upload. This web service is an exception to most as it *requires* a content type of `application/x-jar`.

If the `application/x-jar` content type is not used in the request, it will be interpreted as JSON, resulting in a failure.

If using the curl library to perform plugin type uploading, the equivalent of the command-line option `--data-binary` must be used to send the request body. Otherwise compilation errors may be encountered when uploading the plugin type.

Sample response

The response of this task is the same as the **Get all plugin types** task. Note that when using a JAR file, multiple plugin types may be uploaded in the same request.

Related topics

- **Fields: Plugin Types** on page 2035
- **Resources introduction** on page 1424
- **Plugins** on page 1577

Policies

This section describes behavior of the **Policies** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The **Fields: Policies** reference section contains the type and description of fields of all **Policies**.

Supported policies

Name	ID
Automatic VM Migration	auto-vm-migration
Fairshare	fairshare
Hypervisor Allocation Overcommit	hv-allocation-overcommit
Migration Exclusion List	migration-exclusion-list
Node Allocation	node-allocation

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/policies	Get all policies	--	--	--
/rest/policies/<id>	Get single policy	Modify policy	--	--

This topic contains these sections:

Getting policies

The HTTP GET method is used to retrieve **Policies** information.

Quick reference

```
GET http://localhost:8080/mws/rest/policies?api-version=3
```

Get all policies

URLs and parameters

```
GET http://localhost:8080/mws/rest/policies?api-version=3
```

Parameter	Required	Type	Description	Example
query	No	JSON	Query for specific results.	`query={"state":"DISABLED","conflicted":"false"}`
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	`sort={"id":-1}`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/policies?api-version=3&fields=id,state,conflicted
------------------------------------

{
  "totalCount": 4,
  "resultCount": 4,
  "results": [  {
    "conflicted": false,
    "state": "DISABLED",
    "id": "auto-vm-migration"
  },{
    "conflicted": false,
    "state": "DISABLED",
    "id": "hv-allocation-overcommit"
  },{
    "conflicted": false,
    "state": "DISABLED",
    "id": "node-allocation"
  },{
    "conflicted": false,
    "state": "DISABLED",
    "id": "migration-exclusion-list"
  }]
}
```

Get single policy

URLs and parameters

```
GET http://localhost:8080/mws/rest/policies/<id>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample responses

```
Auto VM Migration
------------------------------------

{
  "conflicted": false,
  "description": "Controls how virtual machines are automatically migrated.",
  "id": "auto-vm-migration",
  "name": "Auto VM Migration",
  "potentialConflicts": [],
  "priority": 1,
  "state": "DISABLED",
  "tags": [],
  "types": [],
  "version": 0,
  "genericMetricThresholds":{
       "GMETRIC1":1.3
  },
  "processorUtilizationThreshold":0.5,
  "memoryUtilizationThreshold":0.4
}
```

```
Fairshare
------------------------------------

{
    "conflicted": false,
    "decayFactor": 0.44,
    "depth": 4,
    "description": "Control job feasibility and priority decisions based on system
utilization targets for users, groups, accounts, classes, and QoS levels.",
    "intervalSeconds": 600,
    "name": "Fairshare",
    "potentialConflicts": [],
    "priority": 16,
    "state": "ENABLED",
    "tags": [],
    "types": [],
    "usageMetric": "DEDICATED_PROCESSOR_SECONDS_DELIVERED",
    "version": 3,
    "id": "fairshare"
}
```

```
Hypervisor Allocation Overcommit
------------------------------------

{
  "conflicted": false,
  "description": "Controls how hypervisors are overallocated with regards to
processors and memory.",
  "id": "hv-allocation-overcommit",
  "name": "Hypervisor Allocation Overcommit",
  "potentialConflicts": [],
  "priority": 2,
  "state": "DISABLED",
  "tags": [],
  "types": [],
  "version": 0,
  "processorAllocationLimit":29.5,
  "memoryAllocationLimit":1.2
}
```

```
Migration Exclusion List
------------------------------------

{
    "conflicted": false,
    "description": "Controls which machines are excluded from automatic live migration
operations.",
    "hvExclusionList": ["blade05", "blade02"],
    "name": "Migration Exclusion List",
    "potentialConflicts": [],
    "priority": 100,
    "state": "DISABLED",
    "tags": [],
    "types": [],
    "version": 1,
    "vmExclusionList": ["vm1", "vm5"],
    "id": "migration-exclusion-list"
}
```

```
Node Allocation
------------------------------------

{
  "conflicted": false,
  "description": "Controls how nodes are selected for workload placement.",
  "id": "node-allocation",
  "name": "Node Allocation",
  "potentialConflicts": [],
  "priority": 3,
  "state": "DISABLED",
  "tags": [],
  "types": [],
  "version": 0,
  "nodeAllocationAlgorithm": "CustomPriority",
  "customPriorityFunction": "-100*GMETRIC[vmcount]"
}
```

Modifying policies

The HTTP PUT method is used to modify **Policies.**

Quick reference

```
PUT http://localhost:8080/mws/rest/policies/<id>?api-version=3
```

Modify policy

URLs and parameters

```
PUT http://localhost:8080/mws/rest/policies/<id>?api-version=3[&change-mode=set]
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Additional URL parameters

URL parameters for modifying a Migration Exclusion Lists Policy.

Migration Exclusion Lists parameter	Required	Type	Valid values	Description
change-mode	No	String	set (default) add remove	If **set**, replace existing exclusion list(s) with the given one. If **add**, add the given VMs/HVs to the existing exclusion list(s). If **remove**, remove the given VMs/HVs from the existing exclusion list(s).

Request body

In general, the fields shown in the **Fields: Policies** reference section are *not* available for modification. However, the `state` field may be modified to a valid `PolicyState`. All other fields listed in the specific policy type sections (i.e. `AutoVMMigrationPolicy`) may be modified unless documented otherwise.

- The request body below shows all the fields that are available when modifying a Auto VM Migration Policy, along with some sample values.

```
JSON request body for Auto VM Migration Policy
------------------------------------

{
    "genericMetricThresholds": {
        "GENERICTHRESHOLD": 5
    },
    "memoryUtilizationThreshold": 0.5,
    "processorUtilizationThreshold": 0.4
}
```

- The request body below shows all the fields that are available when modifying a Fairshare Policy, along with some sample values.

```
JSON request body for Fairshare Policy
------------------------------------

{
    "decayFactor": 0.44,
    "depth": 4,
    "intervalSeconds": 600,
    "usageMetric": "DEDICATED_PROCESSOR_SECONDS_DELIVERED",
}
```

- The request body below shows all the fields that are available when modifying a Hypervisor Allocation Overcommit Policy, along with some sample values.

```
JSON request body for Hypervisor Allocation Overcommit Policy
------------------------------------

{
    "processorAllocationLimit":29.5,
    "memoryAllocationLimit":1.2
}
```

- The request body below shows all the fields that are available when modifying a Migration Exclusion Lists Policy, along with some sample values.

```
JSON request body for Migration Exclusion Lists Policy
------------------------------------

{
   "vmExclusionList" : ["vm1","vm3","vm5"],
   "hvExclusionList" : ["hv2","hv3","hv6"]
}
```

- The request body below shows all the fields that are available when modifying a Node Allocation Policy, along with some sample values.

```
JSON request body for Node Allocation Policy
------------------------------------

{
   "nodeAllocationAlgorithm" : "CustomPriority",
   "customPriorityFunction" : "-100*GMETRIC[vmcount]"
}
```

Sample response

```
JSON response
------------------------------------

{
    "messages": ["Policy auto-vm-migration updated"]
}
```

Samples

Enable the Auto VM Migration Policy and set values.

```
PUT http://localhost:8080/mws/rest/policies/auto-vm-migration?api-version=3
------------------------------------

{
  "state": "enabled",
  "migrationAlgorithmType": "overcommit",
  "processorUtilizationThreshold": 0.5,
  "memoryUtilizationThreshold": 0.4
}
```

As noted in the **Fields: Policies** reference section documentation for `AutoVMMigrationPolicy`, if the `state` is set to ENABLED, then the `migrationAlgorithmType` must *not* be set to NONE.

Restrictions

All policies:

- Fields cannot be modified while the policy is disabled. Enable the policy to modify the field.

Auto VM Migration

- Arbitrary metrics can be added to **genericMetricThresholds**, but they cannot be removed once added.
- The **migrationAlgorithmType** field cannot be modified while the policy is disabled. Enable the policy to modify the field.
- Moab is configured with a default limit of 10 generic metrics. If this limit is reached, such as when arbitrary metrics are added to **genericMetricThresholds**, the metric will not be reported. To increase this limit, set the `MAXGMETRIC` property in the Moab configuration file.

Fairshare

- Updating the **usageMetric** field will clear all credential-based fairshare interval data.

Related topics

- **Fields: Policies** on page 2039
- **Fairshare** on page 1596
- **Resources introduction** on page 1424

Fairshare

This section describes behavior of the **Fairshare** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The supported methods table below requires each resource to be accessed with a URL parameter of `api-version=3`.

For more information, see **Requesting specific API versions** on page 1406.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/policies/fairshare	**Get all fairshare interval data** on page 1598	--	--	--
/rest/policies/fairshare/<credentialType>	**Get all fairshare interval data for a single credential type** on page 1601	--	--	--
/rest/policies/fairshare/<credentialType>/<name>	**Get all fairshare interval data for a single credential** on page 1604	--	--	--

This topic contains these sections:

Getting credential-based fairshare interval data

The HTTP GET method is used to retrieve **Policies** information.

Quick reference

```
GET http://localhost:8080/mws/rest/policies/fairshare/credentials?api-version=3
GET http://localhost:8080/mws/rest/policies/fairshare/credentials/accounts?api-
version=3
GET http://localhost:8080/mws/rest/policies/fairshare/credentials/classes?api-
version=3
GET http://localhost:8080/mws/rest/policies/fairshare/credentials/groups?api-version=3
GET http://localhost:8080/mws/rest/policies/fairshare/credentials/qoses?api-version=3
GET http://localhost:8080/mws/rest/policies/fairshare/credentials/users?api-version=3
GET http://localhost:8080/mws/rest/policies/fairshare/credentials/accounts/<name>?api-
version=3
GET http://localhost:8080/mws/rest/policies/fairshare/credentials/classes/<name>?api-
version=3
GET http://localhost:8080/mws/rest/policies/fairshare/credentials/groups/<name>?api-
version=3
GET http://localhost:8080/mws/rest/policies/fairshare/credentials/qoses/<name>?api-
version=3
GET http://localhost:8080/mws/rest/policies/fairshare/credentials/users/<name>?api-
version=3
```

Get all fairshare interval data

URLs and parameters

```
GET http://localhost:8080/mws/rest/policies/fairshare/credentials?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/policies/fairshare/credentials?api-version=3
------------------------------------

{
  "totalCount": 4,
  "resultCount": 4,
  "results":   [
    {
      "name": "jbethune",
      "target_type": null,
      "target": null,
      "interval_data":       [
        0,
        0,
        0,
        0
      ],
      "credential_type": "USER"
    },
    {
      "name": "jfoote",
      "target_type": null,
      "target": null,
      "interval_data":       [
        2104.16,
        2377.06,
        2240.1,
        2550
      ],
      "credential_type": "GROUP"
    },
    {
      "name": "NOGROUP",
      "target_type": null,
      "target": null,
      "interval_data":       [
        0,
        0,
        0,
        0
      ],
      "credential_type": "GROUP"
    },
    {
      "name": "DEFAULT",
      "target_type": null,
      "target": null,
      "interval_data":       [
        0,
        0,
        0,
        0
      ],
      "credential_type": "ACCOUNT"
    },
    {
      "name": "Administration",
      "target_type": null,
      "target": null,
      "interval_data":       [
        5256.28,
        6247.05,
        6048.27,
        6948.67
      ],
      "credential_type": "ACCOUNT"
    }
```

```
  ]
}
```

Get all fairshare interval data for a single credential type

URLs and parameters

```
GET
http://localhost:8080/mws/rest/policies/fairshare/credentials/<accounts|classes|groups
|qoses|users>?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample responses

```
GET http://localhost:8080/mws/rest/policies/fairshare/credentials/accounts?api-
version=3
------------------------------------
{
  "totalCount": 6,
  "resultCount": 6,
  "results":   [
    {
      "name": "jbethune",
      "target_type": null,
      "target": null,
      "interval_data":       [
        0,
        0,
        0,
        0
      ],
      "credential_type": "ACCOUNT"
    },
    {
      "name": "Administration",
      "target_type": null,
      "target": null,
      "interval_data":       [
        5256.28,
        6247.05,
        6048.27,
        6948.67
      ],
      "credential_type": "ACCOUNT"
    },
    {
      "name": "Shared",
      "target_type": null,
      "target": null,
      "interval_data":       [
        4261.38,
        4951.09,
        4480.2,
        5000.54
      ],
      "credential_type": "ACCOUNT"
    },
    {
      "name": "Engineering",
      "target_type": null,
      "target": null,
      "interval_data":       [
        15034.64,
        17245.93,
        15008.67,
        17085
      ],
      "credential_type": "ACCOUNT"
    },
    {
      "name": "Test",
      "target_type": null,
      "target": null,
      "interval_data":       [
        1808.08,
        1873.96,
        1568.07,
        1757.33
      ],
      "credential_type": "ACCOUNT"
    },
```

```
    {
      "name": "Research",
      "target_type": null,
      "target": null,
      "interval_data":       [
        47606.8,
        52861.83,
        46370.07,
        52785
      ],
      "credential_type": "ACCOUNT"
    }
  ]
}
```

Get all fairshare interval data for a single credential

URLs and parameters

```
GET
http://localhost:8080/mws/rest/policies/fairshare/credentials/<accounts|classes|groups
|qoses|users>/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
name	Yes	String	--	The unique name of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET
http://localhost:8080/mws/rest/policies/fairshare/credentials/accounts/DEFAULT?api-
version=3
------------------------------------
{
  "name": "DEFAULT",
  "target_type": null,
  "target": null,
  "interval_data":   [
    0,
    0,
    0,
    0
  ],
  "credential_type": "ACCOUNT"
}
```

Related topics

- **Policies** on page 1589
- **Resources introduction** on page 1424

Principals

This section describes behavior of the **Principal** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The **Fields: Principals** reference contains the type and description of all fields in the **Principal** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/principals	**Get all principals**	--	**Create single principal**	--
/rest/principals/<id>	**Get single principal**	**Modify single principal**	--	**Delete single principal**
/rest/principals/<name>	**Get single principal**	**Modify single principal**	--	**Delete single principal**

This topic contains these sections:

Getting principals

The HTTP GET method is used to retrieve **Principal** information. You can query all objects or a single object.

Quick reference

```
GET http://localhost:8080/mws/rest/principals?api-version=3[&query={"field":"value"}
&sort={"field":<1|-1>}]
GET http://localhost:8080/mws/rest/principals/<id>?api-version=3
GET http://localhost:8080/mws/rest/principals/<name>?api-version=3
```

Get all principals

URLs and parameters

```
GET http://localhost:8080/mws/rest/principals?api-version=3[&query={"field":"value"}
&sort={"field":<1|-1>}]
```

Parameter	Required	Type	Description	Example
query	No	JSON	Queries for specific results. It is possible to query principals by one or more fields based on MongoDB query syntax.	`query={"name":"Acme Principal"}`
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	`sort={"name":-1}`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/principals?api-version=3&fields=name,group
------------------------------------

{
  "totalCount": 2,
  "resultCount": 2,
  "results":    [
    {
      "groups": [      {
        "name": "CN=Engineering,CN=Users,DC=corp,DC=cloud,DC=dev",
        "type": "LDAPGROUP"
      }],
         "name": "Engineering-Principal"
    },
       {
         "groups": [   {
        "name": "CN=Marketing,CN=Users,DC=corp,DC=cloud,DC=dev",
        "type": "LDAPGROUP"
      }],
         "name": "Marketing-Principal"
    }
  ]
}
```

Sorting and Querying

See the sorting and querying sections of **Global URL parameters** on page 1403.

Get single principal

URLs and parameters

```
GET http://localhost:8080/mws/rest/principals/<id>?api-version=3
GET http://localhost:8080/mws/rest/principals/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the principal.
name	Yes	String	--	The name of the principal.

You must specify either id or name, but you do not have to specify both.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/principals/principal8?api-version=3
------------------------------------

{
         "attachedRoles": [  {
               "description": "This is a role for normal users in the Acme BU Group.",
               "id": "5033b8eae4b09cc61bedb895",
               "name": "Acme-User-Role",
               "permissions":   [
                                {
                       "action": "read",
                       "description": "The permission to read all nodes",
                       "id": "5033b842e4b09cc61bedb818",
                       "label": "",
                       "resource": "nodes",
                       "resourceFilter": null,
                       "type": "api",
                       "version": 1
                 },
               ],
               "version": 2
         }],
         "description": "Principal 8",
         "groups": [  {
               "name": "CN=Engineering,CN=Users,DC=corp,DC=cloud,DC=dev",
               "type": "LDAPGROUP"
         }],
         "id": "5033d33fe4b018b28745fecd",
         "name": "principal8",
         "users":   [
               {
                 "name": "jhammon",
                 "type": "LDAP"
               },
               {
                 "name": "bjones",
                 "type": "LDAP"
               }
         ],
       "version": 0
}
```

Creating principals

The HTTP POST method is used to submit **Principals**.

Quick reference

```
POST http://localhost:8080/mws/rest/principals?api-version=3
```

Create single principal

URLs and parameters

```
POST http://localhost:8080/mws/rest/principals?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

The **name** field is required and must contain only letters, digits, periods, dashes, and underscores.

The **attachedRoles** field expects an array of Role IDs *or* names:

The following is an example request body to create a principal:

```
POST http://localhost:8080/mws/rest/principals?api-version=3
------------------------------------

{
       "name" : "Acme-Principal",
       "attachedRoles" : [{"name":"Acme-User-Role"}],
       "description" : "A cool principal",
       "groups" : [{"name": "CN=Engineering,CN=Users,DC=corp,DC=cloud,DC=dev",
"type":"LDAPGROUP"}],
       "users" : [{
               "name" : "john",
               "type" : "LDAP"
       } ]
}
```

Sample response

If the request was successful, the response body is the new principal that was created, exactly as shown in **Get single principal**. On failure, the response is an error message.

Modifying principals

The HTTP PUT method is used to modify **Principals**.

Quick reference

```
PUT http://localhost:8080/mws/rest/principals/<id>?api-version=3
PUT http://localhost:8080/mws/rest/principals/<name>?api-version=3
```

Modify single principal

URLs and parameters

```
PUT http://localhost:8080/mws/rest/principals/<id>?api-version=3
PUT http://localhost:8080/mws/rest/principals/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the Principal.

Parameter	Required	Type	Valid values	Description
name	Yes	String	--	The name of the Principal. The **name** field must contain only letters, digits, periods, dashes, and underscores.
change-mode	Yes	String	add remove set (default)	If **add**, add the given objects (ldapGroups, ldapOUs, etc.) to the objects that already exist. If **remove**, delete the given objects from the objects that already exist. If **set**, add the given objects (ldapGroups, ldapOUs, etc.) and remove the objects that already exist.

See **Global URL parameters** on page 1403 for available URL parameters.

You must specify either **id** or **name**, but you do not have to specify both.

The **attachedRoles** field expects an array of Role IDs *or* names:

Example request

```
PUT http://locahost/mws/rest/principals/Acme-Principal?api-version=3
-----------------------------------
{
       "groups" : [ {
               "name" : "CN=Marketing,CN=Users,DC=mycompany,DC=com",
               "type" : "LDAPGROUP"
       },{
               "name" : "CN=Sales,CN=Users,DC=mycompany,DC=com",
               "type" : "LDAPGROUP"
       }],
       "users" : [ {
               "name" : "jhammon",
               "type" : "LDAP"
       }]
}
```

The **version** field contains the current version of the database entry. This field cannot be updated directly. However, if **version** is included in the modify request, it will be used to verify that another client did not update the object between the time that the data was retrieved and the modify request was delivered.

Sample response

If the request was successful, the response body is the modified principal as shown in **Get single principal**. On failure, the response is an error message.

Deleting principals

The HTTP DELETE method is used to delete **Principals**.

Quick reference

```
DELETE http://localhost:8080/mws/rest/principals/<id>?api-version=3
DELETE http://localhost:8080/mws/rest/principals/<name>?api-version=3
```

Delete single principal

URLs and parameters

```
DELETE http://localhost:8080/mws/rest/principals/<id>?api-version=3
DELETE http://localhost:8080/mws/rest/principals/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the principal.
name	Yes	String	--	The name of the principal.

See **Global URL parameters** on page 1403 for available URL parameters.

You must specify either **id** or **name**, but you do not have to specify both.

Sample response

```
JSON response
------------------------------------

{}
```

Related topics

- **Fields: Principals** on page 2065
- **Resources introduction** on page 1424

Priority

This section describes behavior of the **priority** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/priority	**Get all priorities** on page 1612	**Modify priorities** on page 1614	--	--

This topic contains these sections:

Getting priorities

The HTTP GET method is used to retrieve **priority** information.

Quick reference

```
GET http://localhost:8080/mws/rest/priority?api-version=3
```

Get all priorities

URLs and parameters

```
GET http://localhost:8080/mws/rest/priority?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response


```
------------------------------------
{
  "service":   {
    "weight": 1,
    "queue_time": 1,
    "x_factor": 0,
    "policy_violation": 0,
    "bypass": 0
  },
  "target":   {
    "weight": 1,
    "queue_time": 0,
    "x_factor": 0
  },
  "credential":   {
    "weight": 1,
    "user_credential": 0,
    "group_credential": 0,
    "account_credential": 0,
    "class_credential": 0,
    "qos_credential": 0
  },
  "attribute":   {
    "weight": 1,
    "attribute": 0,
    "state": 0
  },
  "fairshare":   {
    "weight": 1,
    "user_credential": 1000,
    "group_credential": 0,
    "account_credential": 0,
    "class_credential": 0,
    "qos_credential": 0,
    "jobs_per_user": 0,
    "processor_seconds_per_user": 0,
    "processors_per_user": 0
  },
  "resource":   {
    "weight": 1,
    "node": 0,
    "disk": 0,
    "memory": 0,
    "swap": 0,
    "processor_equivalent_seconds": 0,
    "walltime": 0
  },
  "usage":   {
    "weight": 1,
    "consumed": 0,
    "remaining": 0,
    "percentage_consumed": 0
  }
}
```


Modifying priorities

The HTTP PUT method is used to update **priority** information.

Quick reference

```
PUT http://localhost:8080/mws/rest/priority?api-version=3
```

Modify priorities

URLs and parameters

```
PUT http://localhost:8080/mws/rest/priority?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample body

```
PUT http://localhost:8080/mws/rest/priority?api-version=3
------------------------------------
{
  "service":   {
    "weight": 2,
    "queue_time": 2,
    "x_factor": 1,
    "policy_violation": 1,
    "bypass": 1
  }
}
```

Related topics

- **Resources introduction** on page 1424

Reports

This section describes behavior of the reporting framework in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The **Fields: Reports**, **Fields: Report Samples**, and **Fields: Report Datapoints** reference sections contain the type and description of all fields in the **Report**, **Sample**, and **Datapoint** objects. They also contains details regarding which fields are valid during PUT and POST actions.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/reports	Get all reports (no data)	--	Create report	Delete report
/rest/reports/<name>	Get single report (with data)	--	--	--
/rest/reports/<id>	Get single report (with data)	--	--	--

Resource	GET	PUT	POST	DELETE
/rest/reports/<name>/datapoints	Get datapoints for single report	--	--	--
/rest/reports/<id>/datapoints	Get datapoints for single report	--	--	--
/rest/reports/<name>/samples	Get samples for report	--	Create samples for report	--
/rest/reports/<id>/samples	Get samples for report	--	Create samples for report	--

This topic contains these sections:

Getting reports

The HTTP GET method is used to retrieve **Report** information. Queries for all reports with no attached data and a single report with associated data are available.

Quick reference

```
GET http://localhost:8080/mws/rest/reports?api-version=3[&query={"field":"value"}
&sort={"field":<1|-1>}]
GET http://localhost:8080/mws/rest/reports/<id>?api-version=3
GET http://localhost:8080/mws/rest/reports/<name>?api-version=3
```

Get all reports (no data)

URLs and parameters

```
GET http://localhost:8080/mws/rest/reports?api-version=3[&query={"field":"value"}
&sort={"field":<1|-1>}]
```

Parameter	Required	Type	Description	Example
query	No	JSON	Queries for specific results. It is possible to query reports by one or more fields based on MongoDB query syntax.	`query={"reportS-ize":4}`
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	`sort={"name":-1}`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response
------------------------------------

{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
    "id": "3efe5c670be86ba8560397ff",
    "name": "cpu-util"
       …
  }]
}
```

Samples

```
GET http://localhost:8080/mws/rest/reports?api-version=3&fields=id,name
------------------------------------

{
  "totalCount": 3,
  "resultCount": 3,
  "results":   [
        {
      "id": "3efe5c670be86ba8560397ff",
      "name": "cpu-util"
    },
        {
      "id": "3efe5c670be86ba856039800",
      "name": "cpu-temp"
    },
        {
      "id": "3efe5c670be86ba856039801",
      "name": "cpu-load"
    }
  ]
}
```

Get single report (with data)

URLs and parameters

```
GET http://localhost:8080/mws/rest/reports/<id>?api-version=3
GET http://localhost:8080/mws/rest/reports/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the report.
name	Yes	String	--	The name of the report.

Only one of **id** or **name** are required.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

In the example below, the first datapoint has a `null` data element, which means that the `minimumSampleSize` configured for the report was not met when consolidating the datapoint. The second datapoint contains actual data.

```
JSON response
------------------------------------

{
  "consolidationFunction": "average",
  "datapointDuration": 15,
  "datapoints":   [
        {
      "endDate": "2011-12-02 17:28:22 UTC",
      "startDate": "2011-12-02 17:28:22 UTC",
      "firstSampleDate": null,
      "lastSampleDate": null,
      "data": null
    },
        {
      "endDate": "2011-12-02 17:28:23 UTC",
      "startDate": "2011-12-02 17:28:37 UTC",
      "firstSampleDate": "2011-12-02 17:28:23 UTC",
      "lastSampleDate": "2011-12-02 17:28:30 UTC",
      "data":       {
        "utilization": 99.89,
        "time": 27.433333333333337
      }
    }
  ],
  "description": "Example of CPU utilization reporting",
  "id": "3efe5c670be86ba8560397ff",
  "keepSamples": false,
  "minimumSampleSize": 1,
  "name": "cpu-util",
  "reportSize": 2
}
```

Get datapoints for single report

URLs and parameters

```
GET http://localhost:8080/mws/rest/reports/<id>/datapoints?api-version=3[&query=
{"field":"value"}&sort={"field":<1|-1>}]
GET http://localhost:8080/mws/rest/reports/<name>/datapoints?api-version=3[&query=
{"field":"value"}&sort={"field":<1|-1>}]
```

Parameter	Required	Type	Description	Example
id	Yes	String	The unique identifier of the report.	--
name	Yes	String	The name of the report.	--
query	No	JSON	Queries for specific results.	`query={"reportS-ize":4}`
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	`sort={"name":-1}`

> Only one of **id** or **name** are required.

It is possible to query reports by one or more fields based on MongoDB query syntax.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

This function is exactly the same as **Get single report (with data)**. No report metadata (i.e. `description`, `minimumSampleSize`, etc.) is returned.


```
JSON response
------------------------------------

{
  "resultCount":1,
  "totalCount":1,
  "results":[
        {
      "endDate": "2011-12-02 17:28:22 UTC",
      "startDate": "2011-12-02 17:28:22 UTC",
      "firstSampleDate": null,
      "lastSampleDate": null,
      "data": null
    },
        {
      "endDate": "2011-12-02 17:28:37 UTC",
      "startDate": "2011-12-02 17:28:37 UTC",
      "firstSampleDate": "2011-12-02 17:28:23 UTC",
      "lastSampleDate": "2011-12-02 17:28:23 UTC",
      "data":       {
        "utilization": 99.89,
        "time": 27.433333333333337
      }
    }
  ]
}
```


Getting samples for reports

The HTTP GET method is used to retrieve **Sample** information.

Quick reference

```
GET http://localhost:8080/mws/rest/reports/<id>/samples?api-version=3[&query=
{"field":"value"}&sort={"field":<1|-1>}]
GET http://localhost:8080/mws/rest/reports/<name>/samples?api-version=3[&query=
{"field":"value"}&sort={"field":<1|-1>}]
```

Get samples for report

URLs and parameters

```
GET http://localhost:8080/mws/rest/reports/<id>/samples?api-version=3[&query=
{"field":"value"}&sort={"field":<1|-1>}]
GET http://localhost:8080/mws/rest/reports/<name>/samples?api-version=3[&query=
{"field":"value"}&sort={"field":<1|-1>}]
```

Parameter	Required	Type	Description	Example
id	Yes	String	The unique identifier of the report.	--
name	Yes	String	The name of the report.	--
query	No	JSON	Queries for specific results.	`query={"reportS-ize":4}`
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	`sort={"name":-1}`

Only one of **id** or **name** are required.

It is possible to query reports by one or more fields based on MongoDB query syntax.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response
------------------------------------

{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
       "timestamp": "2011-12-02 17:28:37 UTC"
       "data":{
               "cpu1":2.3,
               "cpu2":1.2,
               "cpu3":0.0,
               "cpu4":12.1
       },
       …
  }]
}
```

Creating reports

The HTTP POST method is used to create **Reports**. Operations are available to create reports with or without historical datapoints.

Quick reference

```
POST http://localhost:8080/mws/rest/reports?api-version=3
```

Create report

URLs and parameters

```
POST http://localhost:8080/mws/rest/reports?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

To create a report, several fields are required as documented in **Fields: Reports** on page 2074.

The request body below shows all the fields that are available during report creation.

```
JSON request body
------------------------------------

{
	"name":"cpu-util",
	"description":"An example report on cpu utilization",
	"consolidationFunction":"average",
	"datapointDuration":15,
	"minimumSampleSize":1,
	"reportSize":2,
	"keepSamples":true,
	"reportDocumentSize":1024,
	"datapoints":[
		{
			"startDate":"2011-12-01 19:16:57 UTC",
			"endDate":"2011-12-01 19:16:57 UTC",
			"data":{
				"time":30,
				"util":99.98
			}
		}
	]
}
```

Sample response

```
{
	"messages":["Report cpu-util created"],
	"id":"3efe5c670be86ba8560397ff",
	"name":"cpu-util"
}
```

Samples

```
POST http://localhost:8080/mws/rest/reports?api-version=3 (Minimal report without
datapoints)
-----------------------------------

{
        "name":"cpu-util",
        "datapointDuration":15,
        "reportSize":2
}
```

Creating samples

The HTTP POST method is used to create samples for **Reports**.

Quick reference

```
POST http://localhost:8080/mws/rest/reports?api-version=3
```

Create samples for report

URLs and parameters

```
POST http://localhost:8080/mws/rest/reports/<id>/samples?api-version=3
POST http://localhost:8080/mws/rest/reports/<name>/samples?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the report.
name	Yes	String	--	The name of the report.

Only one of **id** or **name** are required.

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

To create samples for a report, simply send data and an optional timestamp to the URL above. The request body below shows all the fields that are available during sample creation. Note that the `data` field can contain arbitrary JSON.

```
JSON request body
-----------------------------------

{
	"timestamp":"2011-12-01 19:16:57 UTC",
	"agent":"my agent",
	"data":{
		"cpu1":2.3,
		"cpu2":1.2,
		"cpu3":0.0,
		"cpu4":12.1
	}
}
```

Sample response

```
{"messages":["1 sample(s) created for report cpu-util"]}
```

Deleting reports

The HTTP DELETE method is used to delete **Reports**.

Quick reference

```
DELETE http://localhost:8080/mws/rest/reports/<id>?api-version=3
DELETE http://localhost:8080/mws/rest/reports/<name>?api-version=3
```

Delete report

URLs and parameters

```
DELETE http://localhost:8080/mws/rest/reports/<id>?api-version=3
DELETE http://localhost:8080/mws/rest/reports/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the report.
name	Yes	String	--	The name of the report.

Only one of **id** or **name** are required.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response
-----------------------------------

{"messages":["Report cpu-util deleted"]}
```

Related topics

- **Fields: Reports** on page 2074
- **Resources introduction** on page 1424

Reservations

This section describes behavior of the **Reservations** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The **Fields: Reservations** reference contains the type and description of all fields in the **Reservations** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/reservations	Get all reservations	--	Create reservation	--
/rest/reservations/<id>	Get single reservation	Modify reservation	--	Release reservation

This topic contains these sections:

Getting reservations

The HTTP GET method is used to retrieve **Reservation** information. Queries for all objects and a single object are available.

Quick reference

```
GET http://localhost:8080/mws/rest/reservations/<id>?api-version=3
```

Restrictions

Only admin or user reservations are returned with this call.

Get all reservations

URLs and parameters

```
GET http://localhost:8080/mws/rest/reservations?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/reservations?api-version=3&fields=id
------------------------------------

{
  "totalCount": 3,
  "resultCount": 3,
  "results":   [
    {"id": "system.1"},
    {"id": "system.2"},
    {"id": "system.3"}
  ]
}
```

Get single reservation

URLs and parameters

```
GET http://localhost:8080/mws/rest/reservations/<id>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response
------------------------------------

{
  "accountingAccount": "",
  "accountingGroup": "",
  "accountingQOS": "",
  "accountingUser": "root",
  "aclRules": [  {
    "affinity": "NEUTRAL",
    "comparator": "LEXIGRAPHIC_EQUAL",
    "type": "RESERVATION_ID",
    "value": "system.43"
  }],
  "allocatedNodeCount": 1,
  "allocatedProcessorCount": 8,
  "allocatedTaskCount": 1,
  "allocatedNodes": [
       {"id":"node001"}
  ],
  "comments": "",
  "creationDate": null,
  "duration": 200000000,
  "endDate": "2018-03-17 16:49:10 UTC",
  "excludeJobs":   [
    "job1",
    "job2"
  ],
  "expireDate": null,
  "flags":   [
    "REQFULL",
    "ISACTIVE",
    "ISCLOSED"
  ],
  "globalId": "",
  "hostListExpression": "",
  "id": "system.43",
  "idPrefix": "",
  "isActive": true,
  "isTracked": false,
  "label": "",
  "maxTasks": 0,
  "messages": [],
  "owner":   {
    "name": "adaptive",
    "type": "USER"
  },
  "partitionId": "switchB",
  "profile": "",
  "requirements":   {
    "architecture": "",
    "featureList":     [
      "feature1",
      "feature2"
    ],
    "featureMode": "",
    "memory": 0,
    "nodeCount": 0,
    "nodeIds": ["node001:1"],
    "os": "",
    "taskCount": 1
  },
```

```
  "reservationGroup": "",
  "resources": {"PROCS": 0},
  "startDate": "2011-11-14 20:15:50 UTC",
  "statistics":   {
    "caps": 0,
    "cips": 2659.52,
    "taps": 0,
    "tips": 0
  },
  "subType": "Other",
  "taskCount": 0,
  "trigger": null,
  "triggerIds": [],
  "uniqueIndex": "",
  "variables": {}
}
```

Creating reservations

The HTTP POST method is used to create **Reservations**.

Quick reference

```
POST http://localhost:8080/mws/rest/reservations?api-version=3
```

Create reservation

URLs and parameters

```
POST http://localhost:8080/mws/rest/reservations?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

The request body below shows all the fields that are available when creating a **Reservation**, along with some sample values.

```
JSON request body
------------------------------------

{
  "accountingAccount": "",
  "accountingGroup": "",
  "accountingQOS": "",
  "accountingUser": "root",
  "aclRules": [  {
    "affinity": "POSITIVE",
    "comparator": "LEXIGRAPHIC_EQUAL",
    "type": "GROUP",
    "value": "staff"
  }],
  "comments": "",
  "duration": 200000000,
  "endDate": "2018-03-17 16:49:10 UTC",
  "excludeJobs":   [
    "job1",
    "job2"
  ],
  "flags":   [
    "SPACEFLEX",
    "ACLOVERLAP",
    "SINGLEUSE"
  ],
  "hostListExpression": "",
  "idPrefix": "",
  "label": "myreservation",
  "owner":   {
    "name": "adaptive",
    "type": "USER"
  },
  "partitionId": "",
  "profile": "",
  "requirements":   {
    "architecture": "",
    "featureList":     [
      "feature1",
      "feature2"
    ],
    "memory": 0,
    "os": "",
    "taskCount": 1
  },
  "reservationGroup": "",
  "resources":   {
    "PROCS": 2,
    "MEM": 1024,
    "DISK": 1024,
    "SWAP": 1024,
    "other1": 17,
    "other2": 42
  },
  "startDate": "2011-11-14 20:15:50 UTC",
  "subType": "Other",
  "trigger": {
    "eventType":"START",
    "actionType":"EXEC",
    "action":"date"
  },
  "variables":   {
```

```
    "var1": "val1",
    "var2": "val2"
  }
}
```

This example is to create a reservation if no conflicting reservations are found. (This is the equivalent to `mrsvctl -c -h node01 -E`.)

```
JSON request body
------------------------------------

{
  "flags":   [
    "DEDICATEDRESOURCE"
  ],
  "hostListExpression": "node01"
}
```

Sample response

```
JSON Response for successful POST
------------------------------------

{"id": "system.44"}
```

Modifying reservations

The HTTP PUT method is used to modify **Reservations**.

Quick reference

```
PUT http://localhost:8080/mws/rest/reservations/<id>?api-version=3&change-
mode=<add|remove|set>
```

Modify reservation

URLs and parameters

```
PUT http://localhost:8080/mws/rest/reservations/<id>?api-version=3&change-
mode=<add|remove|set>
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.
change-mode	Yes	String	add remove set	If **add**, add the given variables to the variables that already exist. If **remove**, delete the given variables from the variables that already exist. If **set**, replace all existing variables with the given variables.

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

The request body below shows all the fields that are available when modifying a **Reservation**, along with some sample values.

```
JSON request body for reservation modify
------------------------------------

{
  "variables":   {
    "var1": "val1",
    "var2": "val2"
  }
}
```

Sample response

This message may not match the message returned from Moab exactly, but is given as an example of the structure of the response.

```
JSON response
------------------------------------

{"messages":["reservation 'system.43' attribute 'Variable' changed."]}
```

Restrictions

You can change the ACL Rules on a reservation, but not using this resource. See **Create or update ACL** on page 1427.

Releasing reservations

The HTTP DELETE method is used to release **Reservations**.

Quick reference

```
DELETE http://localhost:8080/mws/rest/reservations/<id>?api-version=3
```

Release reservation

URLs and parameters

```
DELETE http://localhost:8080/mws/rest/reservations/<id>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON Response for successful DELETE
------------------------------------

{}
```

Related topics

- **Fields: Reservations** on page 2080
- **Resources introduction** on page 1424

Resource types

This section describes behavior of the **Resource Type** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The **Fields: Resource Types** reference contains the type and description of all fields in the **Resource Type** object.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/resource-types	Get all resource types	--	--	--

This topic contains these sections:

Getting resource types

The HTTP GET method is used to retrieve **Resource Type** information.

Quick reference

```
GET http://localhost:8080/mws/rest/resource-types?api-version=3
```

Get all resource types

URLs and parameters

```
GET http://localhost:8080/mws/rest/resource-types?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/resource-types?api-version=3&fields=id
------------------------------------

{
  "totalCount": 1,
  "resultCount": 1,
  "results":   [
    {"id": "throttle_migrate"}
  ]
}
```

Related topics

- **Fields: Resource Types** on page 2120
- **Resources introduction** on page 1424

Roles

This section describes behavior of the **Role** resource in Moab Web Services. The role resource is used to control access to MWS resources based on the proxy-user. Each role is attached to a principal and contains a list of proxy-user permissions that the group can use in MWS. This section describes the URLs, request bodies, and responses delivered to and from MWS.

The **Fields: Roles** reference section contains the type and description of all fields in the **Role** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/roles	**Get all roles**	--	**Create single role**	--
/rest/roles/<id>	**Get single role**	**Modify single role**	--	**Deleting roles**
/rest/roles/<name>	**Get single role**	**Modify single role**	--	**Delete single role**

This topic contains these sections:

Getting roles

The HTTP GET method is used to retrieve **Role** information. You can query all objects or a single object.

Quick reference

```
GET http://localhost:8080/mws/rest/roles?api-version=3[&query={"field":"value"}&sort=
{"field":<1|-1>}]
GET http://localhost:8080/mws/rest/roles/<id>?api-version=3
GET http://localhost:8080/mws/rest/roles/<name>?api-version=3
```

Get all roles

URLs and parameters

```
GET http://localhost:8080/mws/rest/roles?api-version=3[&query={"field":"value"}&sort=
{"field":<1|-1>}]
```

Parameter	Required	Type	Valid values	Description	Example
query	No	JSON	--	Queries for specific results. It is possible to query roles by one or more fields based on MongoDB query syntax.	`query={"name":"Acme-User-Role"}`
sort	No	JSON	--	Sort the results. Use `1` for ascending and `-1` for descending.	`sort={"name":-1}`

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/roles?api-version=3&fields=id,name
------------------------------------

{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
    "id": "4fa197e68ca30fc605dd1cf0",
    "name": "Acme-User-Role"
  }]
}
```

Sorting and querying

See the sorting and querying sections of **Global URL parameters** on page 1403.

Get single role

URLs and parameters

```
GET http://localhost:8080/mws/rest/roles/<id>?api-version=3
GET http://localhost:8080/mws/rest/roles/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the Role.
name	Yes	String	--	The name of the Role.

You must specify either **id** or **name**, but you do not have to specify both.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/roles/Acme-User-Role?api-version=3
------------------------------------

{
	"description" : "This is a role for normal users in the Acme BU Group.",
	"id" : "5022e695e4b073f54e47c28d",
	"name" : "Acme-User-Role",
	"permissions" : [ {
		"action" : "create",
		"description" : "The permission to create all charts.",
		"id" : "5022e695e4b073f54e47c28e",
		"label" : "Create Chart",
		"resource" : "chart",
		"resourceFilter" : null,
		"type" : "custom",
		"scope" : "GLOBAL",
		"version" : 0
	}, {
		"action" : "read",
		"description" : "The permission to view all charts.",
		"id" : "5022e695e4b073f54e47c28f",
		"label" : "View Chart",
		"resource" : "chart",
		"resourceFilter" : null,
		"type" : "custom",
		"scope" : "GLOBAL",
		"version" : 0
	}, {
		"action" : "update",
		"description" : "The permission to modify the africa chart.",
		"id" : "5022e695e4b073f54e47c290",
		"label" : "Modify Africa Chart",
		"resource" : "chart",
		"resourceFilter" : {
			"name" : "africa"
		},
		"type" : "custom",
		"scope" : "GLOBAL",
		"version" : 0
	}, {
		"action" : "read",
		"description" : "The permissions to view John's services.",
		"id" : "5022e695e4b073f54e47c291",
		"label" : "Read John's services",
		"resource" : "services",
		"resourceFilter" : {
			"user":"john"
		},
		"type" : "api",
		"scope" : "GLOBAL",
		"version" : 0
	} ],
	"version" : 2
}
```

Creating roles

The HTTP POST method is used to submit **Roles**.

Quick reference

```
POST http://localhost:8080/mws/rest/roles?api-version=3
```

Create single role

URLs and parameters

```
POST http://localhost:8080/mws/rest/roles?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Request body

The **name** field is required and must contain only letters, digits, periods, dashes, and underscores.

The following is an example of a request body to create a role:

```
POST http://localhost:8080/mws/rest/roles?api-version=3
------------------------------------

{
	"name" : "Acme-User-Role",
	"description" : "This is a role for normal users in the Acme BU Group.",
	"permissions" :
	[
		{
		"id" : "4fa197e68ca30fc605dd1cf0"
		},
		{
		"id" : "4fa197e68ca30fc605dd1df2"
		}
	]
}
```

Sample response

If the request was successful, the response body is the new role that was created, exactly as shown in **Get single role**. On failure, the response is an error message.

Samples

The **permissions** field only expects an array of permission IDs, as shown in the following example:

```
Example payload of role with 2 permissions
------------------------------------

{
	"name" : "Acme-User-Role",
	"description" : "This is a role for normal users in the Acme BU Group.",
	"permissions" :
	[
		{
		"id" : "4fa197e68ca30fc605dd1cf0"
		}
	]
}
```

Modifying roles

The HTTP PUT method is used to modify **Roles**.

Quick reference

```
PUT http://localhost:8080/mws/rest/roles/<id>?api-version=3
PUT http://localhost:8080/mws/rest/roles/<name>?api-version=3
```

Modify single role

URLs and parameters

```
PUT http://localhost:8080/mws/rest/roles/<id>?api-version=3
PUT http://localhost:8080/mws/rest/roles/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the Role.
name	Yes	String	--	The name of the Role. The **name** field must contain only letters, digits, periods, dashes, and underscores.
change-mode	No	String	add remove set (default)	If **add**, adds the given permissions to the permissions that already exist. If **remove**, deletes the given permissions from the permissions that already exist. If **set**, adds the given permissions and deletes the permissions that already exist.

You must specify either **id** or **name**, but you do not have to specify both.

See **Global URL parameters** on page 1403 for available URL parameters.

Example request

```
PUT http://locahost/mws/rest/role/Acme-User-Role?change-mode=add?api-version=3
------------------------------------

{
       "permissions":[{"id":"4fa197e68ca30fc605dd1cf0"} ]
}
```

Sample response

If the request was successful, the response body is the modified role as shown in **Get single role**. On failure, the response is an error message.

Deleting roles

The HTTP DELETE method is used to delete **Roles**.

Quick reference

```
DELETE http://localhost:8080/mws/rest/roles/<id>?api-version=3
DELETE http://localhost:8080/mws/rest/roles/<name>?api-version=3
```

Delete single role

URLs and parameters

```
DELETE http://localhost:8080/mws/rest/roles/<id>?api-version=3
DELETE http://localhost:8080/mws/rest/roles/<name>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the Role.
name	Yes	String	--	The name of the Role.

You must specify either **id** or **name**, but you do not have to specify both.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response
------------------------------------

{}
```

Related topics

- **Fields: Roles** on page 2121
- **Resources introduction** on page 1424

Standing reservations

This section describes behavior of the **Standing Reservation** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from MWS.

The **Fields: Standing Reservations** reference section contains the type and description of all fields in the **Standing Reservation** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported methods

Resource	GET	PUT	POST	DELETE
/rest/standing-reservations	Get all standing reservations	--	--	--
/rest/standing-reservations/<id>	Get single standing reservation	--	--	--

This topic contains these sections:

Getting standing reservations

The HTTP GET method is used to retrieve **Standing Reservation** information. Queries for all objects and a single object are available.

Quick reference

```
GET http://localhost:8080/mws/rest/standing-reservations/<id>?api-version=3
```

Get all standing reservations

URLs and parameters

```
GET http://localhost:8080/mws/rest/standing-reservations?api-version=3
```

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
GET http://localhost:8080/mws/rest/standing-reservations?api-version=3&fields=id
------------------------------------

{
  "totalCount": 3,
  "resultCount": 3,
  "results":   [
    {"id": "sr1"},
    {"id": "sr2"},
    {"id": "sr3"}
  ]
}
```

Get single standing reservation

URLs and parameters

```
GET http://localhost:8080/mws/rest/standing-reservations/<id>?api-version=3
```

Parameter	Required	Type	Valid values	Description
id	Yes	String	--	The unique identifier of the object.

See **Global URL parameters** on page 1403 for available URL parameters.

Sample response

```
JSON response
------------------------------------

{
  "access": "DEDICATED",
  "accounts": ["account1"],
  "aclRules": [  {
    "affinity": "POSITIVE",
    "comparator": "EQUAL",
    "type": "USER",
    "value": "adaptive",
  }],
  "chargeAccount": "account2",
  "chargeUser": "user2",
  "classes": ["class1"],
  "clusters": ["cluster1"],
  "comment": "comment",
  "days": ["Monday"],
  "depth": 2,
  "disabled": false,
  "endOffset": 86415,
  "flags": ["ALLOWJOBOVERLAP"],
  "groups": ["group1"],
  "hosts": ["host1"],
  "id": "fast",
  "jobAttributes": ["TEMPLATESAPPLIED"],
  "maxJob": 2,
  "maxTime": 0,
  "messages": ["message1"],
  "nodeFeatures": ["feature1"],
  "os": "Ubuntu 10.04.3",
  "owner":   {
    "name": "root",
    "type": "USER"
  },
  "partition": "ALL",
  "period": "DAY",
  "procLimit":   {
    "qualifier": "<=",
    "value": 5
  },
  "psLimit":   {
    "qualifier": "<=",
    "value": 60
  },
  "qoses": ["qos1"],
  "reservationAccessList": [],
  "reservationGroup": "group2",
  "resources":   {
    "PROCS": -1,
    "tapes": 1
  },
  "rollbackOffset": 43200,
  "startOffset": 347040,
  "taskCount": 0,
  "tasksPerNode": 0,
  "timeLimit": -1,
  "triggers": [],
  "type": "type1",
  "users": ["user1"]
}
```

Related topics

- **Fields: Standing Reservations** on page 2129
- **Resources introduction** on page 1424

Reporting framework

Overview of reporting framework

The reporting framework is a set of tools to make time-based reports from numerical data. The following sections will (1) provide an overview of the framework and the concepts related to it, and (2) work through an example report (CPU Utilization) with details regarding which web services to use and with what data.

The REST API reference is located in the Report resource section (see **Reports** on page 1614).

Concepts

The reporting framework uses 3 core concepts: reports, datapoints, and samples.

- Reports (see **Fields: Reports** on page 2074): A report is a time-based view of numerical data.
- Report Datapoints (see **Fields: Report Datapoints** on page 2072): A datapoint is a consolidated set of data for a certain time period.
- Report Samples (see **Fields: Report Samples** on page 2127): A sample is a snapshot of a certain set of data at a particular point in time.

To illustrate, consider the memory utilization of a virtual machine: at any given point in time, you can get the memory utilization by using your operating system's performance utilities (top for Linux, Task Manager for Windows):

```
2400/12040MB
```

By recording the memory utilization and time constantly for 1 minute, you could gather the following data:

Time	Memory utilization
3:53:55 PM	2400/12040 MB
3:54:13 PM	2410/12040 MB
3:54:27 PM	2406/12040 MB
3:54:39 PM	2402/12040 MB

Time	Memory utilization
3:54:50 PM	2409/12040 MB

Each of the rows in the table above represent a **sample** of data. By averaging the rows we can consolidate them into one or more **datapoints**:

Start time	End time	Memory utilization
3:53:30 PM	3:54:00 PM	2400/12040 MB
3:54:00 PM	3:54:30 PM	2408/12040 MB
3:54:30 PM	3:55:00 PM	2406/12040 MB

Note that each datapoint covers exactly the same amount of time, and averages all samples within that period of time.

A **report**, then, is simply a list of datapoints with some additional configuration information:

Field	Value
Name	Memory Utilization Report
Datapoint Duration	30 seconds
Report Size	3 datapoints

Datapoints:

Start time	End time	Memory utilization
3:53:30 PM	3:54:00 PM	2400/12040 MB
3:54:00 PM	3:54:30 PM	2408/12040 MB
3:54:30 PM	3:55:00 PM	2406/12040 MB

Capabilities

While storing simple information like memory utilization is nice, the reporting framework is built to automatically handle much more complex information.

Consolidating Samples

Samples are JSON documents which are pushed into the report using the Samples API (see **Creating samples** on page 1622). Samples are then stored until the consolidation operation creates a datapoint out of them. The table below shows how different data types are handled in this operation:

Type	Consolidation function handling
Numbers	Numerical data is averaged.
Strings	Strings are aggregated into an array.
Objects	The consolidation function recursively consolidates sub-objects.
Lists	Lists are combined into a single flat list containing all elements.
Mixed	If samples have different types of data for the same field, the values are aggregated into an array.
Null	These values will be ignored unless all values for a sample field are set to null, resulting in a null result.

If the mixed data types contains at least one number, it will be treated as numerical data. The non-numerical data will be ignored and the result will be averaged.

Below is an example of how the consolidation function works:

- Samples:

Time	NumberEx	StringEx	ListEx	MixedEx	MixedNumberEx
3:53:55 PM	2400	"str1"	["elem1"]	"str1"	"str1"
3:54:13 PM	2410	"str2"	["elem2", "elem3"]	["elem1"]	["elem1"]
3:54:27 PM	2405	"str3"	["elem4"]	null	5

- Resulting Datapoint after consolidation:

Time	NumberEx	StringEx	ListEx	MixedEx	MixedNumberEx
3:55:00 PM	2405	["str1", "str2", "str3"]	["elem1", "elem2", "elem3", "elem4"]	["str1", "elem1"]	5

Minimum number of samples

If your dataset is highly variable (i.e. values contained in samples are not very close together), converting a single sample into a datapoint may provide misleading information. It may be better to have a datapoint with an "Unknown" value. This can be accomplished by setting the minimum number of samples for a datapoint in the report.

The `minimumSampleSize` field in the Reports reference section (see **Reports** on page 1614) explains that if the specified size of samples is not met when the consolidation function is performed, the datapoint is considered "null" and no data is available for it. When this occurs, the sample data is discarded and the `data` field of the datapoint is set to "null".

For information on how to set this option, see the REST API Report Resource section (see **Reports** on page 1614).

Report size

Reports have a predetermined number of datapoints, or size, which sets a limit on the amount of data that can be stored. After the report size has been reached, as newly created datapoints are pushed into the report, the oldest datapoints will automatically be deleted. This is to aid in managing the storage capacity of the server hosting MWS.

On report creation, a Mongo collection will be initialized that is the configured report document size multiplied by the report size. Be careful in setting a large report size or report document size as this may quickly allocate the entire disk. See the `reportDocumentSize` and `reportSize` fields in **Fields: Reports** on page 2074 for more information.

Related topics

- **Example report (CPU Utilization)** on page 1646

Example report (CPU Utilization)

To understand how the behavior and usage of the reporting framework, a sample report covering CPU Utilization will be shown in this section. It will not cover how to gather or display data for reports, but will cover some basic operations that are available with Moab Web Services to facilitate reporting.

Creating a report

Before any data is sent to Moab Web Services, a report must first be created. A JSON request body with a HTTP method of POST must be used to do this.

```
POST /rest/reports
------------------------------------

{
  "name":"cpu-util",
  "description":"An example report for cpu utilization",
  "consolidationFunction":"average",
  "datapointDuration":600,
  "reportSize":288
}
```

This will result in a report being created which can then be retrieved by sending a GET request to `/rest/reports/cpu-util`. The `datapointDuration` of `600` signifies that the datapoint consolidation should occur once every 10 minutes, while the `reportSize` (i.e. number of the datapoints) shows that the report will retain up to 2 days worth of the latest datapoints.

```
GET /rest/reports/cpu-util
------------------------------------

{
    "consolidationFunction": "average",
    "datapointDuration": 600,
    "datapoints": [],
    "description": "An example report for cpu utilization",
    "id": "aef6f6a3a0bz7bf6449537c9d",
    "keepSamples": false,
    "minimumSampleSize": 1,
    "name": "cpu-util",
    "reportSize": 288,
    "version": 0
}
```

(Note that an ID has been automatically generated and that no datapoints are associated with the report.)

Adding samples

Until samples are added and associated with the report, datapoint consolidation will generate datapoints with a `data` field equal to `null`. Once samples are added, however, they will be averaged and inserted into the next datapoint.

Create samples for the `cpu-util` by sending a POST request as follows:

```
POST /rest/reports/cpu-util/samples
------------------------------------

[
  {
    "agent": "cpu-monitor",
    "timestamp":"2012-01-01 12:00:00 UTC",
    "data": {
      "minutes1": 0.5,
      "minutes5": 0,
      "minutes15": 0
    }
  },
  {
    "agent": "cpu-monitor",
    "timestamp":"2012-01-01 12:01:00 UTC",
    "data": {
      "minutes1": 1,
      "minutes5": 0.5,
      "minutes15": 0.05
    }
  },
  {
    "agent": "cpu-monitor",
    "timestamp":"2012-01-01 12:02:00 UTC",
    "data": {
      "minutes1": 1,
      "minutes5": 0.5,
      "minutes15": 0.1
    }
  },
  {
    "agent": "cpu-monitor",
    "timestamp":"2012-01-01 12:03:00 UTC",
    "data": {
      "minutes1": 0.75,
      "minutes5": 1,
      "minutes15": 0.25
    }
  },
  {
    "agent": "cpu-monitor",
    "timestamp":"2012-01-01 12:04:00 UTC",
    "data": {
      "minutes1": 0,
      "minutes5": 1,
      "minutes15": 0.85
    }
  }
]
```

This sample data contains average load for the last 1, 5, and 15 minute intervals. The samples were recorded at one-minute intervals starting at noon on January 1st, 2012.

Consolidating data

A consolidation function must run to generate datapoints from the given samples. This scheduled consolidation will occur at intervals of `datapointDuration` seconds. For each field in the `data` object in samples, all values will be averaged. If non-numeric values are included, the following strategies will be followed:

1. All fields which contain a single numeric value in any included sample will be averaged and the non-numeric or null values will be ignored.
2. All fields which contain a list will be consolidated into a single, flat list.
3. All fields which contain only non-numeric or null values will be consolidated into a single, flat list.

If no historical datapoints are provided in the creation of a report as in this example, the next consolidation will be scheduled for the current time plus the `datapointDuration`. In this example, the scheduled consolidation is at 10 minutes from the creation date. If historical datapoints are included in the report creation, the latest datapoint's `endDate` plus the `datapointDuration` will be used as the scheduled time. If this date was in the past, the next scheduled consolidation will occur at the appropriate interval from the last `endDate`.

Retrieving report data

To retrieve the consolidated datapoints, simply perform a GET request on the report once again. Alternatively, the GET for a report's datapoints (see **Get datapoints for single report** on page 1618) may be used.

```
GET /rest/reports/cpu-util
------------------------------------

{
    "consolidationFunction": "average",
    "datapointDuration": 600,
    "datapoints": [
        {
            "firstSampleDate": null,
            "lastSampleDate": null,
            "data": null,
            "startDate": "2012-01-01 11:49:00 UTC",
            "endDate": "2012-01-01 11:59:00 UTC"
        },
        {
            "firstSampleDate": "2012-01-01 12:00:00 UTC",
            "lastSampleDate": "2012-01-01 12:04:00 UTC",
            "data": {
                "minutes1": 0.65,
                "minutes15": 0.25,
                "minutes5": 0.6
            },
            "startDate": "2012-01-01 11:59:00 UTC",
            "endDate": "2012-01-01 12:09:00 UTC"
        }
    ],
    "description": "An example report for cpu utilization",
    "id": "aef6f6a3a0bz7bf6449537c9d",
    "keepSamples": false,
    "minimumSampleSize": 1,
    "name": "cpu-util",
    "reportSize": 288,
    "version": 0
}
```

Note that of the two datapoints above, only the second actually contains data, while the other is set to `null`. Only samples lying within the datapoint's duration, or from the `startDate` to the `endDate`, are included in the consolidation. Therefore the first datapoint, which covered the 10 minute period just before the samples' recorded timestamps, contained no data. The second, which covers the 10 minute period matching that of the samples, contains the averaged sample data. This data could be used to display consolidated report data in a custom interface.

Possible configurations

Configuration options may be changed to affect the process of report generation. These are documented in **Fields: Reports** on page 2074 and **Fields: Report Samples** on page 2127.

Related topics

- **Overview of reporting framework** on page 1643

Plugins

About Moab Web Services plugins

This chapter describes MWS plugins, their use, and their creation in Moab Workload Manager. The sections in this chapter provide you with the following information:

- An introduction to the concept of MWS plugins (see **Plugin introduction** on page 1651).
- A description of the plugin lifecycle (see **Lifecycle states** on page 1653).
- How plugins are driven by events (**Handling events** on page 1689).
- How to expose web services from a plugin (**Exposing web services** on page 1674).
- How plugin utility services may be used (**Utility services** on page 1655).
- How data report collisions between plugins are consolidated (**Data consolidation** on page 1655).
- How calls from Moab are routed to MWS plugins (**Routing** on page 1657).

It contains the following sections:

Related topics

- **Configuring Moab Web Services** on page 1373

Plugin overview

This section provides an overview of the plugin layer in web services. It contains these topics:

Related topics

- **About Moab Web Services plugins** on page 1650

Plugin introduction

Moab Web Services plugins provide a highly extensible interface to interact with Moab, MWS, and external resources. Plugins can perform some of the same functions as Moab resource managers (RMs), while also providing many other features not available to RMs. This section will discuss the main features of plugins, some basic terminology, and how MWS plugins can interact with Moab.

Features

Plugins can:

- Be created, modified, and deleted without restarting Moab Workload Manageror MWS.
- Be defined in Groovy and uploaded to MWS without restarting.
- Have individual data storage space and configuration.
- Access MWS configuration and RESTful web services.
- Log to a standard location configured in MWS.
- Be polled at a regular interval (configured on a per-plugin basis).
- Be informed of important system events.
- Be individually stopped, started, paused, and resumed.
- Expose secured and unsecured custom web services for external use.
- Be manipulated via a full RESTful API (for more information, see **Resources introduction** on page 1424).
- Be manipulated via a full user interface in a web browser.

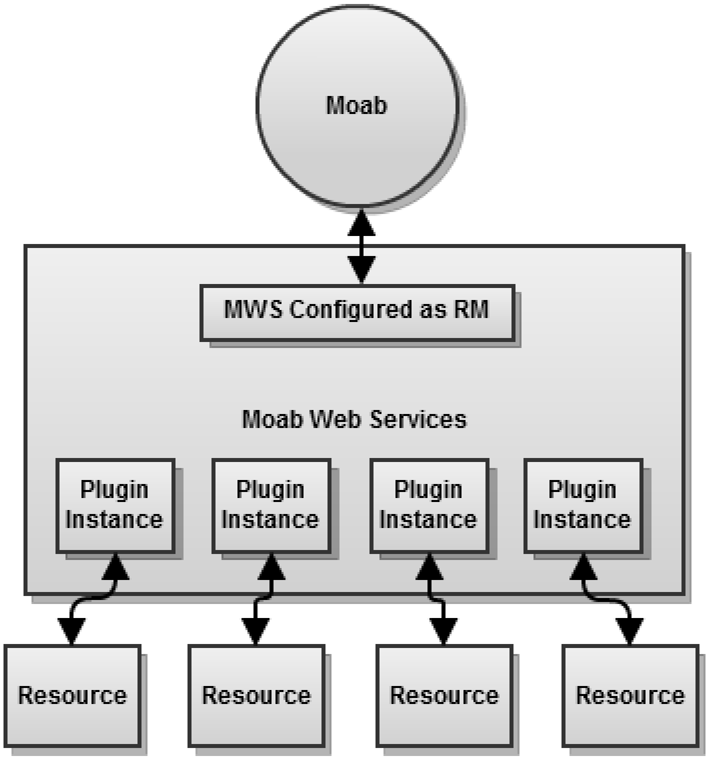
Terminology

There are two distinct terms in the plugin layer: **plugin types** and **plugins (instances)**.

<table>
<tr><th>Term</th><th>Description</th></tr>
<tr><td>plugin types</td><td>Plugin types can be considered plugin templates with built-in logic. In object-oriented programming languages, this relates to the concept of a class. They possess certain abilities, or methods, that can be called by Moab Web Services to query or update information about certain resources. They also can define methods which will be exposed to external clients as web services. They do not contain any configuration or current data, but they are often tied to a type of component, such as components that communicate with Moab's WIKI Protocol, or those that are built on a certain product.
They can define several types of methods:
Instance methods that return information about the current plugin, such as <code>getState</code>. (While these are defined in the plugin type, the plugin type itself does not have a state.)
The poll event method that is called at a configured interval.
Lifecycle event methods of plugins created from the plugin type, such as <code>beforeStart</code> and <code>afterStart</code>.
RM event methods that are called by Moab when certain events occur.
Web service methods that expose custom functionality as public web services.
Some examples of plugin types include the Native and vCenter plugin types.</td></tr>
<tr><td>plugins (instances)</td><td>Plugins (also called plugin instances) are created from plugin types. They contain current data or configuration and use the plugin type methods to interact with resources.</td></tr>
</table>

Interactions with Moab as a resource manager

The plugin layer in MWS is integrated with Moab Workload Manager via the Native Resource Manager (RM) interface. When utilizing plugins, MWS is configured as a RM in Moab, as explained in the next section. Events from Moab are pushed through the RM interface to MWS, which is then pushed to each plugin in turn. The relationship between MWS, Moab, and plugins is shown in the following image:



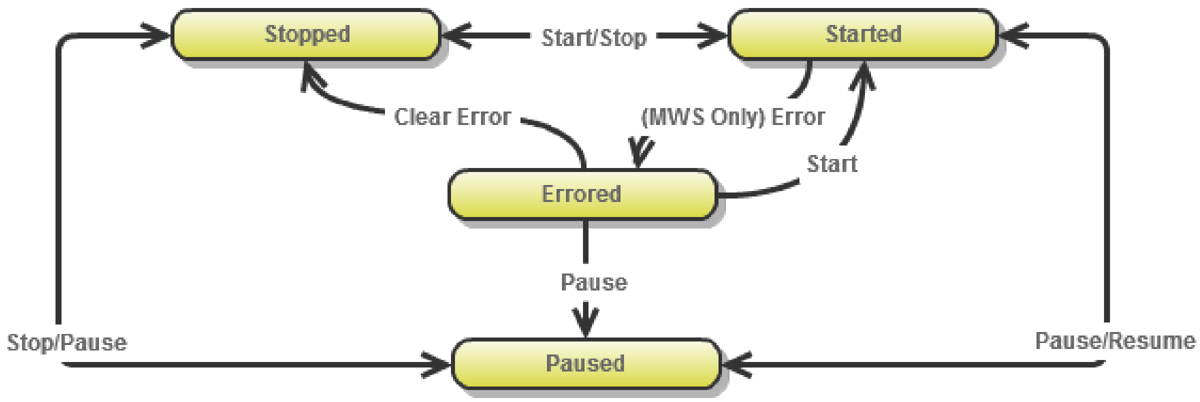
For more information, see **Data consolidation** on page 1655 and **Reporting state data** on page 1676.

Related topics

- **About Moab Web Services plugins** on page 1650

Lifecycle states

During the course of a plugin's use, the state of the plugin may change many times. Plugins have four possible states: **Stopped**, **Started**, **Paused**, and **Errored**. For the descriptions of each state, see the **Fields: Plugins** reference section. The flow of a plugin through the states is shown in the following image:



You can see **Handling events** on page 1689 for information about the events that occur during lifecycle state changes.

Related topics

- **Plugin introduction** on page 1651

Events

Plugins use an event-based model, meaning that methods are called on the plugin when certain criteria are met or situations arise. Events currently exist for polling, lifecycle state changes, and RM events from Moab. For more information, see **Handling events** on page 1689.

Related topics

- **Handling events** on page 1689
- **Plugin introduction** on page 1651

Custom web services

Although the events interface typically serves most cases, there are some instances where an event is not supported that is desired. This is especially true when an external resource is the source of the event. To address these issues, plugins can expose custom web services to external resources. These web services may be named freely and do anything they wish within the plugin framework.

For example, suppose a resource needs to notify a plugin that provisioning of a virtual machine has been completed. Instead of having the plugin poll the resource to verify that the provisioning was finished, the plugin could expose a custom web service to handle notification from the resource itself.

```
Sample custom web service
-----------------------------------


def vmProvisionFinished(Map params) {
        // Handle event
        return [messages:["Event successfully processed"]]
}
```

Additionally, plugin types may define web services which are unsecured, meaning that a user or application account is not required to access it. A full explanation of the syntax and creation of custom secured and unsecured web services may be seen on **Exposing web services** on page 1674.

For information how resources can access plugin web services, see **Accessing Plugin Web Services** on page 1583.

Related topics

- **Plugin introduction** on page 1651

Utility services

Several features of plugins are only available by utilizing bundled services. These include:

- Accessing the individual datastore (see **Individual datastore** on page 1672).
- Reporting state data to Moab through the Resource Manager interface (see **Reporting state data** on page 1676).
- Manipulating other plugins and controlling their lifecycle (see **Controlling lifecycle** on page 1679).
- Accessing REST resources from Moab Web Services (**Accessing MWS REST resources** on page 1680).

It may also be necessary or desired to create additional utility services when creating new plugin types. The easiest way to do this is to create a utility service which is called by convention a translator (see **Using translators** on page 1694), because it can typically "translate" from a specific resource or API to data which can be used by the plugin type.

Finally, custom components (see **Registering custom components** on page 1695) may be used to fulfill use cases not covered by bundled services or custom translators.

Related topics

- **Plugin introduction** on page 1651

Data consolidation

At times, plugins can report differing or even contradictory data for nodes, virtual machines, and jobs. This is called a data "collision". The act of resolving these collisions is called "Consolidation." Plugins also have the concept of "precedence," where the plugins with the lowest precedence value are considered more authoritative than the greater precedence values plugins. For example, a plugin with a precedence value of 1 has a higher precedence and is considered more authoritative than a plugin with a precedence value of 5. If no precedence is provided when creating plugins, the plugin is automatically assigned to the lowest precedence, or 1 greater than the highest precedence value. The precedence value may not be less than 1.

When data from one plugin "collides" with another, the data from the highest precedence plugin will be considered the authoritative source for information. If multiple sets of data (reports) are provided by the same plugin, the latest set of data will take precedence. Additionally, MWS supports the concept of treating node and virtual machine data with state information **optimistically**, **pessimistically**, or **neither**. This is configured using the `plugins.stateConsolidationPolicy` configuration property in the MWS configuration file. If this property is set to `optimistic` and *any* plugin reports the state for a node or VM as "Up," the consolidated state will be "Up." Inversely, if the property is set to `pessimistic` and *any* plugin reports the state as "Down," the consolidated state will be "Down." If it is set to `null` (neither), consolidation will occur for the state field just as with any other field, with higher precedence and later reports being considered authoritative.

> When MWS is upgraded to a version that supports plugin precedence from an older version, existing plugins will not have the precedence field set. The administrator should assign precedence to each plugin manually through the API (see **Modifying plugins** on page 1581) or through the user interface (see **Modifying a plugin** on page 1722) to ensure that the consolidation will occur as expected. By default, data from a plugin without a precedence defaults to a precedence of 1, or the highest precedence.

Consolidation examples

Suppose two plugins exist, `pluginA` and `pluginB`. Plugin "A" has a precedence of 1, and plugin "B" has a precedence of 2, meaning that plugin "A" is more authoritative. These plugins both report data for a node with an ID of `node1`. However, each reports a different node power state. Plugin "A" reports the power as `ON`, while plugin "B" reports the power as `OFF`. The data collision that occurs due to these two contradictory reports is resolved by the precedence of the plugins. Since plugin A has a higher precedence (lower number), it is considered authoritative and the node will be reported as `ON`.

Now suppose that the plugins also report differing node state for `node1`. In this case, the node state would depend on the `plugins.stateConsolidationPolicy` property. The different combinations of report values compared to the state consolidation policy and the final reported state are shown in the table below.

Plugin "A" node state	Plugin "B" node state	State consolidation policy	Consolidated node state
ON	OFF	null (neither)	ON
OFF	ON	null (neither)	OFF
ON	OFF	optimistic	ON
OFF	ON	optimistic	ON

Plugin "A" node state	Plugin "B" node state	State consolidation policy	Consolidated node state
ON	OFF	pessimistic	OFF
OFF	ON	pessimistic	OFF

In general, it is recommended that no two plugins report the same resource or that they report different properties of the same resource. For example, if plugin "A" only modified the power state and plugin "B" only modified the available disk resource, these two plugins would work in harmony to provide a consistent view of the node resource.

For more information, see **Reporting state data** on page 1676 and **Resource manager queries** on page 1713.

Related topics

- **Plugin introduction** on page 1651

Routing

Plugin routing is currently in *Beta*. Interfaces may change significantly in future releases.

Because Moab Web Services is configured as a Resource Manager (RM) in Moab Workload Manager, events are sometimes triggered by Moab through the RM interface. These actions could be migrating a virtual machine, starting a job, submitting a job, modifying a node, and so forth. The decisions regarding which plugins are affected and notified is termed **routing**.

Currently all plugins receive all commands from Moab. This means that each plugin will receive the command to start a job if sent from Moab, even if that plugin does not handle the job. This means that plugins must ensure they handle actions or commands only for resources which they report or handle.

Related topics

- **Plugin introduction** on page 1651

Plugin developer's guide

Plugin types comprise the methods by which Moab may communicate with resource managers or other external components. They define all operations that can be performed for a "type" or "class" of plugins, hence the name "plugin type."

Several plugin types are provided with Moab Web Services, but it is easy to create additional plugin types and add their functionality to web services. This involves using Groovy, which is based on the Java programming language. This section describes the general guidelines and specifics of implementing new plugin types.

API classes and interfaces

There are several packages and classes available to assist in creating plugin types. These can all be found in the API documentation.

This section contains these topics:

Related topics

- **About Moab Web Services plugins** on page 1650

Requirements

This section discusses the requirements to create a basic functional plugin. The `com.adaptc.mws.plugins` package contains the abstract class AbstractPlugin that should form the basis of any new plugin type. However, this class need not be extended to create a functional plugin type. Only two requirements must be fulfilled for this:

1. The class name must end in `Plugin`.
2. There must exist `id` field getter and setter methods:

```
* public String getId();
* public void setId(String id);
```

The `id` field may be stored in whichever way desired as long as the getter and setter are available as shown above, but will most likely be implemented as follows:

```
class BasicPlugin {
       String id
}
```

In this case, `String id` will be expanded by the Groovy compiler to the full getter and setter method definitions given above. In other words, no explicit method definitions are actually needed. Note that the `BasicPlugin` shown above is able to be uploaded as a plugin type to MWS, but does not actually do anything.

It must also be noted that the `AbstractPlugin` class already implements an `id` field. Therefore, a plugin type that extends this class does not need to define the field as shown in the following example.

```
import com.adaptc.mws.plugins.AbstractPlugin

class BasicPlugin extends AbstractPlugin {
       // No ID field is needed since it exists in AbstractPlugin
}
```

Related topics

- **Plugin developer's guide** on page 1657

Dynamic methods

> These methods are currently in *Beta*. Interfaces may change significantly in future releases.

Several methods are dynamically inserted onto each plugin. These methods do not need to be included in the plugin class, and will be overwritten if included. Additionally, a logger is inserted into each plugin as discussed in the next section. The inserted methods are shown below (full definitions can be found in AbstractPlugin and AbstractPluginInfo):

- `public void start() throws PluginStartException;` (Equivalent to the start method in the **Plugin control service** on page 1728.)
- `public void stop() throws PluginStopException;` (Equivalent to the stop method in the **Plugin control service** on page 1728.)
- `public Log getLog();` (See **Logging** on page 1660.)
- `public ConfigObject getAppConfig();` (See **Configuration** on page 1663.)
- `public String message(Map parameters);` (See **i18n messaging** on page 1661.)
- `public String getPluginType();`
- `public PluginState getState();`
- `public Integer getPollInterval();`
- `public Boolean getAutoStart();`
- `public Map<String, Object> getConfig();` (See **Configuration** on page 1663.)

Many of these methods are provided for convenience and are discussed in the linked pages or the following sections.

Related topics

- **Plugin developer's guide** on page 1657

Logging

Logging in plugin types uses the Apache Commons Logging and log4j libraries. Each plugin is injected with a method called `getLog` which can be used to access the configured logger. It returns an instance of org.apache.commons.logging.Log. Examples of using the logger are shown below.

The logger may used to register messages to the MWS log at several levels (in order of severity):

1. trace
2. debug
3. info
4. warn
5. error
6. fatal

Each of these levels is available as a method on the logger, for example:

```
public void poll() {
        getLog().debug("getLog() is equivalent to just using 'log' in Groovy")
        log.debug("This is a debug message and is used for debugging purposes only")
        log.info("This is a informational message")
        log.warn("This is a warning")
        log.error("This is an error message")
}
```

Logger name

Each logger in the MWS logging configuration has a name. In the case of plugins, it is comprised of the full class name, including the package, prepended by "`plugins.`". For example, a plugin class of "example.LoggingPlugin" will have access to a logger configured as `plugins.example.LoggingPlugin`.

Logging configuration

The logging configuration is done through the MWS configuration file. For more information on configuring loggers, see **Configuring Moab Web Services** on page 1373. A good configuration for developing plugin types may be to add "plugins" at the debug level. Be sure to set the log level threshold down for the desired appender.

```
log4j = {
  …
  // Appender configuration
  ...
  debug "plugins"
}
```

Related topics

- **Plugin developer's guide** on page 1657

i18n messaging

Plugins, translators, and custom components all have access to i18n messages. Utilizing messages requires the two following steps:

1. Including a file (or multiple files) that ends in "`messages.properties`" in the plugin JAR file.
2. Using the `message` method on a plugin type, translator, or custom component.

Including messages in plugin JAR file

Messages are defined using property files. These may be named anything as long as they end with "`messages.properties`" and must be placed at the root or top level of the plugin JAR file. If they are present, they will be loaded automatically. Multiple property files may be used within a single plugin JAR file.

Each property file consists of an arbitrary amount of lines that define a message property (also called a code) with letters, numbers, and periods, associated with a human-readable message that can span multiple lines, have quotes, or contain arguments. These are demonstrated in the following example.

```
first.message.code=This is the first message
second.message=This message can span multiple lines, \\
  and will not show the linebreaks when retrieved
message.with.arguments=This message has arguments: first - {0}, second - {1}, third - {2}, etc.
message.with.quotes=This message uses single quotes around ''this phrase''.
```

It is recommended to namespace the messages by using the property definitions and multiple property files if necessary. For example, suppose a plugin JAR existed which actually contained two plugin types: `Message1Plugin` and `Message2Plugin`. The first suggestion is to namespace the messages for each plugin by the property definition, such as the following:

```
message1Plugin.first.message=This is a message for Message1Plugin
message2Plugin.first.message=This is a message for Message2Plugin
```

These messages could be stored in a file named "`messages.properties`" in the root of the plugin JAR file. If there are many messages contained for each plugin type, it may be necessary to split each plugin type's messages into a separate file, such as "`message1-messages.properties`" and "`message2-messages.properties`". Note that it is essential that each property file ends with "`messages.properties`" so that it is registered correctly.

It is important that no two message codes are identical within a single plugin JAR file, even if they are defined in separate property files. If this is done, a conflict will exist with the messages and behavior is undefined.

Using the message method

Each plugin, translator, and custom component is injected with a method named `message`. This method takes a Map as its parameter, which can contain one or several of the following properties:

Parameter	Type	Description
code	String	The message property definition (everything before the equals sign in the property file for a single message), for example, `first.message.code`.
args	List<Object>	A list of arguments to insert into the message.
default	String	A default message to be used when the message code cannot be resolved.
error	org.springframework.context.MessageSourceResolvable	An object that represents a hierarchy of message codes. This is typically used to display errors.

The most utilized parameters are `code` and `args`, as these combined provide great flexibility in generating messages. If a message cannot be resolved, or in other words the message definition does not exist, the code will simply be returned as the resolved message. Below are several examples of messages resolved using the property files given above. While these are contained in the polling method, the `message` may be used anywhere within a plugin type.

```
package example
import com.adaptc.mws.plugins.AbstractPlugin

class MessagingPlugin extends AbstractPlugin {
	def poll() {
		assert message(code:"first.message.code")=="This is the first message"
		assert message(code:"message.with.arguments", args:[
				"1st", 2, true
			])=="This message has arguments: first - 1st, second - 2, third - true, etc."
		assert message(code:"message.with.quotes")=="This message uses single quotes around 
'this phrase'."
		assert message(code:"invalid.message.code")=="invalid.message.code"
	}
}
```

Related topics

- **Plugin developer's guide** on page 1657

Configuration

Plugin types can access two different kinds of configuration: an individual plugin's configuration, and the global MWS application configuration.

Individual plugin configuration

The individual plugin configuration is separate for each instance of a plugin. This may be used to store current configuration information such as access information for linked resources. It should not be used to store cached information or non-configuration related data. The individual datastore should be used instead for these cases (for more information, see **Individual datastore** on page 1672).

It is accessed by using the `getConfig` method discussed in **Dynamic methods** on page 1659.

```
public void poll() {
  def configFromMethod = getConfig()
  // OR an even simpler method…
  def configFromMethod = config
}
```

A common case is to retrieve the configuration in the `configure` method, verify that it matches predetermined criteria, and utilize it perform initial setup of the plugin (e.g. initialize libraries needed to communicate with external resources). For example, to verify that the configuration contains the keys "username" and "password," the following code may be used.

```
public void configure() throws InvalidPluginConfigurationException {
  def myConfig = config
  // This checks to make sure the key exists in the configuration Map and that the
value is not empty or null
  if (!myConfig.containsKey("username") || !myConfig.username)
       throw new InvalidPluginConfigurationException("The username configuration parameter
must be provided")
  if (!myConfig.containsKey("password") || !myConfig.password)
    throw new InvalidPluginConfigurationException("The password configuration
parameter must be provided")
}
```

Access MWS configuration

The MWS application configuration can also be accessed in plugin types. This configuration is global for the entire application and can be modified by the administrator as shown in **Configuring Moab Web Services** on page 1373.

It is accessed by using the `getAppConfig` method discussed in **Dynamic methods** on page 1659. This is demonstrated below:

```
public void poll() {
  // Retrieve the current MWS_HOME location
  def mwsHome = appConfig.mws.home.location
  // OR an even simpler method…
  def mwsHome = getAppConfig().mws.home.location
}
```

Any of the properties shown in the **Configuration** reference may be accessed. Custom properties may also be registered and accessed:

```
mws-config.groovy
------------------------------------

plugins.custom.property = "This is my custom property"
```

```
CustomAppPropertyPlugin
------------------------------------

public void poll() {
  assert appConfig.plugins.custom.property=="This is my custom property"
}
```

Related topics

- **Plugin developer's guide** on page 1657

Configuration constraints

Plugin types can optionally define validation constraints for the polling interval and plugin configuration. These parameters are then checked against the defined constraints during the creation of a new plugin. If the validation fails, meaning the configuration provided does not pass the constraints defined by the plugin type, the plugin will fail to be created with error messages based on the parameters and constraints defined.

Defining constraints

To define constraints for a plugin type and therefore for all plugins created using it, use the following syntax:

```
import com.adaptc.mws.plugins.*
class ConstrainedPlugin extends AbstractPlugin {
       static constraints = {
               // Set plugin's default polling interval
               pollInterval defaultValue:60
               // The "myParam" configuration parameter is automatically required and cannot be
blank
               myParam blank:false
               // The "myEnum" configuration parameter is not required and must set to one of the
values in the list
               myEnum required:false, inList:["val1", "val2", "val3]
               // Insert additional constraints here…
       }
}
```

In the table below, all available constraints are shown, as well as the expected value type, an example, the default message code, and the message suffix. The message columns are described in greater detail in the **Messaging** section below.

Constraint	Default value	Type	Example value	Default message code	Message suffix	Description
blank	--	Boolean	`true`	default.blank.message	blank	If false, the parameter (if present) cannot be a blank string.
creditCard	--	Boolean	`true`	default.invalid.creditCard.message	creditCard.invalid	If **true**, uses `org.apache.commons.validator.CreditCardValidator` to determine if the parameter (if present) is a valid credit card number.
defaultValue	--	Object or Closure	`60`	--	--	If the parameter is not present, it will be set to this default value. Does not return any error messages. See **Default value** below for more information.
email	--	Boolean	`true`	default.invalid.email.message	email.invalid	If **true**, the parameter (if present) must be a valid email address.
inList	--	List	`["first", "second"]`	default.not.inlist.message	not.inList	The parameter (if present) must be set to one of the values specified.
matches	--	String	`"[a-z][A-Z]+"`	default.doesnt.match.message	matches.invalid	The parameter (if present) must match the specified regular expression.

Constraint	Default value	Type	Example value	Default message code	Message suffix	Description
max	--	Integer	`10`	default.invalid.max.message	max.exceeded	The parameter (if present) must not be greater than the defined value.
***maxSize**	--	Integer	`10`	default.invalid.max.size.message	maxSize.exceeded	The parameter's (if present) size must not be greater than the defined value.
min	--	Integer	`1`	default.invalid.min.message	min.notmet	The parameter (if present) must not be less than the defined value.
***minSize**	--	Integer	`1`	default.invalid.min.size.message	minSize.notmet	The parameter's (if present) size must not be less than the defined value.
notEqual	--	Object	`"Invalid Value"`	default.not.equal.message	notEqual	The parameter (if present) must *not* be set to the defined value.
nullable	true	Boolean	`false`	default.null.message	nullable	If **true**, the parameter (if present) must be non-null value. See **required** for how to enforce the parameter to be present.
password	--	Boolean	`true`	--	--	If **true**, the parameter (if present) is hidden from the user both on input and display when managing plugin configuration. It is not, however, hidden in the REST API. Does not return any error messages.

Constraint	Default value	Type	Example value	Default message code	Message suffix	Description
range	--	Range	`1..10`	default.invalid.range.message	range.toosmall/range.toobig	Uses a groovy range to validate that the value is within a specified range.
required	true	Boolean	`false`	default.required.message	required	If **true**, the parameter must be present and non-null for the plugin to be created successfully. Implies the `nullable:false` constraint.
scale	--	Integer	`2`	--	--	Only valid for Double parameters. Rounds the parameter (if present) to the specified number of digits. Does not return any error messages.
***size**	--	Range	`2`	default.invalid.size.message	size.toosmall/size.toobig	Uses a groovy range to restrict the size of a collection, string, or a number.
***type**	--	Class	`Integer.class`	typeMismatch	typeMismatch	See **Type inferencing and conversion** below.
url	--	Boolean	`true`	default.invalid.url.message	url.invalid	If **true**, uses `org.apache.commons.validator.UrlValidator` to determine if the parameter (if present) is a valid URL. Does not support `exec` or `file` scheme URLs.
scriptableUrl	--	Boolean	`true`	default.invalid.scriptable.url.message	scriptableUrl.invalid	Identical to the **url** validator, but adds support for `exec` and `file` scheme URLs.

Constraint	Default value	Type	Example value	Default message code	Message suffix	Description
validator	--	Closure	(See **Custom validator**)	default.invalid.validator.message	validator.error	See **Custom validator** below.

* The user interface (see **Plugin management** on page 1720) does not support parameters whose type is a subclass of Collection (a List, for example). Such parameters are therefore not recommended.

> The polling interval constraints must always apply to Integer types. If this specification is violated, the plugin type cannot be added or updated.

Messaging

When defined constraints are violated for a plugin, error messages are retrieved based on the configuration parameters and the applied constraints using i18n Messaging codes (see **i18n messaging** on page 1661). First, the most specific error message will be attempted to be resolved from a message code generated from the plugin type name, the configuration parameter, and the constraint. This code takes the format of `pluginTypeName.parameterName.suffix` where the plugin type's name has a lowercase first letter and the suffix is shown in the table above. If this message code is not defined, the default message code (as shown in the table above) will be used.

For example, if the **url** constraint validation failed for the "ExamplePlugin" plugin type's "endpoint" configuration parameter, the following message codes would be resolved in order:

- `examplePlugin.endpoint.url.invalid`
- `default.invalid.url.message`

> Plugin types that have two or more uppercase letters at the start of the name will not be converted to have a lowercase first letter for error message codes. In other words, for the example just given using "VCenterPlugin" instead of "ExamplePlugin", the following message codes would be resolved in order:
>
> `VCenterPlugin.endpoint.url.invalid`
>
> `default.invalid.url.message`

Default messages

Default messages may be contained in any `messages.properties` file included in the plugin JAR file as explained in i18n Messaging (see **i18n messaging** on page 1661). Arguments for each constraint vary, but they always include these argument indices:

- `{0}`: The configuration parameter name (for example, `endpoint`).
- `{1}`: The plugin type class name (for example, `my.package.ExamplePlugin`).
- `{2}`: The value of the configuration parameter.

If default messages are not defined in the plugin project, the following messages will be used:

```
default.doesnt.match.message=The ''{0}'' configuration parameter value ({2}) does not
match the required pattern ''{3}''
default.invalid.url.message=The ''{0}'' configuration parameter value ({2}) is not a
valid URL
default.invalid.scriptable.url.message=The ''{0}'' configuration parameter value ({2})
is not a valid scriptable URL
default.invalid.creditCard.message=The ''{0}'' configuration parameter value ({2}) is
not a valid credit card number
default.invalid.email.message=The ''{0}'' configuration parameter value ({2}) is not a
valid e-mail address
default.invalid.range.message=The ''{0}'' configuration parameter value ({2}) does not
fall within the valid range from {3} to {4}
default.invalid.size.message=The ''{0}'' configuration parameter value ({2}) does not
fall within the valid size range from {3} to {4}
default.invalid.max.message=The ''{0}'' configuration parameter value ({2}) is greater
than the maximum value of {3}
default.invalid.min.message=The ''{0}'' configuration parameter value ({2}) is less
than the minimum value of {3}
default.invalid.max.size.message=The ''{0}'' configuration parameter value ({2})
exceeds the maximum size of {3}
default.invalid.min.size.message=The ''{0}'' configuration parameter value ({2}) is
less than the minimum size of {3}
default.invalid.validator.message=The ''{0}'' configuration parameter value ({2}) does
not pass custom validation
default.not.inlist.message=The ''{0}'' configuration parameter value ({2}) is not
contained within the list [{3}]
default.blank.message=The ''{0}'' configuration parameter cannot be blank
default.not.equal.message=The ''{0}'' configuration parameter value ({2}) cannot be
equal to ''{3}''
default.null.message=The ''{0}'' configuration parameter cannot be null
default.required.message=The ''{0}'' configuration parameter is required and cannot be
null
typeMismatch=The ''{0}'' configuration parameter value ({2}) does not match the
required type ''{3}''
```

Labels and help messages

Message codes may also be provided for configuration parameters to aid the admin user with human readable property labels and help messages. Similar to the validation error message codes, labels and help message codes may be defined using the `pluginTypeName.parameterName.label` and `pluginTypeName.parameterName.help` message codes. These values are used only in plugin type management (see **Plugin type management** on page 1715) and are not exposed through the REST API.

Type inferencing and conversion

Due to the dynamic nature of configuration parameters, the expected type or class of values for each parameter are inferred from constraints. The following rules govern how type is inferred, in priority order:

- If the ***type** constraint is applied to a parameter, the constraint value will be used as the expected type.

> Only the `String`, `Date`, `Double`, `Integer`, and `Boolean` classes are supported for the ***type** constraint. If `Float` or `Long` is desired, use `Double` and `Integer` respectively as the type.

- If the **inList** or **range** constraints are applied to a parameter, the class of the first element in the constraint value array is used as the expected type.
- If the ***minSize** or ***maxSize** constraints are applied to a parameter, `java.lang.Collection` is used as the expected type.
- If the **max**, **min**, or **notEqual** constraints are applied to a parameter, the class of the constraint value is used as the expected type.
- If none of the above apply, `java.lang.String` is used as the expected type.

If the configuration parameter values can be converted to the expected types, this will occur automatically. Otherwise, the ***type** constraint is violated and the applicable error messages will be generated.

Custom validator

In cases where the built-in constraints prove inadequate for validation, custom validators may be used. The **validator** constraint expects a Groovy Closure parameter which has one or (optionally) two arguments: the value of the configuration parameter and the plugin object. With these parameters, complex validation logic may be defined. Additionally, custom message codes and arguments may be defined by validator constraints and these will be used in generating error messages when validation fails.

For example, suppose that the parameter "user" cannot be set to the same value as parameter "creator." Additionally, the "creator" parameter must not be equal to either "bob" or "joe." The existing constraints are inadequate to fulfill this use case, but the following code using validators would perform exactly as expected:

```
import com.adaptc.mws.plugins.*
class ConstrainedPlugin extends AbstractPlugin {
	static constraints = {
		user validator:{ val, obj ->
			if (val==obj.config.creator)
				return "invalid.equal.to.creator"
		}
		creator validator:{ val ->
			if ("val"=="joe")
				return ["invalid.equal", "joe"]
			if (val=="bob")
				return ["invalid.equal", "bob"]
		}
	}
}
```

In the examples above, the message codes and output on validation failure is shown below:

```
Message codes
------------------------------------

constrainedPlugin.user.invalid.equal.to.creator=The user configuration parameter value
({2}) must not be equal to the creator parameter.
constrainedPlugin.creator.invalid.equal=The creator configuration parameter must not
be equal to {3}.
```

```
Output error messages
------------------------------------

For user = "jill", creator = "jill"
"The user configuration parameter value (jill) must not be equal to the creator
parameter."
For user = "jill", creator = "bob"
"The creator configuration parameter must not be equal to bob."

For user = "jill", creator = "joe"
"The creator configuration parameter must not be equal to joe."
```

The validator Closure may return:

- Nothing (`null`) or `true` if the validation succeeded without errors.
- `false` if a validation error occurred (in this case the default validator message suffix would be used).
- A string which will be used as the message code suffix in the `pluginTypeName.propertyName.suffix` format.
- A list with the first element being the message code suffix, and all other elements being arguments for the message indexed starting at 3 (as shown in the example above).

All validator constraints automatically have the `appConfig` property available, which contains the application configuration as discussed in the Configuration section (see **Configuration** on page 1663). The `suite` property contains the value of the configured MWS suite. Additionally, services may be retrieved as explained in the next section.

Retrieving services

At times it may be necessary to use Bundled Services in custom validators. A method named getService which takes a single string parameter of the name of the service (as used during injection) is provided to be used in these cases. For example, if a plugin needs a valid server certificate file, the SSL Service may be used as follows:

```
import com.adaptc.mws.plugins.*
class ConstrainedPlugin extends AbstractPlugin {
	static constraints = {
		certificateFile validator:{ val ->
			ISslService sslService = getService("sslService")
			try {
				sslService.getSocketFactory(val)
			} catch(Exception e) {
				// Certificate file is invalid, return an error
				return ["invalid", e.message]
			}
		}
	}
}
```

The `getService` method does not work with translators, custom components, RM services, or the **Individual datastore** on page 1672.

Default value

The default value for a configuration parameter might depend on the MWS configuration or other properties. Therefore, the **defaultValue** constraint can be set to a closure. The `defaultValue` closure does not take any parameters and must return the object to be used as the default value.

For example, if the default value of a parameter must be true if and only if MWS is configured for the Cloud suite, then the following constraints would satisfy these conditions:

```
import com.adaptc.mws.plugins.*
class ConstrainedPlugin extends AbstractPlugin {
	static constraints = {
		myParameter required: true, type: Boolean, defaultValue: {
			return suite == Suite.CLOUD
		}
	}
}
```

As with **validator** closures, `defaultValue` closures have access to `appConfig`, `suite`, and `getService`.

Related topics

- **Plugin developer's guide** on page 1657

Individual datastore

Each plugin has access to an individual, persistent datastore which may be used for a variety of reasons. The datastore is not designed to store Moab data such as nodes, jobs, or virtual machines, but custom, arbitrary data pertinent only to the individual plugin. This may include storing objects in a persistent cache, state information for currently running processes, or any other arbitrary data. The individual datastore has the following properties:

- Data is persisted to the Mongo database and will be available even if the plugin or MWS is restarted.

- The data must be stored in groups of data called **collections**. These correspond directly to MongoDB collections.
- Each plugin may have an arbitrary number of collections.
- Collections are guaranteed not to collide if there are identically named collections between two plugin types or even two plugin instances.
- Each collection contains multiple objects or **entries**. These correspond directly to MongoDB documents.
- The values of entries may be any object which can be serialized to MongoDB: simple types (int or Integer), Maps, and Lists.
- A collection is automatically created whenever an entry is added to it, it does not need to be specifically initialized.

To utilize the datastore, the **Plugin datastore service** must be used. Operations are provided to add, query, and remove data from each collection.

> Simple key/value storage is not currently provided with the datastore. It may easily be done, however, by storing data in the format of `{name:"key", value:"value"}` and then retrieving this entry later by querying on name equals "key."

Example

The example below demonstrates two web services (see **Exposing web services** on page 1674). The first adds multiple entries containing various types of data to an arbitrarily named collection. The second retrieves the data and returns it to the user.

```
package example
import com.adaptc.mws.plugins.*

class DatastorePlugin extends AbstractPlugin {
	IPluginDatastoreService pluginDatastoreService

	def storeData(Map params) {
		def collectionName = params.collectionName
		def data = [[boolVal:true], [stringVal:"String"], [intVal:1], [nullVal:null]]
		if (pluginDatastoreService.addData(collectionName, data))
			log.info("Data successfully added")
		else
			log.info("There was an error adding the data")
		return [success:true]
	}

	def retrieveData(Map params) {
		def collectionName = params.collectionName
		return pluginDatastoreService.getCollection(collectionName)
	}
}
```

Related topics

- **Plugin developer's guide** on page 1657

Exposing web services

Any number of methods may be exposed as public, custom web services by satisfying several criteria:

- The method must declare that it returns `Object` or `def`.
- The method must define a single argument of type `Map`.
- The method must actually return a `List` or `Map`.
- The method must not be declared as private or protected; only public or unscoped methods will be recognized as web services.

Parameters and request body

The `Map` argument will contain all parameters passed into the web service by the client. See **Accessing Plugin Web Services** on page 1583 for additional details.

Parameters may be passed into the web service call as normal URL parameters such as `?param=value&param2=value2`, as key-value pairs in the POST body of a request, or as JSON in the body.

For the first two cases, the parameters will be available on the Map argument passed into the web service call as key value pairs matching those of the request. Note that in these cases all keys and values will be interpreted as strings. However, the parameters object has several helper methods to convert from Strings to simple types, such as Booleans, integers, doubles, floats, and lists. If the value is not a valid simple type, null is returned.

Finally, note that the client may optionally include an `objectId` as the last part of the URL. When this is done, the **id** field will be set to this value in the `Map` argument to the web service.

```
GET <webServiceUrl>?key=value&key2=true&key3=5&list=1&list=2

def serviceMethod(Map params) {
        assert params.key=="value"
        assert params.key2=="true"
        assert params.bool('key2')==true
        assert params.key3=="5"
        assert params.int('key3')==5
        assert params.list('list')==[1, 2]

        // Null is returned if the conversion is invalid
        assert params.int('key')==null
}
```

When the body possesses JSON, the parsed JSON object or array will be available within a parameter called **body** in the `Map` argument. In this scenario, the types of the values are preserved by the JSON format.

```
POST <webServiceUrl> with JSON body of
{"key":"value","key2":true,"key3":5}

def serviceMethod(Map params) {
        assert params.body.key=="value"
        assert params.body.key2==true
        assert params.body.key3==5
}
```

Unsecured web services

There are times when it is desirable to create a plugin with a publicly available web service that does not require a valid application account in order to access it (for details, see **Access control** on page 1398). In these cases, the Unsecured annotation may be used on the plugin web service method. No authentication will be performed on Unsecured web services. An example of using the annotation is given below.

```
Sample unsecured custom web service
------------------------------------

@Unsecured
def retrievePublicData(Map params) {
       return [data:["data item 1", "data item 2"]]
}
```

> Be cautious in using this annotation as it may potentially present a security risk if sensitive data is returned from the web service.

Returning errors

In order to signify an error occurred or invalid data was provided, the WebServiceException class may be thrown from any custom web service. This exception contains constructors and fields for a list of messages and a HTTP response code. For example, suppose that the user provided inadequate information. The web service could use the following code to notify the user and prompt them to take action with custom messages.

```
def service(Map params) {
       // Handle invalid input
       if (!params.int('a'))
               throw new WebServiceException("Invalid parameter 'a' specified, please specify an
integer!", 400)
       // Use params.a correctly …
}
```

For the example above, a 400 response code (bad request) would be returned with a response body as follows:

```
{
  "messages":[
       "Invalid parameter 'a' specified, please specify an integer!"
  ]
}
```

If any other exception is thrown from a web service (ie Exception, IllegalArgumentException, etc.), a 500 response code will be returned with the following response body:

```
{
  "messages":[
       "A problem occurred while processing the request",
       "Message provided in the exception constructor"
  ]
}
```

See **Responses and return codes** on page 1407 for more information on error formats in MWS.

Accessing the HTTP Request Method

The HTTP method used for the request is available from the Map parameters argument. The key used to access it is stored as a static field in PluginConstants called `WEB_SERVICES_METHOD`. The value is a string which can be `GET`, `POST`, `PUT`, or `DELETE`. The following example demonstrates how this could be used with the `WebServiceException` to create a REST API with a plugin.

```
def serviceMethod(Map params) {
        // Check to make sure that this request used the HTTP GET method
        // Throw a 405 error (method not supported) if not
        if (params[PluginConstants.WEB_SERVICES_METHOD]!="GET")
                throw new WebServiceException("Method is not supported", 405)
}
```

Related topics

- **Plugin developer's guide** on page 1657

Reporting state data

As long as Moab Workload Manager is configured with MWS as a Resource Manager (RM), plugins may report state information on jobs, nodes, storage, and virtual machines to Moab. This is done through **Reports** that are generated by the plugin and passed to the bundled RM services (**Job RM service**, **Node RM service**, **Storage RM service**, and **Virtual machine RM service**). Each report is for a specific type of object: job, node, storage, or virtual machine. Each contains current state information on the specific attributes of the type it is for.

Note that storage is a sub-type of node, meaning that it is a specialized node.

Generating reports

To generate a report, simply create a new instance of a report depending on the type of object to be reported:

Object type	Report type
Job	JobReport
Node	NodeReport
Storage	StorageReport
Virtual Machine	VirtualMachineReport

Each report has a single required parameter for creating a new instance—the ID of the object which is being reported. Once the report instance has been created, any property may be modified as shown in the API documentation links in the table above. The following example shows the creation of a simple node report and modification of a few properties:

```
public void poll() {
       NodeReport node = new NodeReport("node1")
       node.timestamp = new Date()
       node.image = "centos-5.4-stateless"
       … // Set other properties and persist the report
}
```

Master and slave reports

At times, you may want to report some additional attributes on objects *only if* the objects are being reported by other plugins. For example, you may want to report the power state of a VM, but sometimes the plugin reporting this data can receive data even after the VM has been destroyed. In this case, you can set the `slaveReport` field on any report to `true`, signifying that the report should only be used if another plugin is reporting on the same object (in other words, creating "master" reports).

> If all reports for an object are "slave" reports, and no "master" reports exist, then the object will not report to Moab Workload Manager.

Special cases in field values

All complex types, such as Lists, Maps, and objects (not including Enumerated values such as NodeReportState and JobReportState) have default values set for them and are not required to be instantiated before use. For example, the metrics property of a node report may be modified as follows:

```
public void poll() {
       NodeReport node = new NodeReport("node1")
       // The following assignments are equivalent in their functionality
       node.features.add("FEAT1")
       node.features << "FEAT2"
       // The following assignments are equivalent in their functionality
       node.metrics.METRIC1 = 4d
       node.metrics["METRIC2"] = 125.5
       … // Set other properties and persist the report
}
```

For the `resources` and `requirements` (jobs only) properties, assignments may be made easily without checking for previously existing values or null objects. For example, resources may be added to the `resources` property simply by accessing it as a Map:

```
public void poll() {
       NodeReport node = new NodeReport("node1")
       node.resources.RES1.total = 10
       node.resources.RES1.available = 3
       node.resources["RES2"].total = 10
       node.resources["RES2"].available = 10
       … // Set other properties and persist the report
}
```

The job report's `requirements` property has some additional handling to allow it to be accessed as a single JobReportRequirement object, such as in the following example:

```
public void poll() {
       JobReport job = new JobReport("job.1")
       job.nodeCountMinimum = 4
       job.processorCountMinimum = 2
       job.requiredNodeFeatures << "FEAT1"
       job.preferredNodeFeatures << "FEAT2"
       … // Set other properties and persist the report
}
```

Although multiple requirements may be added to the `requirements` list to provide consistency with the MWS Job resource (see **Jobs** on page 1525), only the first requirement object's properties will be reported to Moab through the RM interface.

Managing images for nodes

In order to have Moab Workload Manager recognize a node as a virtual machine hypervisor, it must have a valid associated Image (see **Images** on page 1514). In particular, the `image` property on a node report must set to a valid image name. The image's `hypervisorType` and `virtualizedImages` properties are then used to report the correct hypervisor type and supported virtual machine images to Moab.

If the `image` is invalid, it will be ignored and the node will not be recognized as a hypervisor. If the `image` is valid, but no `hypervisorType` value is present, the `extensions.xcat.hvType` field value will be used. If that is also not present, the configuration parameter for default hypervisor type (see **Configuration** on page 1750) will be used instead.

Persisting a Report

After a report has been generated and all desired fields have been updated, the report must be sent to one of the three bundled RM services for persisting. If this is not done, the report will be discarded and will not be considered when reporting state information to Moab. The RM services are shown below according to the object type that they handle:

Object type	RM service
Job	**Job RM service**
Node	**Node RM service**
Storage	**Storage RM service**
Virtual Machine	**Virtual machine RM service**

Each service has two methods: `save` and `update`. The difference between these is that the `save` method first removes all previous reports from the plugin calling the method, and then persists the new reports, thereby only persisting the latest reports, while the `update` method does not remove any reports before persisting the new reports. Typically, the `save` method will be used while a plugin is being polled, while the `update` method will be used in incremental event based reporting. An example of using the `save` method is shown below.

```
INodeRMService nodeRMService

public void poll() {
       NodeReport node = new NodeReport("node1")
       // Change the state
       node.state = NodeReportState.BUSY
       // Persist
       nodeRMService.save([node])
}
```

Once this is done, the reports will be persisted to MongoDB and will be included in RM queries (see **Resource manager queries** on page 1713) from Moab Workload Manager or users.

Related topics

- **Plugin developer's guide** on page 1657

Controlling lifecycle

> Plugin control is currently in Beta. Interfaces may change significantly in future releases.

At times a plugin developer may wish to modify the current state of a plugin or even create plugins programatically. This may be done with the **Plugin control service**. Operations exist on the service to:

- create plugin instances dynamically with specific configuration.
- retrieve plugin instances by ID or based on configuration properties.
- start or stop plugin instances.
- verify plugin instance configuration.

Creating plugins

Several methods are provided to allow on-the-fly creation of new plugins. Generally, they allow a plugin with a specific ID and plugin type (as a string or as a Groovy Class) to be created with optional configuration properties. These properties should match the fields in **Plugins** on page 1577.

If any configuration properties are omitted, the defaults will be used as described in **Setting default plugin configuration** on page 1725. A boolean value is also returned indicating whether the creation succeeded or not.

Note that the `createPlugin` methods will initialize the plugin for retrieval or usage and attempt to start the plugin if the `autoStart` property is true.

Retrieving plugins

Plugins may be retrieved by using an ID, querying by plugin type, or even querying based on configuration parameters. Several methods are provided to perform these functions as shown on **Plugin control service** on page 1728.

Starting and stopping plugins

Plugins may also be started or stopped on demand. These two methods are exposed directly as `start` and `stop` on the plugin control service. Although each method does not return any data, exceptions are thrown if errors are encountered.

Verifying plugin configuration

Finally, the plugin control service may be used to verify plugin configuration at any point instead of just when the plugin is started or modified. This may be useful to attempt to modify plugin configuration directly through the `setConfig` dynamic method (see **Dynamic methods** on page 1659) and then verify that the new configuration is valid for the plugin. Exceptions are thrown if the plugin or the configuration is invalid.

Examples

If an error state is detected it may be necessary to stop the current plugin instance until corrective action can be taken. This may be done using the following code:

```
package example

import com.adaptc.mws.plugins.*

class ErrorPlugin {
	IPluginControlService pluginControlService

	public void poll() {
		// Error is detected, stop plugin instance!
		try {
			log.warn("An error was detected, trying to stop the plugin ${id}")
			pluginControlService.stop(id)
			log.warn("The plugin was successfully stopped")
		} catch(PluginStopException e) {
			log.error("Plugin instance ${id} could not be stopped", e)
		}
	}
}
```

Related topics

- **Plugin developer's guide** on page 1657

Accessing MWS REST resources

Often a plugin type may need to access existing MWS REST Resources in order to extend or complement default MWS functionality. This may be done with the **Moab REST service**, which allows a plugin type developer to utilize the existing Resources documentation see **Resources introduction** on page 1424) to perform these tasks.

All accesses to resources require a HTTP method to use (such as GET, POST, PUT, or DELETE) and a relative URL (such as `/rest/jobs`). Although it mimics the REST resource interface, no actual requests are made and no data is transmitted through the network.

Authentication

All resources are available to the Moab REST Service, and no authentication or Application Accounts are needed.

> Caution must be used when developing plugin types, as there are no restrictions to what may be done with the Moab REST Service. This is especially true when not utilizing hooks as discussed below.

Hooks

If pre and post-processing hooks are utilized in MWS (**Pre and post-processing hooks** on page 1412), the plugin type developer may choose whether or not they are executed when performing a "request" through the Moab REST service. This is done through the hooks option as documented in **Moab REST service** on page 1726.

Verifying API version support

The Moab REST Service provides a method for easily determining which API versions are supported by the current version of MWS. This method includes checks to make sure that the API version will work as expected, including verifying any configuration or external services are running.

```
moabRestService.isAPIVersionSupported(1)
moabRestService.isAPIVersionSupported(2)
```

Converting string dates

Because the Moab REST Service returns data exactly as given to an external consumer of MWS, including dates converted to strings, the service provides a method for converting MWS date strings to actual Date objects.

```
moabRestService.convertDateString("2011-11-08 13:18:47 MST")
```

URL parameters

URL parameters, such as `query`, `sort`, `proxy-user`, and others should be not be appended directly to the URL. Instead, these may be specified with the `params` option:

```
// Query images that are hypervisors
moabRestService.get("/rest/images", params:[query:'{"hypervisor":true}'])
// Sort images by osType
moabRestService.get("/rest/images", params:[sort:'{"osType":1}'])
```

Examples

This code retrieves a list of all nodes, and is equivalent to the **Get all nodes** task.

```
package example

import com.adaptc.mws.plugins.*
import net.sf.json.*

class RestPlugin {
       IMoabRestService moabRestService

       public void poll() {
               def result = moabRestService.get("/rest/nodes")
               // OR with the hook enabled…
               def result = moabRestService.get("/rest/nodes", hooks:true)

               assert result instanceof MoabRestResponse
               assert nodes instanceof List

               log.debug("Nodes list:")
               nodes.each { JSON node ->
                       log.debug(node.id)
               }
       }
}
```

This code adds a flag to a job, and is equivalent to the **Modify job attributes** task. This request also enables the hook (if one is configured) for the "request" and uses a URL parameter. This is the equivalent of making a call to `/rest/jobs/job.1?proxy-user=adaptive`.

```
package example

import com.adaptc.mws.plugins.*
import net.sf.json.*

class RestPlugin {
       IMoabRestService moabRestService

       public void poll() {
               def jobId = "job.1"
               def result = moabRestService.put("/rest/jobs/"+jobId, hooks:true, params:['proxy-
user':'adaptive']) {
                       [flags:["RESTARTABLE"]]
               }
               assert result.isSuccess()
       }
}
```

Related topics

- **Plugin developer's guide** on page 1657

Creating events and notifications

Plugins may easily create new events and create or update notification conditions using the **Plugin event service**. Previously, this was only possible by utilizing the MWS REST resources. The event service eases this burden from plugin developers. There are several operations that are available using the service:

- Create a event with or without specifying an event date.
- Create an event from a enumeration annotated with `EventEnumeration` (see **Plugin event service** on page 1733) with or without specifying an event date.
- Create or update a notification condition with or without specifying an observed date or expiration duration.

Creating events

Events are composed of several properties such as arguments, associated objects, origin, message, severity, escalation level, and a unique event code. The plugin event service removes the need for magic strings such as those for event severity ("INFO", "WARN", "FATAL") and also handles creating unique event codes. In other words, no bitwise manipulation is required to create new events.

The event code is comprised of several elements:

Code element	Description
Severity	If the event is informational, a warning, an error, or fatal.
Escalation level	Who cares about the event, or who should act on the event.
Component code	Internally made up of the MWS component code (stored internally) and the plugin event component code (see **Plugin event component code** on page 1683).
Entry code	The code representing a unique event for the component (for each plugin event component code).

The plugin event service handles the severity, escalation level, and entry code portions of the code by the values passed as parameters to the `createEvent` method. The plugin event component code is described in the next section.

Plugin event component code

The plugin event component code should be a unique number across all plugin types or projects from `1-254`. This number is combined with the MWS component code to represent each plugin as a unique component code across all Adaptive Computing products. `0` is reserved for MWS itself and should not be used. `255` is reserved for plugin types that do not define an event component code and represents an "unknown" plugin component. Additionally, codes `1-150` are reserved for Adaptive Computing plugins, while `151-254` are reserved for Professional Services and/or customer-specific plugins.

This code may be specified by setting an `eventComponent` property (see **Fields: Plugin Types** on page 2035) on the plugin project file or as a static property on the plugin type. As with all other project properties, the plugin type value overrides the project value. For example:

```
class MyExampleProject {
	…
	Integer eventComponent = 2
	…
}
ExamplePlugin {
	static final eventComponent = 1
	…
}
Example2Plugin {
	// no eventComponent property
	…
}
```

In this case, the plugin type `ExamplePlugin` has a plugin event component code of `1`, while the `Example2Plugin` has a code of `2` since it inherits it from the project properties.

Origin suffix

The origin of an event created through the plugin event service is automatically set by the plugin framework to `MWS/plugins/<plugin type>/<plugin id>`. For example, an event created by the plugin created from the "ExamplePlugin" plugin type with an ID of "plugin1" would generate events with an origin of `MWS/plugins/Example/plugin1`.

While this origin is sufficient for an administrator to determine the plugin where the event came from, the plugin developer may want this to be more specific to a class name or method name. This may be done using the optional `originSuffix` parameter to the `createEvent` method. The origin suffix, as its name implies, is appended to the end of the generated origin. For the example above, suppose the plugin developer passed `myMethod/switch1` as the origin suffix parameter when creating a new event. The event would then have an origin of `MWS/plugins/Example/plugin1/myMethod/switch1`.

Event enumerations

While creating events using the plugin event service is quite simple, often there are related events that have properties in common, such as the event type prefix or the origin suffix. Additionally, i18n messages (see **i18n messaging** on page 1661) are typically used for the event's message. Using the `EventEnumeration` annotation (see **Plugin event service** on page 1733) in combination with a enumeration simplifies this process. When this is done, each message is pulled from the `messages.properties` files using a standard convention, and the event type prefix and the origin suffix may optionally added as static properties on the enumeration. Using `EventEnumeration` requires:

- The annotated element is an `enum`, not a `class` or `interface`.
- Each enumeration value must use the constructor with three arguments: the event name, the severity, and the escalation level.
- If an event type prefix is specified, it must be defined as "static String EVENT_TYPE_PREFIX = ...", otherwise the property should not be defined.
- If an origin suffix is specified, it must be defined as "static String ORIGIN_SUFFIX = ...", otherwise the property should not be defined.

If any of these conditions are not fulfilled, using the `EventEnumeration` annotation will result in compilation errors.

Enumeration values are automatically marked as implementing the `IPluginEvent` interface and may be used as the first parameter of the `createEvent` method on the plugin event service. For example:

```
package example

import com.adaptc.mws.plugins.EventEnumeration
import com.adaptc.mws.plugins.IPluginEventService.AssociatedObject
import static com.adaptc.mws.plugins.IPluginEventService.Severity.*
import static com.adaptc.mws.plugins.IPluginEventService.EscalationLevel.*

public class ExamplePlugin {
	void poll() {
		// Event 1 takes no arguments
		pluginEventService.createEvent(ExampleEvents.EVENT1, null, null)
		// Event 2 takes one argument and has an associated object
		pluginEventService.createEvent(ExampleEvents.EVENT2, ["arg1"], [new AssociatedObject
(type:"type1", id:"id1")])
	}
}

@EventEnumeration
enum ExampleEvents {
	EVENT1("Example One", INFO, USER),	// Entry code is 0
	EVENT2("Example Two", INFO, USER)	// Entry code is 1
}
```

It may be noted that several key properties of events are missing from the enumeration definition and create event call parameters:

- Message: retrieved automatically from i18n messages (see **Messages for event enumerations** on page 1685)
- Event type: generated from the enumeration constructor and optional event type prefix property (see **Event type for event enumerations** on page 1686)
- Entry code: generated from the return value of `ordinal()` on the enumeration value; in other words, this is generated from the order of the enumeration values

Messages for event enumerations

The message for events created from enumerations is generated using i18n messages (see **i18n messaging** on page 1661) with codes in the following format:

- `<enumeration type name>.<enumeration value name>.message`
- `<enumeration type name>.<enumeration value name>.comment`

Considering the example in the section above, the message for `ExampleEvents.EVENT1` would be generated using the argument list passed to the `createEvent` method with the "ExampleEvents.EVENT1.message" message from `messages.properties`. This message should contain arguments if needed, such as "My example with ID {0} was created" and is used as the "message" property in the created event. The comment, on the other hand, is not persisted with the event and should be text (typically in paragraph format) describing why the event typically occurs or what actions should be taken when it does occur. Consider the message to contain instance specific information for the event (passed as arguments to the message) and the comment to be general documentation concerning the event.

As a best practice, name event enumeration values using the number and short name of each argument to the message. This makes it easy for the consumer to know which arguments are expected and what each means. For example, if an event is for connection errors and needs two arguments to the message, the URL and the error message, the enumeration value should be named "CONNECT_FAILURE_1URL_2ERROR" or even "CONNECT_TO_1URL_FAILURE_2ERROR". In this way, the consumer knows that the first argument represents the URL and the second is the error message.

Event type for event enumerations

As described above, the static string field `EVENT_TYPE_PREFIX` may be defined on the enumeration. This value is optional and, when present, is prepended with a space to the event name parameter from the constructor to generate the event type. For example, consider the following enumeration:

```
package example

import com.adaptc.mws.plugins.EventEnumeration
import static com.adaptc.mws.plugins.IPluginEventService.Severity.*
import static com.adaptc.mws.plugins.IPluginEventService.EscalationLevel.*

@EventEnumeration
enum MyPluginEvents {
       CONNECT("Connect", INFO, ADMIN),
       DISCONNECT("Disconnect", INFO, ADMIN)

       static String EVENT_TYPE_PREFIX = "My Plugin"
}
```

If `MyPluginEvents.CONNECT` and `MyPluginEvents.DISCONNECT` were used with the plugin event service, the generated event types would be "My Plugin Connect" and "My Plugin Disconnect" respectively.

Origin for event enumerations

The origin for event enumeration values automatically contains more information than those for non-enumerated events, such as those described above. The enumeration type name and value are appended to the origin. For example, consider the following enumeration and plugin fragment:

```
…
class ExamplePlugin {
       …
       assert id=="example1"   // plugin ID is example1
       pluginEventService.createEvent(ExampleEvents.EVENT1, null, null)
       …
}
…
@EventEnumeration
enum ExampleEvents {
       EVENT1("Event One", INFO, ADMIN)
       ...
```

The origin generated for the created event would be `MWS/plugins/Example/example1/ExampleEvents/EVENT1`. The static string field `ORIGIN_SUFFIX` may also be defined on the enumeration. This value is optional and, when present, is appended to the end of the generated origin as described above with the origin suffix parameter to the `createEvent` method.

Example

In order to understand all interactions when event enumerations are used, the following is a complete example.

```
Plugin type
------------------------------------

package example
import com.adaptc.mws.plugins.*

class ConnectPlugin extends AbstractPlugin {
	static eventComponent = 1

	IPluginEventService pluginEventService

	void poll() {
		def errorMessage = connect()
		if (errorMessage)
			pluginEventService.createEvent(ConnectEvents.CONNECT_TO_1URL_FAILURE_2ERROR,
[config.url, errorMessage], null)
		else
			pluginEventService.createEvent(ConnectEvents.CONNECT_SUCCESS, null, null)
	}

	// Returns the error message or null/empty on success
	private String connect() {
		String errorMessage
		…
		return errorMessage
	}
}
```

```
Event enumeration
------------------------------------

package example
import com.adaptc.mws.plugins.EventEnumeration
import static com.adaptc.mws.plugins.IPluginEventService.Severity.*
import static com.adaptc.mws.plugins.IPluginEventService.EscalationLevel.*

@EventEnumeration
enum ConnectEvents {
	CONNECT_SUCCESS("Success", INFO, ADMIN),
	CONNECT_TO_1URL_FAILURE_2ERROR("Failure", ERROR, ADMIN)

	static String EVENT_TYPE_PREFIX = "Connect"
}
```

```
messages.properties
------------------------------------

ConnectEvents.CONNECT_SUCCESS.message=The plugin was successfully connected!
ConnectEvents.CONNECT_SUCCESS.comment=This occurs when the plugin successfully
connects to the configured URL and
       is informational only.
ConnectEvents.CONNECT_TO_1URL_FAILURE_2ERROR.message=The plugin failed to connect to
{0}: {1}
ConnectEvents.CONNECT_TO_1URL_FAILURE_2ERROR.comment=This occurs when the plugin fails
to connect to the configured
       URL for any reason.  The most common reason is that the service is not running and
needs to be started.
```

The following are examples of the events created in MWS:

```
Created events
------------------------------------

{"totalCount": 2, "resultCount": 2, "results": [
       {
               "arguments": ["http://localhost:1000", "The service is not running!"],
               "code": 570523649,
               "eventDate": "2013-06-12 19:16:50 UTC",
               "eventType": "Connect Failure",
               "message": "The plugin failed to connect to http://localhost:1000: The service is
not running!",
               "origin": "MWS/plugins/Connect/connect/ConnectEvents/CONNECT_TO_1URL_FAILURE_
2ERROR",
               "severity": "ERROR",
               "id": "51b8c922a816c6a04af2401d",
               "associatedObjects": []
       },
       {
               "arguments": [],
               "code": 33652736,
               "eventDate": "2013-06-12 19:18:07 UTC",
               "eventType": "Connect Success",
               "message": "The plugin was successfully connected!",
               "origin": "MWS/plugins/Connect/connect/ConnectEvents/CONNECT_SUCCESS",
               "severity": "INFO",
               "id": "51b8c96fa816c6a04af24021",
               "associatedObjects": []
       }
]}
```

Unique event codes

The last topic that must be covered in creating events from plugins is that all efforts should be made to make sure that event codes are unique throughout all Adaptive Computing product suites. Additionally, the codes should be static, meaning they do not change once established. In order to do this, adhere the following recommendations:

- Use a unique (across all plugin types) plugin event component code for each plugin type.
- Follow the guidelines for plugin event component codes established above (see **Plugin event component code** on page 1683) and ensure it is a number 1-254.

- Use event enumerations where possible, otherwise ensure (through testing if possible) that all entry codes are unique for each plugin type.
- Ensure (through testing if possible) that the ordinal value of the event enumeration values do not change.

Creating or updating notification conditions

The plugin event service also makes it easy to create or update notification conditions. Simply use the `updateNotificationCondition` method. Just as the MWS notification condition resource, this is an idempotent operation, meaning it can be called multiple times with the same result. If the notification condition does not exist, it will be created automatically. If it does exist, the observed date and details will be updated accordingly.

Examples

Examples are available on **Plugin event service** on page 1733.

Related topics

- **Resources introduction** on page 1424
- **Events** on page 1506
- **Notifications** on page 1563
- **Notification conditions** on page 1558
- **Plugin developer's guide** on page 1657
- **Fields: Events** on page 1837
- **Plugin event service** on page 1733
- **Handling events** on page 1689
- **System events** on page 1422
- **Securing the connection with the message queue** on page 1395

Handling events

> Plugin events (excepting the poll event) are currently in Beta. Interfaces may change significantly in future releases.

Plugin types may handle specific events by containing methods defined by the conventions below. All events are optional.

The polling event

To maintain current information, each plugin is polled at a specified time interval. The following method definition is required to utilize the polling event.

```
void poll() { … }
```

Typically this polling method is used to report node and virtual machine information. By default, the polling interval is set to 30 seconds, but can be modified for all or individual plugins as explained in **Plugin management** on page 1720.

When a polling event occurs, the `poll` method on the target plugin is called. This method may perform any function desired and should typically make calls to the **Node RM service**, the **Virtual machine RM service**, and the **Job RM service** services to report the current state of nodes and virtual machines. For example, the `poll` method in the Native plugin type is implemented as follows:

This is an extremely simplified version of what is actually implemented in the Native plugin type.

```
INodeRMService nodeRMService;
IVirtualMachineRMService virtualMachineRMService;

public void poll() {
        nodeRMService.save(getNodes());
        virtualMachineRMService.save(getVirtualMachines());
}
```

This simple poll method calls two other helper methods called `getNodes` and `getVirtualMachines` to retrieve node and virtual machine reports. These reports are then sent to the appropriate RM service. See **Reporting state data** on page 1676 for more information on the RM services; however, the objective of this example is to demonstrate one possible use of the poll event handler. Other plugin types, on the other hand, may use the poll event to update internal data from pertinent resources or make calls to external APIs.

Lifecycle events

Events are also triggered for certain lifecycle state changes. The following method definitions are required to receive lifecycle events.

```
public void configure() throws InvalidPluginConfigurationException { … }
public void beforeStart() { … }
public void afterStart() { … }
public void beforeStop() { … }
public void afterStop() { … }
```

Each event is described in the table below with the associated state change when the event is triggered.

State change	Event	Description
configure	Configure	Triggered before `beforeStart` and after the plugin has been configured. May be used to verify configuration and perform any setup needed any time configuration is loaded or modified.
beforeStart	Start	Triggered just before starting a plugin.
afterStart	Start	Triggered just after a plugin has been started.

State change	Event	Description
beforeStop	Stop	Triggered just before stopping a plugin.
afterStop	Stop	Triggered just after stopping a plugin.

Currently, no events are triggered for pausing, resuming, erroring, or clearing errors for plugins.

RM events

When MWS is configured as a Moab Resource Manager (see **Moab Workload Manager resource manager integration** on page 1708, and more specifically, **Configuring Moab Workload Manager** on page 1709), RM events are sent from Moab to each plugin according to the routing specification (see **Routing** on page 1657). The following method definitions are required to receive these events.

```
public boolean jobCancel(String jobName) { … }
public boolean jobModify(String jobName, Map<String, Object> attributes, ModifyMode
modifyMode) { … }
public boolean jobRequeue(String jobName) { … }
public boolean jobResume(String jobName) { … }
public boolean jobStart(String jobName, List<String> nodes, String username) { … }
public boolean jobSubmit(Map<String, Object> job, String submissionString, String
submissionFlags) { … }
public boolean jobSuspend(String jobName) { … }
public boolean nodeModify(List<String> nodes, Map<String, String> attributes,
ModifyMode modifyMode) { … }
public boolean nodePower(List<String> nodes, NodeReportPower state) { … }
public boolean virtualMachinePower(List<String> virtualMachines, NodeReportPower
state) { … }
```

Related topics

- **Events** on page 1506
- **Notifications** on page 1563
- **Notification conditions** on page 1558
- **Plugin developer's guide** on page 1657
- **Fields: Events** on page 1837
- **Resources introduction** on page 1424
- **Plugin event service** on page 1733
- **Creating events and notifications** on page 1682

Handling exceptions

Plugin exceptions are currently in Beta. Interfaces may change significantly in future releases.

The `com.adaptc.mws.plugins` package contains several exceptions that may be used and in some cases, should be caught. All exceptions end with "Exception", as in PluginStartException.

There are several specific cases where Exceptions should or can be used:

- The `reload` method on the **Plugin control service** can throw the InvalidPluginConfigurationException to signify that the configuration contains errors.
- Various methods on the **Plugin control service** throw plugin exceptions which must be caught to diagnose errors when creating plugin types.
- Any exception (including the Exception class) can be thrown from a custom web service to display a 500 Internal Server Error to the client requesting the service with the given error message.

Related topics

- **Plugin developer's guide** on page 1657

Managing SSL connections

At times it is desirable to load and use self-signed certificates, certificates generated from a single trusted certificate authority (CA), or even simple server certificates. It may also be necessary to use client certificates to communicate with external resources. To ease this process, the **SSL service** may be utilized. This service provides methods to load client and server certificates from the filesystem. Methods are also present to aid in creating connections which automatically trust all server certificates and connections.

Several points should be noted when using the SSL Service:

- Certificate files may be in the PEM file format and do not need to be in the DER format (as is typical of Java security).
- Each method returns an instance of SSLSocketFactory, which may then be used to create simple sockets or, in combination with another client library of choice, create a connection.
- If the client certificate password is non-null, it will be used to decrypt the protected client certificate.
- This service is *not* needed when performing SSL communications with trusted certificates, such as those for HTTPS enabled websites that do not have a self-signed certificate.
- If the file name of the certificate file (client or server) is relative (no leading '/' character), it will be loaded from the `mws.certificates.location` configuration parameter (see **Configuration** on page 1750).
    - The default value of `mws.certificates.location is MWS_HOME/etc/ssl.crt`.
- Both the client certificate alias and password may be `null`. In this case, the client certificate must not be encrypted and the client certificate's default alias (the first subject CN) will be used.
- The lenient socket factory and hostname verifier automatically trust all server certificates. Because of this, they present a large security hole. Only use these methods in development or in fully trusted environments.

Example

To create a socket to a server that requires a client certificate, the following code may be used.

```
package example

import com.adaptc.mws.plugins.*

class SSLConnectionPlugin extends AbstractPlugin {
	ISslService sslService

	public void poll() {
		// This certificate is not encrypted and will be the only certificate presented to the
		// connecting end of the socket.
		// This file will be loaded from MWS_HOME + mws.certificates.location + my-cert.pem.
		String clientCert = "my-cert.pem"

		def socketFactory = sslService.getSocketFactory(clientCert, null, null)
		def socket = socketFactory.createSocket("hostname.com", 443)
		// Write and read from the socket as desired…
	}
}
```

To create a HTTPS URL connection to a server that has a self-signed certificate, the following code may be used. Note that this is very typical of client libraries – they have a method to set the SSL socket factory used when creating connections.

```
package example

import com.adaptc.mws.plugins.*

class SSLConnectionPlugin extends AbstractPlugin {
	ISslService sslService

	public void poll() {
		// This certificate represents either the server public certificate or the CA's certificate.
		// Since the path is absolute it will not be loaded from the MWS_HOME directory.
		String serverCert = "/etc/ssl/certs/server-cert.pem"

		def socketFactory = sslService.getSocketFactory(serverCert)

		// Open connection to URL
		HttpsURLConnection conn = "https://hostname.com:443/test".toURL().openConnection()
		conn.setSSLSocketFactory(socketFactory)

		// Retrieve page content and do with as desired…
		def pageContent = conn.getInputStream().text
	}
}
```

Related topics

- **Plugin developer's guide** on page 1657

Utilizing services or custom "helper" classes

There are three general types of services available for use in plugins:

- Bundled services such as the **Moab REST service**.
- Custom built translators loaded by convention of their name.

- Other custom built helper classes registered with Annotations.

These will each be described in this section.

Bundled services

Bundled services are utility classes that are included and injected by default onto all plugin types. It is not required to use any of these services, but they enable several core features of plugin types as discussed in **Utility services** on page 1655.

More information may be found on each bundled service in **Plugin services** on page 1725.

Using translators

Often a plugin type class file becomes so complex that it is desirable to split some of its logic into separate utility service classes. The most typical use case for this is to split out the logic for "translating" from a specific resource API to a format of data that the plugin type can natively understand and utilize. For this reason, there is a convention defined to easily add these helper classes called "Translators."

Simply end any class name with "Translator," and it will be automatically injected just as bundled services onto plugin types, other translators, or even custom registered components. The injection occurs only if a field exists on the class matching the name of the translator with the first letter lower-cased. For example, a translator class called "MyTranslator" would be injected on plugin types, other translators, and custom components that define a field called "myTranslator" as `def myTranslator` or `MyTranslator myTranslator`.

> Do not use two upper-case letters to start the class name of a Translator. Doing this may cause injection to work improperly. For example, use RmTranslator instead of RMTranslator as the class name.

> Be careful not to declare translator and custom component injection such that a cyclic dependency is created.

Logging in translators

All translators automatically have a "getLog" method injected on them which can be used to access the configured logger. It returns an instance of org.apache.commons.logging.Log.

```
package example

class ExampleTranslator {
	public void myMethod() {
		// log will be translated to getLog() by the groovy compiler
		log.info("Starting my method")
	}
}
```

See **Logging** on page 1660 for more information on logging configuration and usage.

Example

Suppose that a translator needs to be created to handle a connection to access an external REST resource. The translator could be defined as follows:

```
package example

class ExampleTranslator {
	public int getExternalNumber() {
		def number = … // Make call to external resource
		return number
	}
}
```

A plugin type can then use the translator by defining a field called "exampleTranslator". Note that an instance does not need to be explicitly created.

```
package example

class ExamplePlugin {
	def exampleTranslator
	// OR …
	//ExampleTranslator exampleTranslator

	public void poll() {
		// Use the translator
		log.info("The current number is "+exampleTranslator.getExternalNumber())
	}
}
```

To extend the example, the translator may also be injected into another translator:

```
package example

class AnotherTranslator {
	def exampleTranslator

	public int modifyNumber(int number) {
		return number + exampleTranslator.getExternalNumber()
	}
}
```

This translator may be used in the plugin type just as the other translator.

Registering custom components

There are cases where the concept of a "Translator" does not fit the desired use of a utility class. In these cases, it is possible to register any arbitrary class as a component to be injected just as a translator would be. This is done using the Spring Framework's annotation `org.springframework.stereotype.Component`. When this annotation is used, the class is automatically registered to be injected just as translators onto plugin types and translators.

> All annotations are available in the dependencies declared by the plugins-commons artifact.

> Do not use two upper-case letters to start the class name of a custom component. Doing this may cause injection to work improperly. For example, use RmUtility instead of RMUtility as the class name.

Changing scope

By default, when a custom component is injected, only a single instance is created for all classes which inject it. This is referred to as the 'singleton' scope. Another scope that is available is 'prototype', which creates a new instance every time it is injected. This is useful when the class contains state data or fields that are modified by multiple methods. To change the scope, use the `org.springframework.context.annotation`.Scope on the class with a single String parameter specifying "singleton" or "prototype."

Injecting translators or components

The need may arise to inject translators or other custom components onto custom components. This is done using the `org.springframework.beans.factory.annotation.Autowired` or `javax.annotation.Resource` annotations. The `Autowired` annotation is used to inject class instances by the type (i.e. `MyTranslator myTranslator`) while the `Resource` annotation is used to inject class instances by the name (i.e. `def myTranslator`). Add the desired annotation to the field that needs to be injected.

> Note that using the `Autowired` annotation does injection by type which differs from translator and plugin type injection. These are done by name just as the `Resource` annotation allows. Due to this fact, a type of "def" cannot be used when doing injection onto custom components using the `Autowired` annotation. See the example below.
>
> Injection of custom components *onto* translators and plugin types are still done by name, only fields injected using the `Autowired` annotation are affected.

> Be careful not to declare translator and custom component injection such that a cyclic dependency is created.

Logging in custom components

Unlike plugins and translators, custom components do *not* automatically have a "getLog" method injected on them. In order to log with custom components, you must use the Apache Commons Logging classes to retrieve a new log. The `PluginConstants` class contains the value of the logger prefix that is used for all plugins and translators. The following is an example of how to retrieve and use a logger correctly in a custom component.

```
package example

import com.adaptc.mws.plugins.PluginConstants
import org.apache.commons.logging.Log
import org.apache.commons.logging.LogFactory
import org.springframework.stereotype.Component

@Component
class ExampleComponent {
    private static final Log log = LogFactory.getLog(PluginConstants.LOGGER_
PREFIX+this.name)

        public void myMethod() {
                log.info("Starting my method")
        }
}
```

See **Logging** on page 1660 for more information on logging configuration and usage.

Example

Suppose that a custom utility class is needed to perform complex logic. A custom component could be defined as follows (notice the optional use of the `Scope` annotation):

```
package example

import org.springframework.stereotype.Component
import org.springframework.context.annotation.Scope

@Component
@Scope("prototype")
class ComplexLogicHandler {
        def handleLogic() {
                … // Perform complex logic and return
        }
}
```

A plugin type or translator could then be defined to inject this component:

```
package example

class CustomPlugin {
        def complexLogicHandler

        public void poll() {
                complexLogicHandler.handleLogic()
        }
}
```

Now suppose another custom component needs to use the ComplexLogicHandler in its code. It can inject it using the `Autowired` annotation:

```
package example

import org.springframework.stereotype.Component
import org.springframework.beans.factory.annotation.Autowired

@Component
class AnotherHandler {
	// Note that this is injected by type, so 'def' may not be used
	@Autowired
	ComplexLogicHandler complexLogicHandler

	def wrapLogic() {
		complexLogicHandler.handleLogic()
	}
}
```

To perform the same injection but by name (as translators and plugin types are injected), use the `Resource` annotation:

```
package example

import org.springframework.stereotype.Component
import javax.annotation.Resource

@Component
class AnotherHandler {
	// Note that this is injected by name based solely on the name defined in
	//	the annotation.  The name of the field itself does not affect the injection.
	@Resource(name="complexLogicHandler")
	def complexLogicHandler

	def wrapLogic() {
		complexLogicHandler.handleLogic()
	}
}
```

Related topics

- **Plugin developer's guide** on page 1657

Packaging plugins

Plugin types may be packaged in two different ways to upload to MWS:

- A simple Groovy file containing a single plugin type definition.
- A JAR file containing one or more plugin types, translators, and custom components.

While each may be uploaded to MWS using the REST API or the User Interface as described in **Add or update plugin types** on page 1717, using a JAR file is recommended. Using a simple Groovy file is useful for testing and generating proof of concept work, but does not allow the use of several features of plugins.

The principles of packaging a plugin type or set of plugin types in a JAR file are very simple. Simply compile the classes and package in a typical JAR file. All classes ending in "Plugin" are automatically attempted to be loaded as a plugin type, all classes ending in "Translator" are attempted to be loaded as a translator, and all classes annotated as a custom component will be attempted to be loaded. It is recommended that a build framework is used to help with compiling and packaging the JAR file, such as

Gradle. This makes it easy to declare a dependency on the necessary JAR files used in plugin development and to debug, compile, and test plugin code.

In addition to using utility services such as translators, packaging plugin types in JAR files allows the creation of a single project for multiple related plugin types and bundling of external dependencies. These two features are discussed in the following sections.

Plugin projects and metadata

Each plugin type has information attached to it, called metadata, which describes the origin and purpose of the plugin type. Additionally, a JAR file may also contain a project file which defines default metadata attributes for all plugin types in the JAR. Initial plugins, or plugins that will be created on loading of the JAR file if they do not exist, are also able to be defined on a project file. In all cases, metadata declared on a plugin type will override the metadata defined on the project file.

To define a project file, simply add a class to JAR file that ends in "Project." This file will attempted to be loaded as the project file. Every field on a project file, and even the file itself, is optional. All available fields are shown in the example below.

```
class SampleProject {
	// Plugin information
	String title = "Sample"
	String description = "Sample plugin types"
	String author = "Our Company."
	String website = "http://example.com"
	String email = "sample@example.com"
	Integer eventComponent = 1
	// Versioning properties
	String version = "0.1"
	String mwsVersion = "7.1 > *"
	String commonsVersion = "0.9 > *"
	String license = "APACHE"

	// Documentation properties
	String issueManagementLink = "http://example.com/ticket-system/sample-plugins"
	String documentationLink = "http://example.com/docs/sample-plugins"
	String scmLink = "http://example.com/git/sample-plugins"

	// Plugins that are to be created with these properties only when they do NOT exist
	// This does not override any existing plugin instance configuration
	def initialPlugins = {
		/*
		// Multiple instances of plugins may be defined here.
		// In this case, 'sample' is the id of the plugin
		sample {
				pluginType = "Sample"
				// All properties except for "pluginType" are optional
				pollInterval = 30
				autoStart = true
				// Although it is possible to set plugin precedence, it is not recommen
this precedence
				// may already be taken and plugin creation will fail in this case
				precedence = 5
				config {
					configParam = "value"
				}
			}
		}
		// Another plugin with an ID of 'sample2'
		sample2 {
			…
		*/
	}
```

As can be seen, metadata information about the plugin type(s), versions, and documentation are available. These are displayed when viewing plugin information in the User Interface or through the REST API.

Any of these properties except for `initialPlugins`, `mwsVersion`, and `commonsVersion` may be overwritten by the plugin type class itself by using static properties. A simple example is shown below.

```
package example

class SamplePlugin {
	// Properties may be typed, untyped, final, or otherwise,
	// but they MUST be static
	static version = "0.2"
	static title = "Sample plugin"
	static description = "This sample plugin is used to demonstrate metadata information"
	static author = "Separate Division"
	static eventComponent = 1

	… // Rest of the plugin type definition
}
```

Event component

The `eventComponent` field is explored in **Creating events and notifications** on page 1682.

MWS and commons versions

The `mwsVersion` and `commonsVersion` fields are used to restrict the versions of MWS and plugin framework with which the plugin project may be used. Each field is of the format `FIRST_VERSION > LAST_VERSION`, where `FIRST_VERSION` is the first supported MWS or plugin framework version (inclusive), and `LAST_VERSION` is the last supported MWS or plugin framework version (inclusive). Each version must take the format of #.# or #.#.#, as in 7.1, or 7.1.2. An asterisk (*) is used to denote any version, and may be used for the first or the last version.

Although support for restricting both the MWS and commons versions are provided, it is recommended to use the commons version restriction always and the MWS version restriction where necessary. Restrictions on the commons version prevent plugin loading errors while restrictions on the MWS version prevent runtime errors such as missing support for certain MWS API versions.

Typically the `mwsVersion` and `commonsVersion` fields are set as shown above, with the first version set to a specific number, and the last version set to any (an asterisk). This is the recommended approach for setting both fields. It is not recommended to use any version (asterisk) for the first version. Some examples of `mwsVersion` and `commonsVersion` values are shown below with explanations of how they behave.

```
String mwsVersion = "7.1 > *" // Any MWS version 7.1.0 and greater is supported
(including 7.2, etc)
String mwsVersion = "7.1.3 > *" // Any MWS version 7.1.3 and greater is supported
(including 7.2, etc)
String mwsVersion = "7.1 > 7.1.3" // Any MWS version between 7.1.0 and 7.1.3 is
supported
String mwsVersion = "* > *" // Any MWS version is supported (not recommended!)
String mwsVersion = "* > 7.2" // Any MWS version up to 7.2 is supported (not
recommended!)

String commonsVersion = "0.9 > *" // Any framework version 0.9.0 and greater is
supported (including 1.0, etc)
String commonsVersion = "0.9.3 > *" // Any framework version 0.9.3 and greater is
supported (including 1.0, etc)
String commonsVersion = "0.9 > 0.9.3" // Any framework version between 0.9.0 and 0.9.3
is supported
String commonsVersion = "* > *" // Any framework version is supported (not
recommended!)
String commonsVersion = "* > 1.0" // Any framework version up to 1.0 is supported (not
recommended!)
```

If the `mwsVersion` or `commonsVersion` fields are formatted incorrectly, the plugin project will fail to load. If a plugin project is uploaded to MWS and the version check fails, the project will fail to load with an error message about the `mwsVersion` or `commonsVersion`.

> The `mwsVersion` and `commonsVersion` fields cannot be overridden by a single plugin type, but can be set only at the plugin project level. This prevents mixing of MWS and commons version requirements within a single project.

Initial plugins

The initial plugins closure provides the flexibility to insert plugin instances when the JAR is loaded. This occurs at two points: when the plugin JAR is first uploaded to MWS, and when MWS is restarted. As shown in the example above, the ID, pluginType, and other properties may be configured for multiple plugins.

The nature of Groovy closures means that programmatic definition of initial plugins is possible. This may even be based on the MWS application configuration. Two properties are automatically available in the initialPlugins closure:

- `appConfig` – Contains the MWS application configuration. Any configuration parameter is available for access as documented on **Configuration** on page 1750.
- `suite` – Contains the currently configured suite that MWS is running in. This is equivalent to the `mws.suite` configuration parameter, and is an instance of Suite.

Native plugin case study

The Native JAR file utilizes many of the features discussed above. In the root of the JAR file, a compiled class called NativeProject exists which defines all of the metadata fields, including `initialPlugins`. Trying to create an initial plugin presents two distinct problems:

- The plugin should be initialized only if the suite is CLOUD.
- The plugin type configuration must contain an entry referencing the configured `mws.home.location` parameter, or the configured MWS_HOME location.

The `initialPlugins` closure is defined as follows:

```
import com.adaptc.mws.plugins.Suite

class NativeProject {
	… // Metadata fields

	def initialPlugins = {
		// Initialize the cloud-native plugin only if the suite is CLOUD
		if (suite==Suite.CLOUD) {
			'cloud-native' {
				pluginType = "Native"
				pollInterval = 30
				config {
					// Use the appConfig property to retrieve the current MWS_HOM
					getCluster = "file://${appConfig.mws.home.location}/etc/nodes
				}
			}
		}
	}
}
```

Managing external dependencies

External dependencies (e.g. JAR files) may be included and referenced in JAR files. Certain rules must also be followed in order to have the dependencies loaded from the JAR file correctly:

The plugin type must bundle all external dependency JARs in the root of the plugin type JAR file.

An entry must be included in the `MANIFEST.MF` file that references each of these bundled JAR files as a space separated list:

```
Class-Path: dependency1.jar dependency2.jar dependency3.jar
```

Assuming that these rules are followed and that the plugin type is uploaded using the REST API or the User Interface, the dependent JARs will first be loaded and then the new plugin type and associated files will be loaded.

Documenting plugin types

Documentation may also be included in JAR files by placing one or more Markdown formatted files in the root of the project JAR file. These files will be processed dynamically by MWS and presented as documentation pages for the respective plugin types within the MWS plugin user interface pages. Markdown is a simple text-to-HTML format used in some of the most popular open-source repositories such as GitHub and BitBucket. To help provide plugin developers use a single place or file for documentation, the conventional use of "README.md" as documentation was followed within MWS.

Documentation file naming

Each documentation filename must start with "README" and end with ".md". If only one documentation file is needed for bundled plugin type(s), it is recommended to call the file "README.md". For multiple plugin types, the file name must contain the plugin type name without the "Plugin" suffix in the format of "README-<PluginName>.md". For example, if a plugin project JAR file contained the plugin type classes "MyPlugin", "ABTestPlugin", and "ImportantPlugin", the documentation files would be located in the root of the JAR file and would be called "README-My.md", "README-ABTest.md", and "README-Important.md" respectively. If a "README" file does not exist for a certain plugin type, the main "README.md" file (if provided) will be used as documentation for that plugin type.

Markdown syntax

The Markdown syntax supported by MWS is very close to GitHub Flavored Markdown. Internally, the pegdown Markdown processor is used to generate the HTML with the `TABLES`, `ABBREVIATIONS`, `FENCED_CODE_BLOCKS`, `SMARTYPANTS`, `DEFINITIONS`, and `QUOTES` extensions enabled. HTML tags may also be used directly in order to create more refined formatting of the documentation, but this is discouraged with the exception of inserting the configuration reference table discussed below.

For example, the `TABLES` extension may be used to easily create HTML tables:

```
Name | Notes
------ | -------
Bob    | Knows how to use MWS plugins but has never created one
George | Writes MWS plugins in his spare time
```

The only main difference from standard Markdown processors is that block quotes (marked by lines prepended with '> ') are shown as highlighted information boxes when displayed in MWS. This may be used to draw more attention to informational or warning messages without writing custom HTML.

```
 > **Warning:** The use of this plugin type requires that MWS and MWM are configured
correctly as described in
 > the MWS user guide.
```

Configuration reference table

A table of available configuration parameters is often constructed in documentation for each plugin type. To ease the burden on the plugin developer of maintaining this documentation and the constraints on the plugin type, a table generated from the constraints (see **Configuration constraints** on page 1664) and included messages is available by using the following HTML in the README file(s):

```
<div class="configuration-table">This section will be replaced by MWS with the
configuration parameters table</div>
```

The text within the `div` container may be anything, but should state something helpful such as that it is placeholder in cases where the documentation may be viewed within other contexts such as on GitHub.

The generated table includes the following columns for each configuration parameter listed in the constraints: name, key, required, type, description. The "name" and "description" values are retrieved from the "help" and "label" messages bundled in the plugin JAR (see the labels and help messages section in **Configuration constraints** on page 1664 for more information).

Web services reference sections

Documentation for exposed web services (see **Exposing web services** on page 1674) is also able to be generated automatically. Instead of a single table as done with configuration parameters, a section with several tables (possible URL access points, URL parameters, and response fields) and additional information is generated for each exposed web service. This is available by using the following HTML in the README file(s):

```
<div class="webservice-sections">This section will be replaced by MWS with the web
service documentation</div>
```

The text within the div container may be anything, but should state something helpful such as that it is placeholder in cases where the documentation may be viewed within other contexts such as on GitHub.

Changing heading sizes

The generated sections each begin with an <h2> heading with the name of the web service. If a different heading size (h3, h4, etc.) is desired, this may be done with the following HTML:

```
<div class="webservice-sections" data-level="3">This section will be replaced by MWS
with the web service documentation</div>
```

Notice the data-level attribute, which contains the number used in the HTML h tag.

Message codes

Just as with the configuration table, the data for the content is generated automatically from the web service method name and from i18n messages (see **i18n messaging** on page 1661) bundled in the plugin JAR file. Message codes are available to customize the label and description of the web service. Codes are also available to define an arbitrary number of URL parameters and response fields. These do not need to be defined, but are helpful. The following table defines each message used in generating the documentation for web services.

Name	Message code	Description
Web Service Label	<pluginType>.webServices.<webServiceMethod>.label	The label used as the heading for the section, defaults to the naturally capitalized method name if not present.
Web Service Description	<pluginType>.webServices.<webServiceMethod>.help	Paragraph text describing the web service and its functionality, outputs, etc.
Parameter Key	<pluginType>.webServices.<webServiceMethod>.parameter<n>.key	The nth URL parameter, starting at 1 (example: id).
Parameter Label	<pluginType>.webServices.<webServiceMethod>.parameter<n>.label	The label for the nth URL parameter, defaults to the naturally capitalized key if not present.

Name	Message code	Description
Parameter Type	<pluginType>.webServices.<webServiceMethod>.parameter<n>.type	The type for the `n`th URL parameter, defaults to `String` if not present.
Parameter Description	<pluginType>.webServices.<webServiceMethod>.parameter<n>.help	The description or help text for the `n`th URL parameter.
Response Field Key	<pluginType>.webServices.<webServiceMethod>.return<n>.key	The nth response field, starting at 1 (example: `success`).
Response Field Label	<pluginType>.webServices.<webServiceMethod>.return<n>.label	The label for the `n`th response field, defaults to the naturally capitalized key if not present.
Response Field Type	<pluginType>.webServices.<webServiceMethod>.return<n>.type	The type for the nth response field, defaults to `String` if not present.
Response Field Description	<pluginType>.webServices.<webServiceMethod>.return<n>.help	The description or help text for the `n`th response field.

As an example, suppose that a web service method called "doSomething" exists on a plugin type named "MyExamplePlugin". This web service expects two URL parameters: `id`, an integer, and `action`, a string. The response body consists of a JSON object with two fields: `success`, a boolean value, and `messages`, a list of strings. The following messages would serve to generate helpful documentation:

```
messages.properties
------------------------------------

# web service messages
myExamplePlugin.webServices.doSomething.label=Do Something Important
myExamplePlugin.webServices.doSomething.help=This web service does something important
with the input parameters.
# parameters
myExamplePlugin.webServices.doSomething.parameter1.key=id
myExamplePlugin.webServices.doSomething.parameter1.label=ID
myExamplePlugin.webServices.doSomething.parameter1.type=Integer
myExamplePlugin.webServices.doSomething.parameter1.help=The identifier of an object
myExamplePlugin.webServices.doSomething.parameter2.key=action
myExamplePlugin.webServices.doSomething.parameter2.label=Action # same as the default
would be
myExamplePlugin.webServices.doSomething.parameter2.type=String # same as the default
would be
myExamplePlugin.webServices.doSomething.parameter2.help=The action to perform

# response fields
myExamplePlugin.webServices.doSomething.return1.key=success
myExamplePlugin.webServices.doSomething.return1.label=Success # same as the default
would be
myExamplePlugin.webServices.doSomething.return1.type=Boolean
myExamplePlugin.webServices.doSomething.return1.help=True if the request succeeded,
false otherwise
myExamplePlugin.webServices.doSomething.return1.key=messages
myExamplePlugin.webServices.doSomething.return1.label=Error Messages
myExamplePlugin.webServices.doSomething.return1.type=List of Strings
myExamplePlugin.webServices.doSomething.return1.help=Error messages describing the
reason why success is false.
```

Note that if the *first* URL parameter key is `id`, the listed resource URLs will include the optional URL with the `id` parameter inline, such as `/rest/plugins/<pluginId>/services/<webService>/<id>`. Therefore, it is recommended to use `id` as parameter `1` if the web service expects a parameter with that key.

Related topics

- **Plugin developer's guide** on page 1657

Example plugin types

Several plugin types are provided by Adaptive Computing for use in Moab Web Services. Examples of these include the Native and vCenter plugin types.

A sample plugin type in Groovy would resemble the following:

```
package sample

import com.adaptc.mws.plugins.*

class SamplePlugin extends AbstractPlugin {
        static author = "Adaptive Computing"
        static description = "A simple plugin in groovy"
        static version = "0.1"

        INodeRMService nodeRMService

        public void configure() throws InvalidPluginConfigurationException {
                def myConfig = config   // "config" is equivalent to getConfig() in groovy
                def errors = []
                if (!myConfig.arbitraryKey)
                        errors << "Missing arbitraryKey!"
                if (errors)
                        throw new InvalidPluginConfigurationException(errors)
        }

        public void poll() {
                NodeReport node = new NodeReport("node1")
                node.resources.RES1.total = 5
                node.resources.RES1.available = 5
                node.state = NodeReportState.IDLE
                nodeRMService.save([node])
        }

        // Access at /rest/plugins/<id>/services/example-service
        public def exampleService(Map params) {
                return [success:true]
        }
}
```

Related topics

- **Plugin developer's guide** on page 1657

Moab Workload Manager resource manager integration

Moab Workload Manager possesses the concept of Resource Managers (RMs). While plugins can be related to RMs, they often provide greater functionality and serve more purposes than a typical RM. MWS must be represented in Moab as a RM to enable certain plugin features such as state reporting and handling RM events. This section describes the process of configuring Moab and additional details of its queries to MWS. It includes the following topics:

- **Configuring Moab Workload Manager** on page 1709
- **Resource manager queries** on page 1713

Related topics

- **About Moab Web Services plugins** on page 1650

Configuring Moab Workload Manager

During each iteration of Moab Workload Manager's cycle, it will query MWS through the RM interface to access current node, virtual machine, and job information. At this point, all reports are loaded from the database and consolidated into a single report of each object as explained in **Data consolidation** on page 1655.

> All unset (or null) values for properties on reports are ignored.

In some cases it may be desired to query MWS directly for the current consolidated node, storage, virtual machine, and job reports. This may be done using the following URLs which return data in a format that is a subset of the API version 3 interface for each object (i.e. `/rest/nodes?api-version=3, /rest/vms?api-version=3, /rest/jobs?api-version=3`).

Query	Description
`/rest/plugins/all/rm/cluster-query?api-version=3`	Retrieves consolidated node, storage, and virtual machine reports from all plugins.
`/rest/plugins/<ID>/rm/cluster-query?api-version=3`	Retrieves consolidated node, storage, and virtual machine reports for the specified plugin ID.
`/rest/plugins/all/rm/workload-query?api-version=3`	Retrieves consolidated job reports from all plugins.
`/rest/plugins/<ID>/rm/workload-query?api-version=3`	Retrieves consolidated job reports for the specified plugin ID.

These queries have no effect on the data itself. In other words, reports are not removed or manipulated when RM queries are performed. These are manipulated only the RM services as described in **Reporting state data** on page 1676.

Examples

The following example uses cURL (see **cURL samples** on page 1749) to perform the query.

```
$ curl -u moab-admin:changeme! http://localhost:8080/mws/rest/plugins/all/rm/cluster-
query?api-version=3&pretty=true
{
    "nodes": {
        "n1.test": {
            "states": {
                "state": "IDLE"
            },
            "lastUpdatedDate": 1382386344,
            "resources": {
                "processors": {
                    "configured": 4
                },
                "memory": {
                    "configured": 8191,
                    "available": 7206
                },
                "gres1": {
                    "configured": 100
                }
            },
            "metrics": {
                "cpuLoad": 0.008233333333333334,
                "vmcount": 0,
                "cpuUtilization": 0.2008333333333333
            },
            "featuresReported": [
                "feature1"
            ],
            "ipAddress": "10.0.8.69",
            "operatingSystem": {
                "hypervisorType": "esx",
                "image": "vcenter-vcenter-esx-4.x",
                "virtualMachineImages": [
                    "centos6-v7"
                ]
            },
            "variables": {
                "VCENTER_DATASTORE_REMOTE1": "datastore-448",
                "VCENTER_DATASTORE_LOCAL1": "datastore-411"
            },
            "attributes": {
                "MOAB_DATACENTER": {
                    "value": "vcenter-datacenter-401",
                    "displayValue": "vcenter-vcenter - adaptive data center"
                }
            }
        },
        "n2.test": {
            "states": {
                "state": "IDLE"
            },
            "lastUpdatedDate": 1382386344,
            "resources": {
                "processors": {
                    "configured": 4
                },
                "memory": {
                    "configured": 10239,
                    "available": 9227
                },
                "gres1": {
```

```
                    "configured": 100
                }
            },
            "metrics": {
                "cpuLoad": 0.00805,
                "vmcount": 0,
                "cpuUtilization": 0.19666666666666666
            },
            "featuresReported": [
                "feature1",
                "feature2"
            ],
            "ipAddress": "10.0.8.76",
            "operatingSystem": {
                "hypervisorType": "esx",
                "image": "vcenter-vcenter-esx-5.0",
                "virtualMachineImages": [
                    "centos6-v7",
                    "centos6",
                    "win2008"
                ]
            },
            "variables": {
                "VCENTER_DATASTORE_REMOTE1": "datastore-448",
                "VCENTER_DATASTORE_LOCAL1": "datastore-415"
            },
            "attributes": {
                "MOAB_DATACENTER": {
                    "value": "vcenter-datacenter-401",
                    "displayValue": "vcenter-vcenter - adaptive data center"
                }
            }
        },
        "n3.test": {
            "states": {
                "state": "IDLE"
            },
            "lastUpdatedDate": 1382386344,
            "resources": {
                "processors": {
                    "configured": 4
                },
                "memory": {
                    "configured": 10239,
                    "available": 9229
                },
                "gres1": {
                    "configured": 100
                }
            },
            "metrics": {
                "cpuLoad": 0.0097,
                "vmcount": 0,
                "cpuUtilization": 0.2375
            },
            "featuresReported": [
                "feature1"
            ],
            "ipAddress": "10.0.8.72",
            "operatingSystem": {
                "hypervisorType": "esx",
                "image": "vcenter-vcenter-esx-5.0",
```

```
                "virtualMachineImages": [
                    "centos6-v7",
                    "centos6",

                    "win2008"
                ]
            },
            "variables": {
                "VCENTER_DATASTORE_REMOTE1": "datastore-448",
                "VCENTER_DATASTORE_LOCAL1": "datastore-416"
            },
            "attributes": {
                "MOAB_DATACENTER": {
                    "value": "vcenter-datacenter-401",
                    "displayValue": "vcenter-vcenter - adaptive data center"
                }
            }
        },
        "n4.test": {
            "states": {
                "state": "IDLE"
            },
            "lastUpdatedDate": 1382386344,
            "resources": {
                "processors": {
                    "configured": 4
                },
                "memory": {
                    "configured": 10239,
                    "available": 9229
                },
                "gres1": {
                    "configured": 100
                }
            },
            "metrics": {
                "cpuLoad": 0.007883333333333334,
                "vmcount": 0,
                "cpuUtilization": 0.1925
            },
            "featuresReported": [
                "feature2"
            ],
            "ipAddress": "10.0.8.77",
            "operatingSystem": {
                "hypervisorType": "esx",
                "image": "vcenter-vcenter-esx-5.0",
                "virtualMachineImages": [
                    "centos6-v7",
                    "centos6",
                    "win2008"
                ]
            },
            "variables": {
                "VCENTER_DATASTORE_REMOTE1": "datastore-448",
                "VCENTER_DATASTORE_LOCAL1": "datastore-958"
            },
            "attributes": {
                "MOAB_DATACENTER": {
                    "value": "vcenter-datacenter-401",
                    "displayValue": "vcenter-vcenter - adaptive data center"
                }
```

```
            }
        }
    },
    "vms": {
        "vm1": {
            "states": {
                "state": "DOWN",
                "powerState": "OFF"
            },
            "host": {
                "name": "n1.test"
            },
            "lastUpdatedDate": 1382386344,
            "resources": {
                "processors": {
                    "configured": 4
                },
                "memory": {
                    "configured": 12288
                }
            },
            "metrics": {
                "vmcount": 1
            }
        }
    },
    "storage": {}
}
```

Related topics

- **Moab Workload Manager resource manager integration** on page 1708
- **Resource manager queries** on page 1713

Resource manager queries

Moab Workload Manager must be configured to use MWS as a resource manager. Do the following:

- First, the following lines must be in the Moab Workload Manager configuration file or one of its included files:

```
RMCFG[mws]                        TYPE=MWS
RMCFG[mws]                        FLAGS=UserSpaceIsSeparate
RMCFG[mws]                        BASEURL=http://localhost:8080/mws
```

  *The **BASEURL** must match the configured URL of MWS.*

- The next step is to edit the MWS credential information in the Moab private configuration file (`/opt/moab/etc/moab-private.cfg`, by default). Here are the default values:

```
CLIENTCFG[RM:mws] USERNAME=moab-admin PASSWORD=changeme!
```

> **USERNAME** and **PASSWORD** must match the values of `auth.defaultUser.username` and `auth.defaultUser.password`, respectively, found in the MWS configuration file. The MWS RM contacts MWS directly using the base URL, username, and password configured.
>
> Optionally, the **USERNAME** and **PASSWORD** configuration values may be specified directly in the Moab configuration file, though this is not recommended. Likewise, the **BASEURL** configuration value can be specified in the Moab private configuration file.

- Lastly, to enable such actions as submitting jobs as different users, the **ENABLEPROXY**=*TRUE* option must be present in the **ADMINCFG** configuration line, and the **OSCREDLOOKUP** option must be set to *NEVER*, as follows:

```
ADMINCFG[1]            USERS=root       ENABLEPROXY=TRUE
OSCREDLOOKUP           NEVER
```

- You may also want to configure SSL by using the following options (in either the RMCFG or CLIENTCFG section):
  - **SSLCACERT**: Lets you specify the absolute path to your SSL CA certificate. (This also enables the use of self-signed certificates, if desired.) It is recommended that you set this option in the Moab private configuration file. For example:

    ```
    CLIENTCFG[RM:mws]     SSLCACERT=/path/to/cert.pem
    ```

  - **SSLNOHOSTCHECK**: Lets you disable the SSL check to make sure that the actual server name matches the certificate's server name. For example:

    ```
    #In moab-private.cfg
    CLIENTCFG[RM:mws]     SSLNOHOSTCHECK=TRUE

    #Or in moab.cfg
    RMCFG[mws]            SSLNOHOSTCHECK=TRUE
    ```

    > WARNING: This setting could compromise the security of the system and should not be used in production environments.

  - **SSLNOPEERCHECK**: Lets you disable the SSL check to make sure that the certificate is valid.

    ```
    #In moab-private.cfg
    CLIENTCFG[RM:mws]     SSLNOPEERCHECK=TRUE

    #Or in moab.cfg
    RMCFG[mws]            SSLNOPEERCHECK=TRUE
    ```

    > WARNING: This setting could compromise the security of the system and should not be used in production environments.

Related topics

- **Moab Workload Manager resource manager integration** on page 1708
- **Configuring Moab Workload Manager** on page 1709

Plugin type management

Plugin types may be managed and accessed with Moab Web Services dynamically, even while running. Operations are provided to upload (add or update) plugin types and to list or show current plugin types. The available fields that are displayed with plugin types are given in **Fields: Plugin Types** on page 2035. For more information on how these fields are set, see **Plugin projects and metadata** on page 1699.

> Plugin Type JAR or groovy files should never be manually copied into the `MWS_HOME/plugins` directory. They must be managed using the methods shown in this section or through the REST API (see **Plugin types** on page 1585).

Bundled plugin types are included automatically in Moab Web Services releases and may be utilized immediately after startup. See **Plugin management** on page 1720 for more information on how to utilize these plugin types.

> The plugin type documentation is now located in the plugin type management pages. See **Plugin type documentation** on page 1716 for more information.

This section contains these topics:

- **Listing plugin types** on page 1715
- **Displaying plugin types** on page 1716
- **Plugin type documentation** on page 1716
- **Add or update plugin types** on page 1717

Related topics

- **About Moab Web Services plugins** on page 1650

Listing plugin types

To list all plugin types, browse to the MWS home page (for example, `https://servername/mws`). Log in as the admin user, then click **Plugins > Plugin Types**.

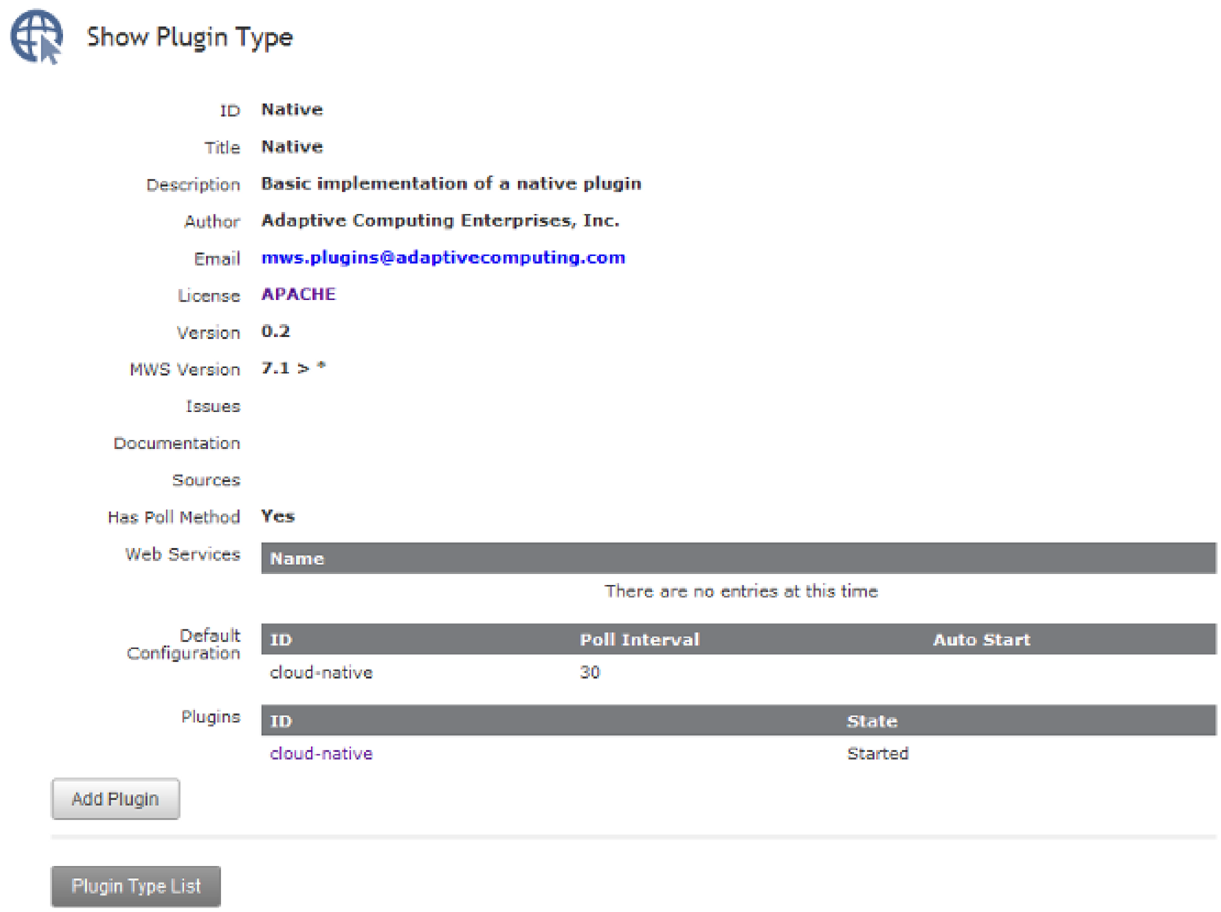
ID	Title	Author	Version	Has Poll Method	Instances
MSM	Moab Services Manager (MSM)	Adaptive Computing Enterprises, Inc.	0.2	Yes	0
Native	Native	Adaptive Computing Enterprises, Inc.	0.2	Yes	1

Related topics

- **Plugin type management** on page 1715

Displaying plugin types

To show information about a plugin type, go to the **Plugin Type List** page and click the desired plugin type.



Related topics

- **Plugin type management** on page 1715

Plugin type documentation

To show the documentation for a plugin type, go to the **Plugin Type List** page and click the desired plugin type. Then, click the **Open Documentation** button. This will display any documentation bundled with the plugin type.

Related topics

- **Plugin type management** on page 1715

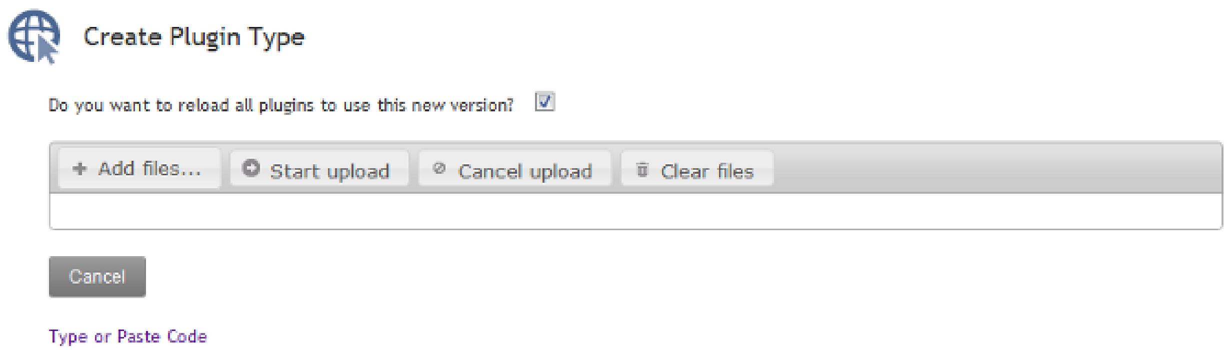
Add or update plugin types

Plugin types can be uploaded into Moab Web Services using a Groovy file, a Java Archive (JAR) file, or pasted Groovy code. To access the plugin type upload page, navigate to the **Plugin Type List** page and click **Add or Update Plugin Type**. The default interface of this page enables the uploading of a single Groovy class file or a JAR file.

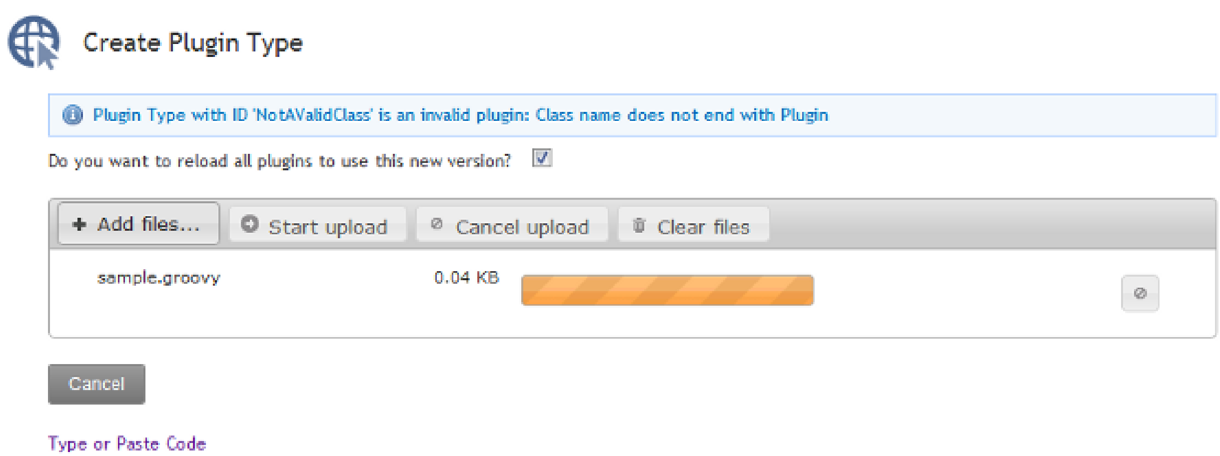
When a plugin type is updated, by default all corresponding plugins created from the plugin type will be recreated. If this behavior is not desired, clear the **Do you want to reload all plugins to use this new version?** checkbox before uploading the plugin type.

Single class file

Groovy files containing a single plugin type may be uploaded at the `/mws/admin/plugin-types/create` URL.



If the upload failed or an error occurred during initialization of the plugin, an error message will be displayed.

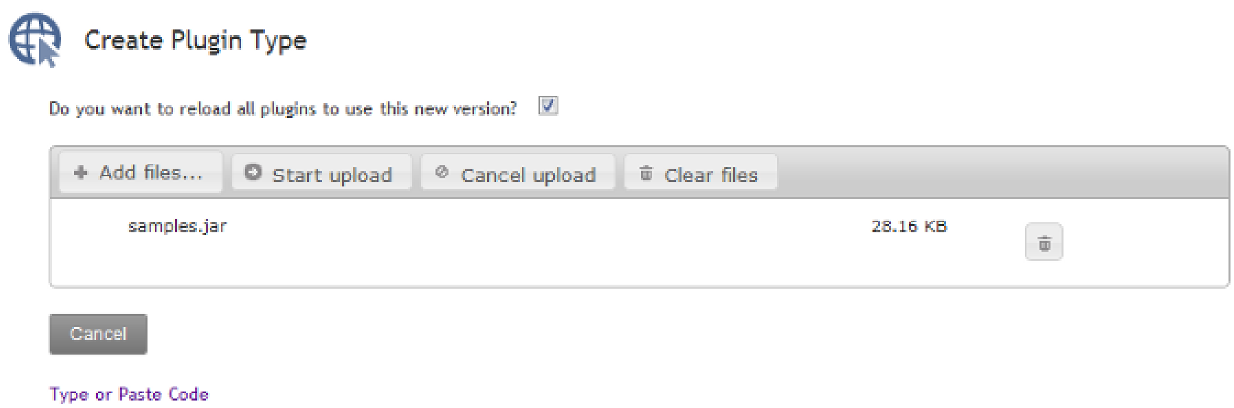


JAR file

A JAR file, as described in **Packaging plugins** on page 1698, containing one or more plugins may also be uploaded using the same process as the Groovy file.

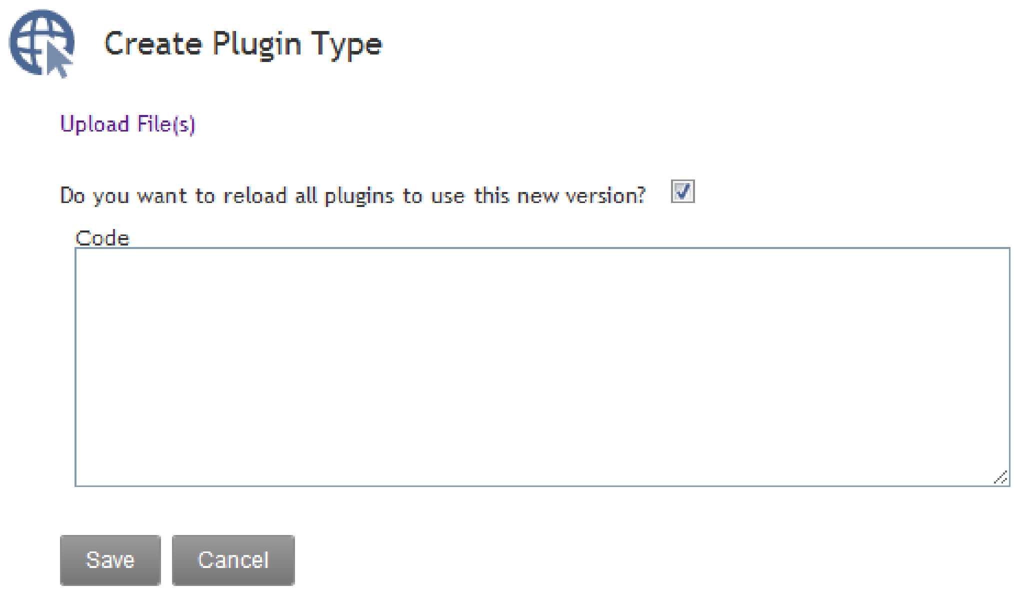
Click **Add files...**, select the `.jar` file, and click the **Start upload** button. If the upload failed or an error occurred during initialization of the plugin(s), an error message will be displayed.

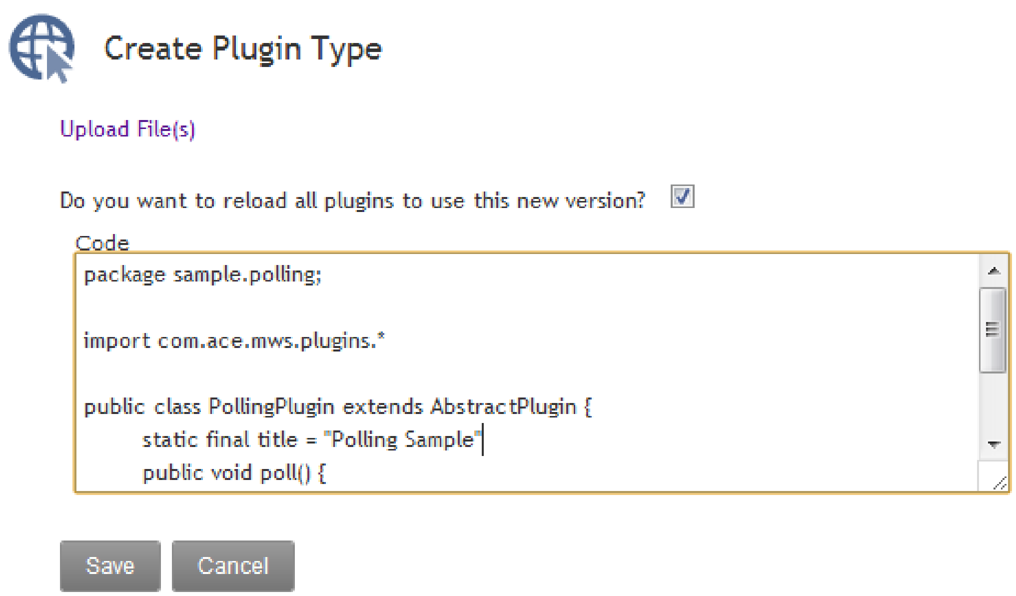
The JAR upload process differs from the single file in that if successful, the name of the JAR file itself is displayed instead of the plugin name(s).



Code

To paste or type code directly into MWS and have it be loaded as a single class file, click **Type or Paste Code**, and type or paste the code into the presented text box.





When the code is in the box, click **Create**. If the upload succeeded and the code was able to be compiled as Groovy, the browser will be redirected to the **Show Plugin Type** page. If the upload failed or an error occurred during compilation or initialization of the plugin, an error message will be displayed.

> You may need to refer to the MWS log file for additional details and error messages in the case of a failure.

Related topics

- **Plugin type management** on page 1715

Plugin management

Plugins may be managed and accessed with Moab Web Services dynamically, even while running. This includes plugin instance and lifecycle management. Additionally, default configuration values may be set for new plugins. In order to access custom web services, the REST API must be utilized as described in **Accessing Plugin Web Services** on page 1583. The available fields that are displayed with plugins are given in the **Fields: Plugins** reference.

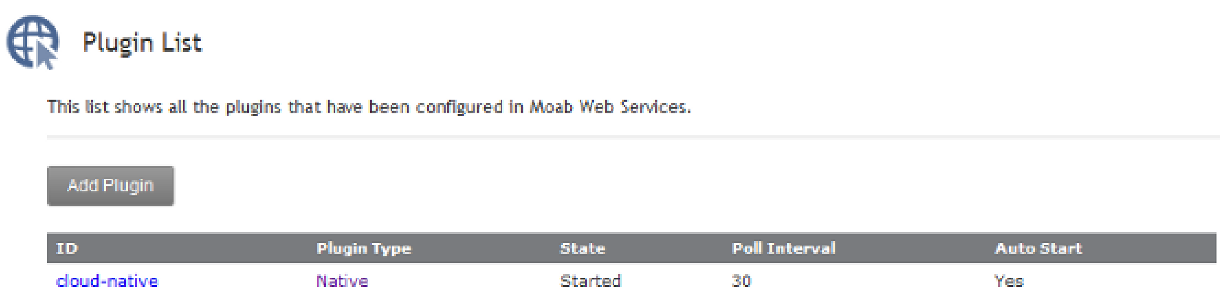
This section contains these topics:

Related topics

- **About Moab Web Services plugins** on page 1650

Listing plugins

To list all plugins, browse to the MWS home page (for example, `https://servername/mws`). Log in as the admin user, then click **Plugins > Plugins**.

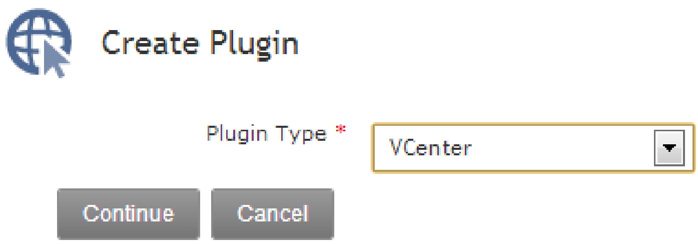


Related topics

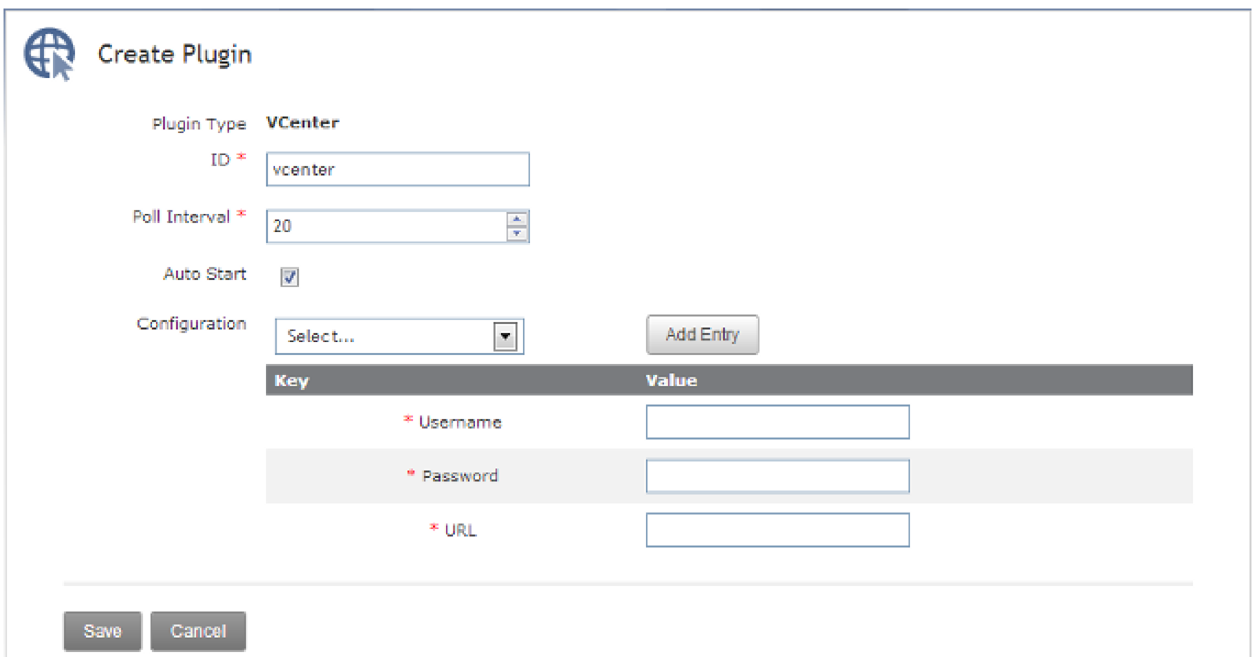
- **Plugin management** on page 1720

Creating a plugin

To create a plugin, go to the **Plugin List** page and click **Add Plugin**. First, a **Plugin Type** must be selected to continue to actually create the plugin.



The page is automatically built to support the plugin type's constraints (see **Configuration constraints** on page 1664). The **ID** field will be automatically filled in with a suggested value, and the **Poll Interval** field will be displayed only if the plugin type has a `poll` method. The required configuration fields are displayed by default, and optional fields may be selected and added to the configuration from the drop down at the top of the configuration section. See the **Fields: Plugins** reference section for more information on the fields.

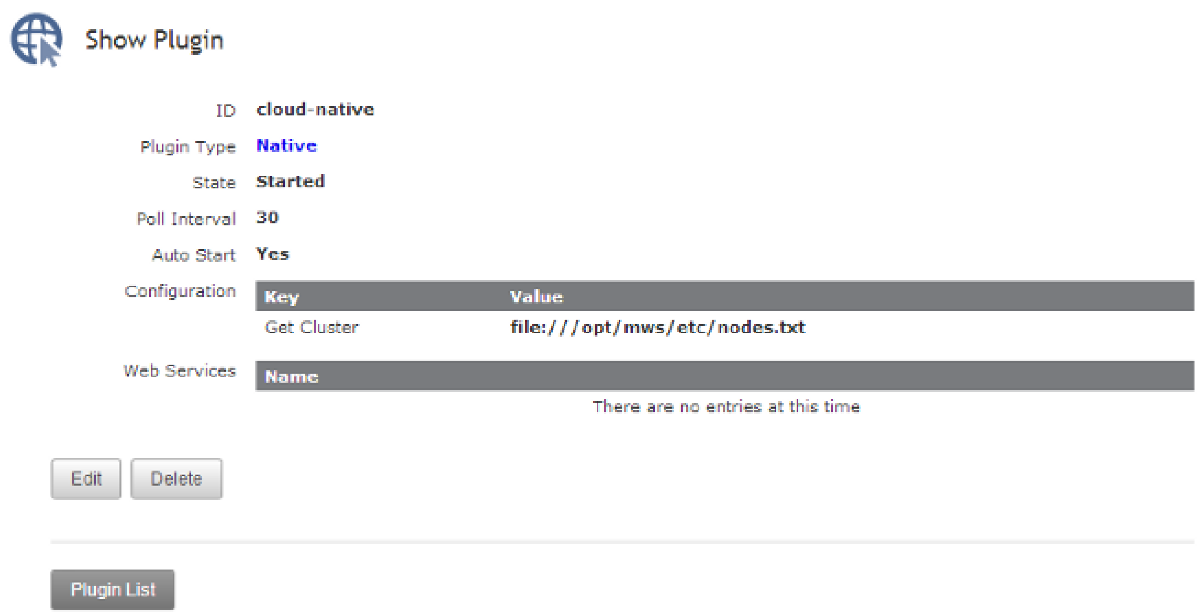


Related topics

- **Plugin management** on page 1720

Displaying a plugin

To show information about a plugin, go to the **Plugin List** page and click the desired plugin ID.

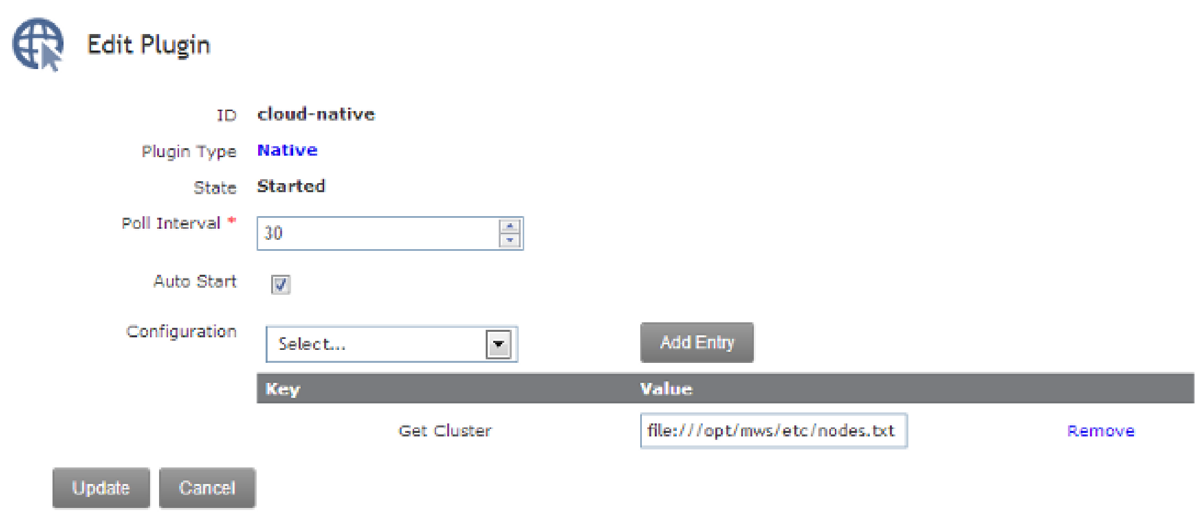


Related topics

- **Plugin management** on page 1720

Modifying a plugin

To modify a plugin, go to the **Plugin List** page, click the desired plugin ID, and then click **Edit**. See the **Fields: Plugins** reference section for more information on available fields.



Related topics

- **Plugin management** on page 1720

Deleting a plugin

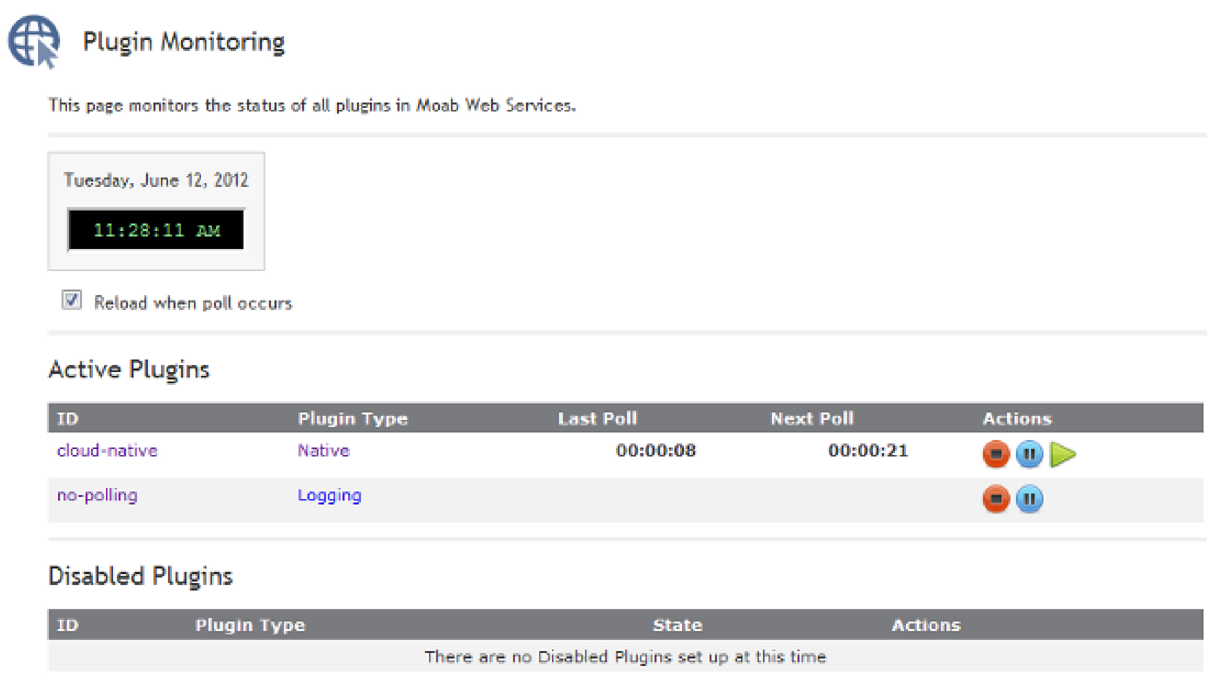
To delete a plugin, go to the **Plugin List** page, click the desired plugin ID, and then click **Delete**. A confirmation message is shown. If the **OK** button is clicked, the plugin is deleted from the system and cannot be recovered, including all configuration.

Related topics

- **Plugin management** on page 1720

Monitoring and lifecycle controls

To monitor and control the lifecycle of plugins, browse to the MWS home page (for example, `https://servername/mws`). Log in as the admin user, then click **Plugins > Plugin Monitoring**. This page displays the current state of all plugins as well as their polling status.

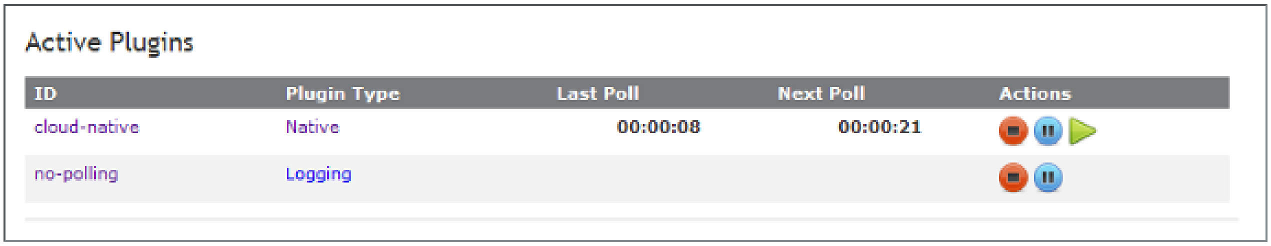


> If plugins are created from plugin types which do not have a `poll` method, their lifecycle controls will be limited. Any information below which mentions polling does not apply to the 'no-polling' plugin shown in the screenshots.

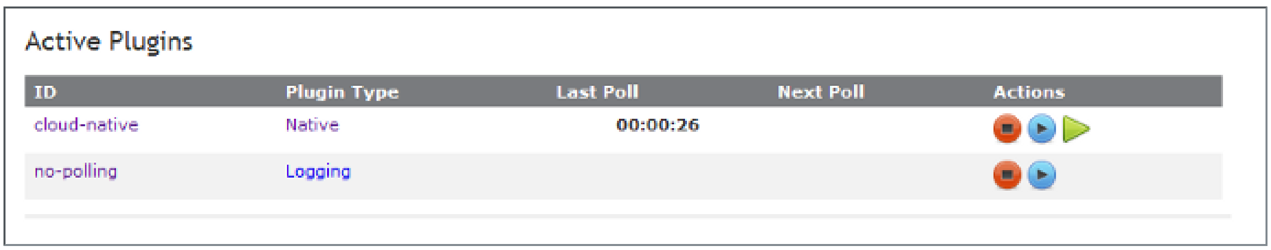
Active plugins

Active plugins are those which are in the Started or Paused states. These are available to receive events such as polling. If paused, a plugin will not receive events but is not actually stopped, therefore no stop events are triggered.

The following images demonstrate the status of plugins in the active states.



Started plugins which can include the relative time of the last poll as well as the time of the next poll in a countdown format. Action buttons are available to stop or pause the plugin as well as trigger an immediate poll event.

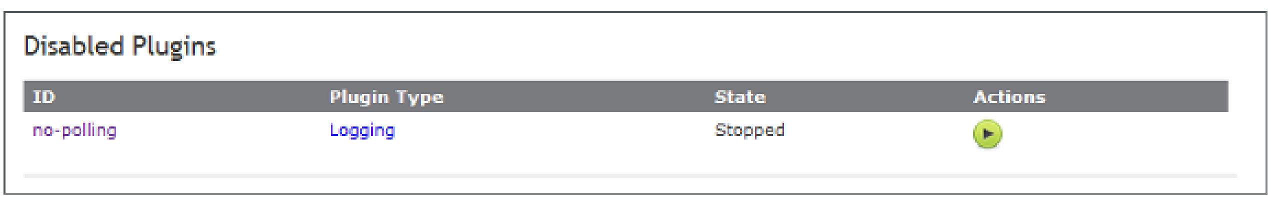


Paused plugins which can include only the last polling time. Action buttons are available to stop or resume the plugin, as well as trigger an immediate poll event.

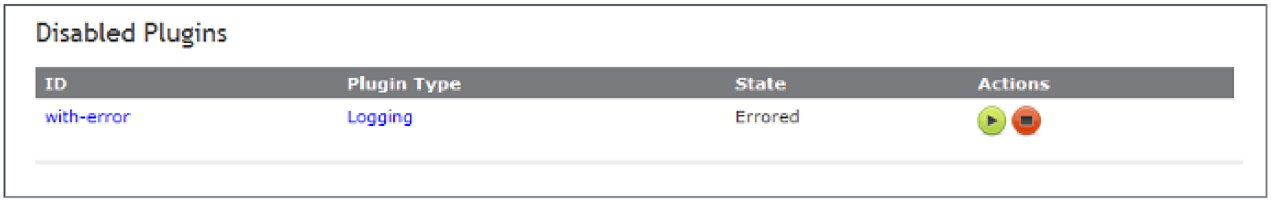
Disabled plugins

Disabled plugins are those which are in the Stopped or Errored states. These plugins do not receive events such as polling. If errored, a plugin may either be stopped, which represents a "clearing" of the error, or started normally. However, if no action is taken on an errored plugin, it likely will not start due to the fact that most plugins are put into the errored state during startup of the plugin.

The following images demonstrate the representation of plugins in the disabled states.



Stopped plugins. A single action button is available to attempt to start the plugin.



An errored plugin. As mentioned previously, action buttons are available to stop the plugin or clear the error as well as attempt to start the plugin. If the start fails, an error message will be displayed.

Related topics

- **Plugin management** on page 1720

Setting default plugin configuration

Configuration of default values for plugin configuration parameters involves setting fields in the MWS configuration file. These values are used if no values are provided when creating a new plugin. Additionally, the default values will be displayed to the user on the **Create Plugin** page.

The parameters to configure are documented on **Configuration** on page 1750 and comprise most values starting with plugins.

Related topics

- **Plugin management** on page 1720

Plugin services

To use the built-in services, declare a variable with the correct name as a property in the plugin class.

The convention for each service name is to remove the leading "I" and lower case the resulting first letter. For example, the property to use the IMoabRestService would be called moabRestService. The following is an example of using the IPluginControlService in this manner.

```
Using the IPluginControlService
------------------------------------

package example;
import com.adaptc.mws.plugins.*;

class ExamplePlugin {
       IPluginControlService pluginControlService;

       public poll() {
               // Use service…
               pluginControlService.stop("pluginId");
       }
}
```

Use of the Groovy anonymous type "def" may also be used. For example, the service definition above would use `def pluginControlService` instead of `IPluginControlService pluginControlService`.

Do *not* attempt to create a new instance of the services before use, such as in a constructor. The services will be automatically injected before any methods are called on the plugin.

API documentation

The `com.adaptc.mws.plugins` package contains interfaces for all bundled services available to plugin types. These may be used as discussed above. All services begin with "I" and end with "Service", as in IMoabRestService (**Moab REST service** on page 1726).

Related topics

- **About Moab Web Services plugins** on page 1650

Job RM service

The job RM service may be used to report job state data to Moab Workload Manager through the RM interface. See **Reporting state data** on page 1676 for more information. It may also be used to retrieve previous reports made by a plugin. Please note that due to data consolidation (see **Data consolidation** on page 1655), old job reports may no longer exist in the database by the time the query is done.

The `jobRMService` property will be injected with a class of type IJobRMService in all plugin types. Note that it is not available for injection in translators or custom components.

Related topics

- **Plugin services** on page 1725

Moab REST service

The Moab REST service may be used to access the MWS RESTful API (see **Resources introduction** on page 1424) in plugins. All "requests" made through this service are internal only and no data is actually transmitted over the network. See **Accessing MWS REST resources** on page 1680 for more information.

The `moabRestService` property will be injected with a class of type IMoabRestService in all plugin types.

Accessing resources

In order to access a resource, a relative URL matching that in the documentation must be used along with a HTTP method, such as GET, POST, PUT, or DELETE. The method names on IMoabRestService match the HTTP methods directly. For example, to call a GET operation on `/rest/jobs`, use the following code:

```
moabRestService.get("/rest/jobs")
```

Using parameters correctly

Although the ordering of the parameters for each method on IMoabRestService may seem confusing at first glance, this is to allow for easy use with Groovy. Examples are given below for each combination of parameters:

```
String URL
------------------------------------

moabRestService.get("/rest/jobs")
```

```
Map options, String URL
------------------------------------

moabRestService.get("/rest/jobs", hooks:true, contentType:"application/json")
```

```
String URL, Closure data
------------------------------------

moabRestService.get("/rest/jobs/job.1") {
       [flags:"RESTARTABLE"]
}
```

```
Map options, String URL, Closure data
------------------------------------

moabRestService.get("/rest/jobs/job.1", hooks:true, contentType:"application/json") {
       [flags:"RESTARTABLE"]
}
```

Options

The following options are valid in each method call supporting the `options` parameter:

Name	Type	Default	Description
data	See **Valid data types** on page 1728	--	Specifies the body of the "request." This can be overwritten by the `data` Closure parameter.
hooks	Boolean	false	Specifies whether or not hooks (see **Pre and post-processing hooks** on page 1412) are run as part of the "request."
contentType	String	application/json	Indicates the content type used for the request.
params	Map	--	Indicates URL query parameters to use for the "request," such as `query`, `sort`, `proxy-user`, or others.

Valid data types

If the data Closure parameter is specified, it overwrites the data option. In each case, there are four valid types for the data option or return value of the data closure:

- A non-null JSON instance.
- A valid JSON string. This will be converted into a JSON instance.
- A valid Map instance. This will be converted into a JSONObject instance.
- A valid List instance. This will be converted into a JSONArray instance.

> A JSONException may be thrown if the JSON string is invalid or the Map or List contains values that cannot be serialized to JSON.

Related topics

- **Plugin services** on page 1725

Node RM service

The node RM service may be used to report node state data to Moab Workload Manager through the RM interface. See **Reporting state data** on page 1676 for more information. It may also be used to retrieve previous reports made by a plugin. Please note that due to data consolidation (see **Data consolidation** on page 1655), old node reports may no longer exist in the database by the time the query is done.

The nodeRMService property will be injected with a class of type INodeRMService in all plugin types. Note that it is not available for injection in translators or custom components.

Related topics

- **Plugin services** on page 1725

Plugin control service

> This service is currently in Beta. Interfaces may change significantly in future releases.

The control service allows lifecycle management operations to be performed on plugins. It also provides methods to create and retrieve plugins. Note that the plugin control service may be used by other plugins, allowing one plugin to dynamically create, retrieve, start, or stop plugins.

The pluginControlService property will be injected with a class of type IPluginControlService in all plugin types.

Examples

```
Create plugin with default configuration
------------------------------------

try {
       if (pluginControlService.createPlugin("myPlugin", "Native"))
               log.info "myPlugin was created successfully!"
       else
               log.warn "There was an error creating myPlugin"
} catch(PluginStartException e) {
       log.warn "There was a problem starting the new plugin: ${e.message}"
} catch(InvalidPluginConfigurationException e) {
       log.warn "There were errors with the plugin's configuration: ${e.errors}"
}
```

```
Create plugin with custom configuration
------------------------------------

if (pluginControlService.createPlugin("myPlugin", "Native", [autoStart:false,
pollInterval:600]))
       log.info "myPlugin was created successfully!"
else
       log.warn "There was an error creating myPlugin"
```

```
Start plugin
------------------------------------

try {
       pluginControlService.start("myPlugin")
} catch(PluginStartException e) {
       log.warn "There was a problem starting the plugin: ${e.message}"
} catch(InvalidPluginException) {
       log.warn "The plugin 'myPlugin' is invalid"
} catch(InvalidPluginConfigurationException e) {
       log.warn "The plugin has an invalid configuration: ${e.errors}"
}
```

```
Stop plugin
------------------------------------

try {
       pluginControlService.stop("myPlugin")
} catch(PluginStopException e) {
       log.warn "There was a problem stopping the plugin: ${e.message}"
} catch(InvalidPluginException) {
       log.warn "The plugin 'myPlugin' is invalid"
}
```

```
Configure plugin
------------------------------------

try {
       pluginControlService.configure("myPlugin")
} catch(InvalidPluginException) {
       log.warn "The plugin 'myPlugin' is invalid"
} catch(InvalidPluginConfigurationException e) {
       log.warn "The plugin has an invalid configuration: ${e.errors}"
}
```

Related topics

- **Plugin services** on page 1725

Plugin datastore service

The datastore service is provided to allow a plugin to persist data to the database that is isolated from all other persistent data. In other words, this service provides access to a plugin's individual datastore (see **Individual datastore** on page 1672).

The `pluginDatastoreService` property will be injected with a class of type IPluginDatastoreService in all plugin types. Note that it is not available for injection in translators or custom components.

Examples

```
Adding a single custom entry
------------------------------------

package example

public class ExamplePlugin {
       def pluginDatastoreService

       public void poll() {
               def collectionName = "collection1"
               def data = [:]
               … // Add data here to the Map
               if (pluginDatastoreService.addData(collectionName, data))
                       log.info("Data successfully added")
               else
                       log.warn("There was an error adding the data")
       }
}
```

```
Adding multiple entries
------------------------------------

package example

public class ExamplePlugin {
       def pluginDatastoreService

       public void poll() {
               def collectionName = "collection1"
               def dataList = []
               dataList.add( /* Custom Map of data here */)
               dataList << … // Custom Map of data here
               if (pluginDatastoreService.addData(collectionName, dataList))
                       log.info("Data entries successfully added")
               else
                       log.warn("There was an error adding the data entries")
       }
}
```

```
Updating a single entry
------------------------------------

package example

public class ExamplePlugin {
	def pluginDatastoreService

	public void poll() {
		def collectionName = "collection1"
		def data = [:]
		… // Add data here to the Map
		if (pluginDatastoreService.updateData(collectionName, "key", "value", data))
			log.info("Data successfully updated")
		else
			log.warn("There was an error updating the data")
	}
}
```

```
Querying if a collection exists
------------------------------------

package example

public class ExamplePlugin {
	def pluginDatastoreService

	public void poll() {
		def collectionName = "collection1"
		if (pluginDatastoreService.exists(collectionName))
			log.info("Collection exists")
		else
			log.warn("The collection does not exist")
	}
}
```

```
Querying contents of a collection
------------------------------------

package example

public class ExamplePlugin {
	def pluginDatastoreService

	public void poll() {
		def collectionName = "collection1"
		def dataList = pluginDatastoreService.getCollection(collectionName)
		if (dataList!=null)
			log.info("Collection successfully queried")
		else
			log.warn("The collection does not exist!")
	}
}
```

```
Retrieving a single entry
------------------------------------

package example

public class ExamplePlugin {
	def pluginDatastoreService

	public void poll() {
		def collectionName = "collection1"
		def data = pluginDatastoreService.getData(collectionName, "key", "value")
		if (data!=null)
			log.info("Data successfully retrieved")
		else
			log.warn("The entry with key==value does not exist")
	}
}
```

```
Removing a collection
------------------------------------

package example

public class ExamplePlugin {
	def pluginDatastoreService

	public void poll() {
		def collectionName = "collection1"
		def data = pluginDatastoreService.clearCollection(collectionName)
		// Data now contains the collection that was cleared
		if (data!=null)
			log.info("Collection successfully cleared")
		else
			log.warn("The collection does not exist!")
	}
}
```

```
Removing a single entry
------------------------------------

package example

public class ExamplePlugin {
	def pluginDatastoreService

	public void poll() {
		def collectionName = "collection1"
		if (pluginDatastoreService.removeData(collectionName, "key", "value"))
			log.info("Data entry successfully removed")
		else
			log.warn("The entry where key==value does not exist!")
	}
}
```

Related topics

- **Plugin services** on page 1725

Plugin event service

The event service is provided to ease the burden and reduce boilerplate code for creating new events and updating notification conditions. For more information on how to use this service, see **Creating events and notifications** on page 1682.

The `pluginEventService` property will be injected with a class of type IPluginEventService in all plugin types. Note that it is not available for injection in translators or custom components.

Examples

```
Creating a custom event
------------------------------------

package example

import com.adaptc.mws.plugins.IPluginEventService.Severity
import com.adaptc.mws.plugins.IPluginEventService.EscalationLevel
import com.adaptc.mws.plugins.IPluginEventService.AssociatedObject

public class ExamplePlugin {
	def pluginEventService

	public void poll() {
		// Create a completely custom event
		pluginEventService.createEvent(Severity.INFO, EscalationLevel.USER, 0x4F, "Custom 
Type",
								"poll", "My event occurred", null, null)
	}
}
```

```
Creating a custom event with messages
------------------------------------

package example

import com.adaptc.mws.plugins.IPluginEventService.Severity
import com.adaptc.mws.plugins.IPluginEventService.EscalationLevel
import com.adaptc.mws.plugins.IPluginEventService.AssociatedObject

public class ExamplePlugin {
	def pluginEventService

	public void poll() {
	// Use i18n messages for another event
	def args = ["arg1", "arg2"]
		pluginEventService.createEvent(Severity.WARN, EscalationLevel.POWER_USER, 0x5F, 
"Custom Type",
								"poll", message
(code:"examplePlugin.customEvent.message", args:args), args,
								// AssociatedObjects or simple maps may be 
used
								[new AssociatedObject(type:"type1", id:"id1"), 
[type:"type2", id:"id2"])
	}
}
```

```
Creating an event from EventEnumeration
------------------------------------

package example

import com.adaptc.mws.plugins.EventEnumeration
import com.adaptc.mws.plugins.IPluginEventService.Severity
import com.adaptc.mws.plugins.IPluginEventService.EscalationLevel
import com.adaptc.mws.plugins.IPluginEventService.AssociatedObject

public class ExamplePlugin {
       def pluginEventService

       public void poll() {
               // Messages are pulled for messages.properties file(s) and the arguments are used
       def args = ["arg1", "arg2"]
               pluginEventService.createEvent(MyEvents.EVENT_INFO, args, [[type:"type1", id:"id1"]])
               pluginEventService.createEvent(MyEvents.EVENT_WARN, args, [[type:"type2", id:"id2"]])
       }
}

@EventEnumeration
enum MyEvents {
       EVENT_INFO("Information", INFO, USER),
       EVENT_ERROR("Warning", WARN, USER)

       static final String EVENT_TYPE_PREFIX = "Example Plugin"
       static final String ORIGIN_SUFFIX = "poll"
}
```

```
Create or update a notification
------------------------------------

package example

import com.adaptc.mws.plugins.IPluginEventService.EscalationLevel
import com.adaptc.mws.plugins.IPluginEventService.AssociatedObject

public class ExamplePlugin {
       def pluginEventService

       public void poll() {
               pluginEventService.updateNotification(EscalationLevel.POWER_USER, "There is an error
with node1",
                                       // If non-null, this must always be an associated object, never
                                       new AssociatedObject(id:"node1", type:"node"), null)
       }
}
```

Related topics

- **Events** on page 1506
- **Notifications** on page 1563
- **Notification conditions** on page 1558
- **Plugin services** on page 1725
- **Plugin developer's guide** on page 1657
- **Fields: Events** on page 1837

- **Resources introduction** on page 1424
- **Creating events and notifications** on page 1682

SSL service

The SSL service may be used to manage and load certificates or keys from disk and create socket connections. See **Managing SSL connections** on page 1692 for more information.

The `sslService` property will be injected with a class of type ISslService in all plugin types.

Related topics

- **Plugin services** on page 1725

Storage RM service

The storage RM service may be used to report storage state data to Moab Workload Manager through the RM interface. See **Reporting state data** on page 1676 for more information. It may also be used to retrieve previous reports made by a plugin. Please note that due to data consolidation (see **Data consolidation** on page 1655), old storage reports may no longer exist in the database by the time the query is done.

The `storageRMService` property will be injected with a class of type IStorageRMService in all plugin types. Note that it is not available for injection in translators or custom components.

Related topics

- **Reporting state data** on page 1676
- **Plugin services** on page 1725

Virtual machine RM service

The virtual machine RM service may be used to report virtual machine state data to Moab Workload Manager through the RM interface. See **Reporting state data** on page 1676 for more information. It may also be used to retrieve previous reports made by a plugin. Please note that due to data consolidation **Data consolidation** on page 1655, old virtual machine reports may no longer exist in the database by the time the query is done.

The `virtualMachineRMService` property will be injected with a class of type IVirtualMachineRMService in all plugin types. Note that it is not available for injection in translators or custom components.

Related topics

- **Plugin services** on page 1725

Plugin types

Power Management Plugin

The Power Management plugin is used as a resource manager to Moab to report and manipulate the power state (On or Off) for each node. Moab considers nodes in the power state On or Off; however, through TORQUE and scripts, we are able to separate the Off state into those controlled through the operating system (Standby, Suspend, Hibernate, Shutdown) and those controlled through hardware (Off). This plugin provides an easy way to integrate with Moab to translate Moab's Off action into the desired TORQUE or script action for each node. A cluster will have multiple instances of this plugin when it has varied hardware integration and/or credentials.

Creating a Power Management Plugin

To create a Power Mangement plugin, see **Creating a plugin** on page 1720. During plugin creation, refer to the **Configuration** on page 1736 section.

Configuration

Configuration Parameters

Name	Key	Required	Type	Description
Node Configuration File	nodeConfigurationFile	Yes	String	File containing list of nodes that use the scripts and credentials in this plugin instance. This is also the file to configure a particular node's off state, or an off state that will override the default off state for this instance.
Username File	usernameFile	Yes	String	File containing username issued to the scripts with the -u option.
Password File	passwordFile	Yes	String	File containing password issued to the scripts with the -p option.
Node Power Script	nodePowerScript	Yes	String	Script that powers on and off nodes and wakes them from a low power state.
Node Query Script	nodeQueryScript	Yes	String	Script that queries power state using an intelligent platform management interface.

Name	Key	Required	Type	Description
Default Power Off State	defaultPowerOffState	Yes	String	Actual state (Standby, Suspend, Hibernate, Shutdown, or Off) nodes will go into when Moab powers them off.
Max Threads	maxThreads	Yes	Integer	Thread count issued to the scripts with the -t option (defaults to 4).

Plugin Management

For information on managing the IPMI plugin, including stopping it, starting it, and checking on its status, see the **Plugin management** on page 1720 section of the MWS Guide.

Web Services

Node Power (Secured)

Resource URLs

Resource
`/rest/plugins/<pluginId>/services/nodePower`
`/rest/plugins/<pluginId>/services/node-power`

URL Parameters

Parameter	Name	Type	Description
nodes	Moab Nodes	String	A comma-delimited list of Moab node names. It is required.
power	The Power State	String	The power command Moab issues the node (On or Off).

Response Fields

Field	Name	Type	Description
success	Success Indicator	Boolean	True if the power script and/or **pbsnodes** on page 2356 was successful, otherwise false.

Field	Name	Type	Description
messages	Messages	List of Strings	Only present when the request was not successful or the node was not configured with the plugin instance. Contains error messages describing why the pbsnodes or the power script failed.

Additional Information

This web service was intended for Moab's use only and is exposed for debugging and testing your customized scripts.

Reload Node Configuration (Secured)

Resource URLs

Resource
/rest/plugins/<pluginId>/services/reloadNodeConfiguration
/rest/plugins/<pluginId>/services/reload-node-configuration

URL Parameters

Parameter	Name	Type	Description
No URL parameters required			

Response Fields

Field	Name	Type	Description
success	Success Indicator	Boolean	True if the reload succeeded, otherwise false.
messages	Messages	List of Strings	Only present when the request failed. Contains error messages describing why the reload failed.

Additional Information

The reloadNodeConfiguration web service must be run after any change to the node configuration file for it to take effect.

Node Configuration File

The node configuration file is used when the plugin is first instantiated or the `reloadNodeConfiguration` web service is called. The plugin expects a file that is readable by the tomcat user and has a Moab node name on each line. If the user would like to override the default power-off state of the node, then the node name is followed by a space and the state. For example, a node configuration file might look like this:

```
node01.ac
node02.ac
node03.ac Hibernate
node04.ac Suspend
```

The valid power-off states include Standby, Suspend, Hibernate, Shutdown, and Off. If no power-off state is provided for the node in the configuration file, then the default power-off state will be used.

The Node Power and Query Script

The plugin uses the power script to power on nodes from all power states and to power off nodes only into the Off power state. The plugin uses the power state of the node to decide whether to power on the node with `wake` or `on`. If the node is in Standby or Suspend, the plugin will call the script with the `wake` parameter. If the node is in Hibernate, Shutdown, or Off, the plugin will call the script with the `on` parameter. The plugin calls the power node script with the `off` parameter to put the node in the Off state (it uses TORQUE to put the node in the Standby, Suspend, Hibernate, and Shutdown state).

The plugin uses the query script to know if a node is in the Off power state. If the query script reports the node as Off, the plugin will report the node as Off to Moab. If the query script reports the node as On, the plugin will look to TORQUE to make sure the node is in a Running power state before it reports it as On.

The plugin passes the `usernameFile`, `passwordFile`, and `maxThreads` configuration parameters down to the scripts. The node power script is called with this syntax:

```
<nodePowerScript> -u <usernameFile> -p <passwordFile> -t <maxThreads> node01 node02
node03 ... <on|off|wake>
```

The node query script is called with this syntax:

```
<nodeQueryScript> -u <usernameFile> -p <passwordFile> -t <maxThreads> node01 node02
node03 ...
```

The plugin expects the scripts to print JSON to standard out. An example query script output would look like this:

```
[
    {
        "name": "node01.ac",
        "power": "ON",
        "Processor_2_Temp": 61,
        "Processor_1_Temp": 54
    },
    ...
]
```

Notice it is a list of nodes where each node has the required fields `name` and `power`. All the other key-value pairs will be reported to Moab as a generic resource as long as the value is a number.

The output for the node power plugin is not required; however, the output is read to give the user a detailed error message if needed. For both the node power and query scripts, if the field `error` exists, the plugin will log an error with all the strings in the list. An example error returned to the plugin would look like this:

```
[
    {
        "command": "ipmitool -I lan -H node01i -U admin -f /opt/moab/etc/power-
management/abc-plugin-password-file sdr type temperature",
        "name": "node01.ac",
        "error": [
            "big error"
        ]
    }
    ...
]
```

Troubleshooting

The Power Management plugin logs all errors and warnings to the MWS log file, which is `/opt/mws/log/mws.log` by default. The stacktrace.log file, in the same directory as `mws.log`, can also be helpful in diagnosing problems. If your MWS supports notifications, they are also helpful in diagnosing the error states the plugin is in, if any. Just check for notifications from the `PowerManagement` plugin type and the instance that you are interested in. When the issue has been resolved, you can dismiss the notification. For more information, see the Notification and Notification Condition Resource in the MWS documentation.

Set the appropriate MWS RM precedence

The Create/Edit Plugin pages give the option to set the precedence of the Moab RM plugin. The purpose of the Power Management Plugin is to report node power; however, if the precedence is too low another Moab RM plugin with a higher precedence and conflicting node might overwrite the node power. To check what MWS is reporting to Moab, go to the URL:

```
http://<MWS host>:8080/mws/rest/plugins/all/rm/cluster-query[?api-version=3]
```

To check what your plugin instance is reporting to Moab, use the URL:

```
http://<MWS host>:8080/mws/rest/plugins/<instance-name>/rm/cluster-query[?api-
version=3]
```

If the power is reported in your instance but not to Moab, please increase the precedence of the Moab RM plugin.

Configure the MWS RM in Moab

First, the following lines must be in the Moab Workload Manager configuration file or one of its included files:

```
RMCFG[mws]                      TYPE=MWS
RMCFG[mws]                      FLAGS=UserSpaceIsSeparate
RMCFG[mws]                      BASEURL=http://<mws host>:8080/mws
```

Next, edit the MWS credential information in the Moab private configuration file (`/opt/moab/etc/moab-private.cfg`, by default). Here are the default values:

```
CLIENTCFG[RM:mws] USERNAME=moab-admin PASSWORD=changeme!
```

For more information see the Resource manager queries section in the MWS documentation.

Configure TORQUE with tomcat administrator

The plugin assumes that TORQUE is installed on the same host as MWS and that tomcat is an administrator. This can be verified with **qmgr** on page 2377. Run the command:

```
qmgr -c 's s managers += tomcat@<mws_host>'
```

For more information see the Specifying non-root administrators section of the TORQUE documentation.

Make sure the Node and Power scripts work first.

The default scripts are included in `/opt/moab/tools/mws/power_management` and have their own documentation with the `-h` option. They need to have a file that maps each node in the Moab cluster to the IPMI address that the script will need to call using ipmitool. It also needs a file that includes the IPMI password. After that is provided and ipmitool is installed and working, the scripts will successfully implement the interface needed for this plugin.

Related topics

- pbsnodes -m
- Green computing overview

References

Client code samples

The code samples contained in this section of the reference material are provided to help quick start integration with MWS. They are provided as a convenience and not as fully developed APIs.

All examples use the default configuration of MWS, including the default username and password, and assume that MWSis deployed at `http://localhost:8080/mws`.

This section contains these topics:

- **Perl code samples** on page 1747
- **Python code examples** on page 1749
- **cURL samples** on page 1749

Related topics

- **Configuration** on page 1750

Javascript code samples

When utilizing Javascript to interact with MWS, it is recommended to use libraries that provide a simple browser-independent syntax for performing REST calls. It must also be noted that it is not recommended to make calls to MWS directly from client- side Javascript, as this will contain the username and password for MWS and could potentially be retrieved and utilized by a malicious user.

Examples for specific libraries are given below.

jQuery

The jQuery `ajax` function makes it extremely simple to perform all REST calls. There are also some shorthand methods which utilize the `ajax` method in a simple manner—such as `getJSON` and `post`—but these do not have the ability to specify authentication parameters, which is why they are not used below.

In all examples given, it is assumed that the `$` variable is mapped to `jQuery`, which is the default configuration of jQuery.

GET

```
$.ajax({
  url: "http://localhost:8080/mws/rest/jobs",
  dataType: 'json',
  username: 'admin',
  password: 'secret',
  success: new function(data) {
    $('.result').html(data);
    alert("GET was successful");
  )
});
```

POST

```
$.ajax({
  url: "http://localhost:8080/mws/rest/jobs",
  data: {
    "commandFile":"/tmp/test.sh",
    "initialWorkingDirectory":"/tmp",
    "user":"adaptive",
    "requirements":[{"requiredNodeCountMinimum":1}]
  }
  dataType: 'json',
  username: 'admin',
  password: 'secret',
  success: new function(data) {
    $('.result').html(data);
    alert("GET was successful");
  )
});
```

PUT

```
$.ajax({
  type: 'PUT',
  url: 'http://localhost:8080/mws/rest/jobs/Moab.1',
  data: {"holds":["user"]},
  dataType: "json",
  username: 'admin',
  password: 'secret',
  success: function(data) {
    $('.result').html(data);
    alert("PUT was successful");
  }
});
```

DELETE

```
$.ajax({
  type: 'DELETE',
  url: 'http://localhost:8080/mws/rest/jobs/Moab.1',
  username: 'admin',
  password: 'secret',
  success: function(data) {
    $('.result').html(data);
    alert("DELETE was successful");
  }
});
```

Related topics

- **Client code samples** on page 1741

PHP code samples

All of these examples make use of the cURL PHP Extension. While this library is easy to use for GET and POST requests, it is somewhat more difficult to use for PUT and DELETE requests. Each will be covered in this topic.

Notice the use of the JSON PHP Extension's `json_decode` and `json_encode` functions.

GET

```
<?php
$baseUrl = "http://localhost:8080/mws/rest";
$resource = "/jobs";
$username = "admin";
$password = "secret";
$ch = curl_init();
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
curl_setopt($ch, CURLOPT_TIMEOUT, 10);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_USERPWD, "$username:$password");
curl_setopt($ch, CURLOPT_URL, "$baseUrl$resource");

$responseBody = curl_exec($ch);
$responseInfo = curl_getinfo($ch);
curl_close($ch);

if ($responseInfo["http_code"]!=200 && $responseInfo["http_code"]!=201) {
  print_r($responseInfo);
  echo $responseBody;
} else {
  print_r(json_decode($responseBody));
}
?>
```

POST

```
<?php
$baseUrl = "http://localhost:8080/mws/rest";
$resource = "/jobs";
$username = "admin";
$password = "secret";
$requestPayload = array(
  "commandFile"=>"/tmp/test.sh",
  "initialWorkingDirectory"=>"/tmp",
  "user"=>"adaptive",
  "requirements"=>array(
      array("requiredNodeCountMinimum"=>1)
  )
);
$ch = curl_init();
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
curl_setopt($ch, CURLOPT_TIMEOUT, 10);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, array("Content-Type: application/json"));
curl_setopt($ch, CURLOPT_USERPWD, "$username:$password");
curl_setopt($ch, CURLOPT_URL, "$baseUrl$resource");
// Setup POST request
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($requestPayload));
curl_setopt($ch, CURLOPT_POST, 1);

$responseBody = curl_exec($ch);
$responseInfo = curl_getinfo($ch);
curl_close($ch);

if ($responseInfo["http_code"]!=200 && $responseInfo["http_code"]!=201) {
  print_r($responseInfo);
  echo $responseBody;
} else {
  print_r(json_decode($responseBody));
}
?>
```

PUT

```
<?php
$baseUrl = "http://localhost:8080/mws/rest";
$resource = "/jobs/Moab.1";
$username = "admin";
$password = "secret";
$requestPayload = array(
  "holds"=>array("user")
);
$ch = curl_init();
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
curl_setopt($ch, CURLOPT_TIMEOUT, 10);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, array("Content-Type: application/json"));
curl_setopt($ch, CURLOPT_USERPWD, "$username:$password");
curl_setopt($ch, CURLOPT_URL, "$baseUrl$resource");
// Setup PUT request
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($requestPayload));
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "PUT");

$responseBody = curl_exec($ch);
$responseInfo = curl_getinfo($ch);
curl_close($ch);

if ($responseInfo["http_code"]!=200 && $responseInfo["http_code"]!=201) {
  print_r($responseInfo);
  echo $responseBody;
} else {
  print_r(json_decode($responseBody));
}
```

DELETE

```
<?php
$baseUrl = "http://localhost:8080/mws/rest";
$resource = "/jobs/Moab.1";
$username = "admin";
$password = "secret";
$requestPayload = array(
  "holds"=>array("user")
);
$ch = curl_init();
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
curl_setopt($ch, CURLOPT_TIMEOUT, 10);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, array("Content-Type: application/json"));
curl_setopt($ch, CURLOPT_USERPWD, "$username:$password");
curl_setopt($ch, CURLOPT_URL, "$baseUrl$resource");
// Setup DELETE request
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "DELETE");

$responseBody = curl_exec($ch);
$responseInfo = curl_getinfo($ch);
curl_close($ch);

if ($responseInfo["http_code"]!=200 && $responseInfo["http_code"]!=201) {
  print_r($responseInfo);
  echo $responseBody;
} else {
  print_r(json_decode($responseBody));
}
```

Related topics

- **Client code samples** on page 1741

Perl code samples

These examples all utilize the `LWP::UserAgent` module, which must be installed before running them.

GET

```
#!/usr/bin/perl -w
use strict;
use warnings;
# Create a user agent object
use LWP::UserAgent;
my $ua = LWP::UserAgent->new;
$ua->agent("MyApp/0.1");

# Create a request
my $req = HTTP::Request->new(GET => 'http://localhost:8080/mws/rest/images');
$req->content_type('application/json');
$req->authorization_basic("admin", "secret");

# Pass request to the user agent and get a response back
my $res = $ua->request($req);

# Check the outcome of the response
if ($res->is_success) {
  print $res->content;
} else {
  print $res->status_line, "n";
}
```

POST

```
#!/usr/bin/perl -w
use strict;
use warnings;
# Create a user agent object
use LWP::UserAgent;
my $ua = LWP::UserAgent->new;
$ua->agent("MyApp/0.1");

# Create a request
my $req = HTTP::Request->new(POST => 'http://localhost:8080/mws/rest/images');
$req->content_type('application/json');
$req->authorization_basic("admin", "secret");
$req->content('
{"profile":"compute","osVersion":"5","name":"centos5stateless","hypervisor":0,"archite
cture":"x86_64","osName":"centos","osType":"linux","type":"stateless"}');

# Pass request to the user agent and get a response back
my $res = $ua->request($req);

# Check the outcome of the response
if ($res->is_success) {
  print $res->content;
} else {
  print $res->status_line, "n";
}
```

PUT

```
#!/usr/bin/perl -w
use strict;
use warnings;
# Create a user agent object
use LWP::UserAgent;
my $ua = LWP::UserAgent->new;
$ua->agent("MyApp/0.1");

# Create a request
my $req = HTTP::Request->new(PUT => 'http://localhost:8080/mws/rest/images/centos5-
stateless');
$req->content_type('application/json');
$req->authorization_basic("admin", "secret");
$req->content('{"osVersion":"5.5"}');

# Pass request to the user agent and get a response back
my $res = $ua->request($req);

# Check the outcome of the response
if ($res->is_success) {
  print $res->content;
} else {
  print $res->status_line, "n";
}
```

DELETE

```
#!/usr/bin/perl -w
use strict;
use warnings;
# Create a user agent object
use LWP::UserAgent;
my $ua = LWP::UserAgent->new;
$ua->agent("MyApp/0.1");

# Create a request
my $req = HTTP::Request->new(DELETE => 'http://localhost:8080/mws/rest/images/centos5-
stateless');
$req->content_type('application/json');
$req->authorization_basic("admin", "secret");

# Pass request to the user agent and get a response back
my $res = $ua->request($req);

# Check the outcome of the response
if ($res->is_success) {
  print $res->content;
} else {
  print $res->status_line, "n";
}
```

Related topics

- **Client code samples** on page 1741

Python code examples

Notice the use of the json module to build a Python object from the return JSON data. If you want, you an also use `json.dumps` to create a JSON string from a Python object.

Simple request (GET)

```
import httplib
import base64
import string
import json
def get(base, port, url):
       conn = httplib.HTTPConnection(base, port, timeout=60)
       conn.request('GET', url, None, { 'Authorization' : 'Basic '+string.strip
(base64.encodestring('admin:secret'))})
       return conn.getresponse().read()

data = get("localhost", 8080, "/mws/rest/jobs?format=json")
print json.loads(data)
```

Complex request (POST)

```
import httplib
import base64
import string
import json
def post(base, port, url, payload):
       conn = httplib.HTTPConnection(base, port, timeout=60)
       conn.request('POST', url, payload, { 'Authorization' : 'Basic '+string.strip
(base64.encodestring('admin:secret')), 'Content-Type' : 'application/json' })
       r = conn.getresponse()
       return r.read()

# Note that json.dumps may also be used to create the json string from a python object
data = post("localhost", 8080, "/mws/rest/jobs", '
{"commandFile":"/tmp/test.sh","initialWorkingDirectory":"/tmp","user":"adaptive","requ
irements":[{"requiredNodeCountMinimum":1}]}')
print json.loads(data)
```

Related topics

- **Client code samples** on page 1741

cURL samples

Unlike the other code samples given in this section, these samples are simple commands that can be run from any server command line with the `curl` program installed in order to communicate with MWS.

GET

```
curl -u admin:secret -X GET -H "Content-Type: application/json"
http://localhost:8080/mws/rest/jobs
```

POST

```
curl -u admin:secret -X POST -H "Content-Type: application/json"
http://localhost:8080/mws/rest/jobs
  -d '
{"commandFile":"/tmp/test.sh","initialWorkingDirectory":"/tmp","user":"adaptive","requ
irements":[{"requiredNodeCountMinimum":1}]}'
```

PUT

```
curl -u admin:secret -X PUT -H "Content-Type: application/json"
http://localhost:8080/mws/rest/jobs
  -d '{"holds":["user"]}'
```

DELETE

```
curl -u admin:secret -X DELETE -H "Content-Type: application/json"
http://localhost:8080/mws/rest/jobs
```

Related topics

- **Client code samples** on page 1741

Configuration

These properties can be modified by setting the appropriate values in the `mws-config.groovy` file. This file is located in `MWS_HOME/etc/` or `/opt/mws/etc/` by default as explained in **Configuring Moab Web Services** on page 1373.

> The configuration file is read not only on startup, but also each time it is changed. Several properties, including those for Moab Workload Manager (`moab`), Moab Accounting Manager (`mam`), Mongo (`grails.mongo`), and authentication (`auth`) are processed after each change and can affect the runtime behavior of MWS.
>
> Configuration files can also be placed in the `MWS_HOME/etc/mws.d` directory. Any configuration files here get merged with `MWS_HOME/etc/mws-config.groovy`. In case of conflict, the configuration in `MWS_HOME/etc/mws.d` takes precedence.

Configuration reference

For all possible values that can be set, please see the Grails reference guide. For project specific settings (usually the only ones you'll need to change), you may set the following properties:

Property	Type	Default	Description
auth.defaultUser.password	String	changeme!	Unencoded password of the default admin user.

Property	Type	Default	Description
auth.defaultUser.username	String	moab-admin	Username of the default admin user (only created if no other users exist).
grails.mongo.host	String	127.0.0.1	The MongoDB host to use (Note that MongoDB runs on `127.0.0.1` and *not* `localhost` by default).
dataSource_insight.password	String	changeme!	The password for the username used to log in to the Insight database.
dataSource_insight.url	String	jdbc:postgresql://127.0.0.1:5432/n insight	The JDBC URL for the Insight database. For more information, see **Configuring Moab Web Services** on page 1373.
dataSource_insight.username	String	mws	The username used to log into the Insight database.
grails.mongo.port	Integer	27017	The MongoDB port to use.
grails.mongo.replicaSet	List of Strings	n/a	The MongoDB replica set servers to use (for example, `["moab1:27017","moab2:27017"]`); note that `grails.mongo.host` *must* be set to `null` to use this option.
grails.mongo.databaseName	String	mws	The MongoDB database name to use.

Property	Type	Default	Description
grails.mongo.username	String	-	(Optional) The user-name to use when con-necting to MongoDB.
grails.mongo.password	String	-	(Optional) The pass-word to use when con-necting to MongoDB.
grails.mongo.options.connectionsPerHost	Integer	50	The number of con-nections allowed per host.
grails.-mongo.-options.threadsAllowedToBlockForConnectio	Integer	5	The number of threads per con-nection allowed to wait for an available con-nection.
grails.mongo.options.autoConnectRetry	Boolean	true	Controls whether the system retries auto-matically on con-nection errors.
grails.mime.use.accept.header	Boolean	false	When enabled, uses the HTTP Content-Accept header to determine the content type used for return data (JSON only for now).
grails.plu-gins.springsecurity.basic.realmName	String	Moab Web Services	The HTTP realm used when using basic auth.
grails.plugins.springsecurity.active	Boolean	true	Enables or disables security for MWS as a whole, including all providers.
grails.plugins.springsecurity.useBasicAuth	Boolean	true	Enables or disables basic auth with a simple user-name/password.

Property	Type	Default	Description
grails.plugins.springsecurity.oauthProvider.active	Boolean	true	Enables or disables the OAuth2 provider.
ldap.baseDNs	List of Strings	-	A list of distinguished names that are the root entries for LDAP searches.
ldap.bindUser	String	-	The distinguished name of the LDAP bind user.
ldap.directory.type	String	-	The type of LDAP directory (for example, "Microsoft Active Directory"). See **Configuring Moab Web Services** on page 1373 for valid values..
ldap.password	String	-	The password of the LDAP bind user
ldap.port	Integer	-	LDAP server's port
ldap.security.server.certificate	String	-	The filename of the LDAP server's PEM encoded X.509 certificate. See **Setting up MWS security** on page 1388 for more information.
ldap.security.type	String	-	How the connection between MWS and LDAP is secured. See **Setting up MWS security** on page 1388 for more information.
ldap.server	String	-	LDAP server hostname or IP address

Property	Type	Default	Description
mam.server	String	localhost	Moab Accounting Manager server hostname or IP address
mam.port	Integer	7112	Moab Accounting Manager server's port
mam.secretKey	String	mamsecret	Secret key used to communicate with Moab Accounting Manager
moab.databaseName	String	moab	The name of the MongoDB database to use to retrieve current Moab data; this should match the database setting in Moab.
moab.messageQueue.port	Integer	5570	The port on which Moab publishes ZeroMQ messages.
moab.messageQueue.secretKey	String	-	Used to encrypt and decrypt messages on the message queue using AES. Must be a Base64-enoded 128-bit (16-byte) key. For example: `"1r6RvfqJa6voezy5wAx0hw=="`
moab.port	Integer	42559	Moab server's port
moab.secretKey	String	moabsecret	Secret key used to communicate with Moab. See Moab Configuration.
moab.server	String	localhost	Moab server hostname or IP address

Property	Type	Default	Description
mws.cache.duration.default	Integer	60	The default number of seconds to use for caching objects from Moab. This is only supported in certain objects such as policies.
mws.cache.duration.policy	Integer	180	The number of seconds that the cache for policies is valid. If set to null, the default is used.
mws.certificates.location	String	etc/ssl.crt	The directory (relative or absolute) where plugin certificates are stored. See the **Managing SSL connections** on page 1692.
mws.events.expireAfterSeconds	Integer	2592000	Events older than this many seconds (30 days by default) will be deleted from the database. Effective only with MongoDB 2.2 or later.
mws.health.check.period	Integer	30	The number of seconds in between health checks. Used in creating notification conditions if problems exist in configuration or connections. For more information, see **Notification conditions** on page 1558.

Property	Type	Default	Description
mws.hooks.location	String	hooks	The directory (relative or absolute) where Hooks are stored. See **Pre and post-processing hooks** on page 1412 for more information.
mws.plugins.location	String	plugins	The directory (relative or absolute) where Plugins are stored. See **About Moab Web Services plugins** on page 1650 for more information.
mws.messageQueue.port	Integer	5564	The port on which MWS publishes ZeroMQ messages.
mws.messageQueue.address	String	-	The IP address on which MWS publishes ZeroMQ messages.
mws.services.hooks.syncInterval	Integer	30	The number of seconds between each time MWS checks for service phase transition hooks that completed or timed out.
mws.services.phases.syncInterval	Integer	14400	The number of seconds between each time MWS checks with Moab Workload Manager to verify that the service phases are correctly synchronized.
mws.suite	String	CLOUD	The suite or context that MWS is running in (see Suite for valid values)

Property	Type	Default	Description
pam.configuration.service	String	-	The name of the PAM configuration file located in `/etc/pam.d`. This parameter and specification tells MWS which PAM configuration file you want to use. For more information, see **PAM (pluggable authentication module) configuration using mws-config.groovy** on page 1383.
plugins.pluginType	String	-	Default configuration value for the plugin `pluginType` field (see **Setting default plugin configuration** on page 1725).
plugins.autoStart	Boolean	true	Default configuration value for the plugin `autoStart` field (see **Setting default plugin configuration** on page 1725).
plugins.pollInterval	Integer	30	Default configuration value for the plugin `pollInterval` field (see **Setting default plugin configuration** on page 1725).
plugins.config	Map	-	Default configuration value for the plugin `config` field (see **Setting default plugin configuration** on page 1725).

Property	Type	Default	Description
plugins.loadInitialPlugins	Boolean	true	If true, loads the initial plugins defined for uploaded or built-in plugin types (see **Plugin projects and metadata** on page 1699).
plugins.stateConsolidationPolicy	NodeStatePolicy	null	If "optimistic", treats state data optimistically. If "pessimistic", treats state state pessimistically. May be null. See **Data consolidation** on page 1655 for more information.
plugins.defaultHypervisorType	String	ESX	This is reported to Moab when a node report references a hypervisor image that does not have the `hypervisorType` or `extensions.xcat.hvType` fields set. See **Fields: Images** on page 1843.

Logging reference

The following loggers are available to use for debugging purposes:

Logger	Default	Description
grails.app	debug	Most classes in the main MWS application.
grails.app.bootstrap.BootStrap	debug	Handles startup and initialization of MWS.

Logger	Default	Description
com.ace.mws	debug	The base logger for MWS specific functionality not included in other loggers (this comprises very few classes).
grails.app.services.com.ace.mws.plugins.PluginUtilityService	debug	Class for initializing and helper methods of plugins.
com.ace.mws.hooks.HookUtils	debug	Helper class for loading hooks during startup process.
plugins	debug	All MWS plugins (see **About Moab Web Services plugins** on page 1650).
com.ace.mws.plugins	debug	MWS plugin helper class, used to create and initialize plugins.
com.ace.mws.gapi	warn	Base logger for all Moab connections, requests, and responses.
com.ace.mws.gapi.Connection	info	Logger which controls all requests and responses from Moab.
com.ace.mws.gapi.parsers	info	Loggers for parsers of Moab's data.
com.ace.mws.gapi.serializers	info	Loggers for all serialization from MWS to Moab Wire Protocol.
grails.app.service.grails.plugins.reloadconfig	info	Handles dynamic reloading of configuration files.
net.sf.json	error	JSON and XML processing library.
org.springframework.security	info	Authentication/authorization logger.

Logger	Default	Description
org.codehaus.groovy.grails.web.servlet	error	Loggers for request handlers.
org.codehaus.groovy.grails.web.mapping	error	URL mapping.
org.codehaus.groovy.grails.web.mapping.filter	error	URL mapping.
org.codehaus.groovy.grails.plugins	error	All grails plugins (MWS internal).
org.codehaus.groovy.grails.commons	error	Core application and class-loading.

Related topics

- **Configuring Moab Web Services** on page 1373

Resource reference

Resources reference

This section contains the type and description of all possible fields in each MWS resource object. Because of significant changes in the API introduced between releases, MWS possesses a versioned API. Each resource contains drop-down sections for each API version.

This section contains these topics:

- **Fields: Job Templates** on page 1974
- **Fields: Jobs** on page 1914
- **Fields: Liens** on page 1815
- **Fields: Metric Types** on page 2003
- **Fields: Nodes** on page 2004
- **Fields: Notification Conditions** on page 2024
- **Fields: Notifications** on page 2028
- **Fields: Organizations** on page 1819
- **Fields: User's Permissions** on page 2186
- **Fields: Plugins** on page 2030
- **Fields: Plugin Types** on page 2035
- **Fields: Policies** on page 2039
- **Fields: Principals** on page 2065
- **Fields: Quotes** on page 1821
- **Fields: Report Datapoints** on page 2072
- **Fields: Report Samples** on page 2127
- **Fields: Reports** on page 2074
- **Fields: Reservations** on page 2080
- **Fields: Resource Types** on page 2120
- **Fields: Roles** on page 2121
- **Fields: Standing Reservations** on page 2129
- **Fields: Transactions** on page 1826
- **Fields: Usage Records** on page 1830
- **Fields: Users** on page 1834

Related topics

- **Resources introduction** on page 1424
- **Global URL parameters** on page 1403

Fields: Access Control Lists (ACLs)

See the associated **Access control lists (ACLs)** on page 1426 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`acl-rules`	**Permissions** on page 1571
Hooks filename	`acl-rules.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

AclRule

This class represents a rule that can be in Moab's access control list (ACL) mechanism.

The basic AclRule information is the object's name and type. The type directly maps to an AclType value. The default mechanism Moab uses to check the ACL for a particular item is if the user or object coming in has ANY of the values in the ACL, then the user or object is given access. If no values match the user or object in question, the user or object is rejected access.

Field Name	Type	PUT	Description
affinity	**AclAffinity**	Yes	Reservation ACLs allow or deny access to reserved resources but they may also be configured to affect a job's affinity for a particular reservation. By default, jobs gravitate toward reservations through a mechanism known as positive affinity. This mechanism allows jobs to run on the most constrained resources leaving other, unreserved resources free for use by other jobs that may not be able to access the reserved resources. Normally this is a desired behavior. However, sometimes, it is desirable to reserve resources for use only as a last resort-using the reserved resources only when there are no other resources available. This last resort behavior is known as negative affinity. Defaults to AclAffinity.POSITIVE.
comparator	**ComparisonOperator**	Yes	The type of comparison to make against the ACL object. Defaults to ComparisonOperator.EQUAL.
type	**AclType**	Yes	The type of the object that is being granted (or denied) access.
value	**String**	Yes	The name of the object that is being granted (or denied) access.

AclAffinity

This enumeration describes the values available for describing how a rule is used in establishing access to an object in Moab. Currently, these ACL affinities are used only for granting access to reservations.

Value	Description
NEGATIVE	Access to the object is repelled using this rule until access is the last choice.
NEUTRAL	Access to the object is not affected by affinity.
POSITIVE	Access to the object is looked at as the first choice.
PREEMPTIBLE	Access to the object given the rule gives preemptible status to the accessor. Supported only during GET.
REQUIRED	The rule in question must be satisified in order to gain access to the object. Supported only during GET.
UNAVAILABLE	The rule does not have its affinity available. Supported only during GET.

ComparisonOperator

This enumeration is used when Moab needs to compare items. One such use is in Access Control Lists (ACLs).

Value	Description
GREATER_THAN	Valid values: ">", "gt"
GREATER_THAN_OR_EQUAL	Valid values: ">=", "ge"
LESS_THAN	Valid values: "<", "lt"
LESS_THAN_OR_EQUAL	Valid values: "<=", "le"
EQUAL	Valid values: "==", "eq", "="
NOT_EQUAL	Valid values: "!=", "ne", "<>"
LEXIGRAPHIC_SUBSTRING	Valid value: "%<"
LEXIGRAPHIC_NOT_EQUAL	Valid value: "%!"
LEXIGRAPHIC_EQUAL	Valid value: "%="

AclType

This enumeration describes the values available for the type of an ACL Rule.

Value	Description
USER	User
GROUP	Group
ACCOUNT	Account or Project
CLASS	Class or Queue
QOS	Quality of Service
CLUSTER	Cluster
JOB_ID	Job ID
RESERVATION_ID	Reservation ID
JOB_TEMPLATE	Job Template
JOB_ATTRIBUTE	Job Attribute
DURATION	Duration in Seconds
PROCESSOR_SECONDS	Processor Seconds
JPRIORITY	Not supported
MEMORY	Not supported
NODE	Not supported
PAR	Not supported
PROC	Not supported
QTIME	Not supported

Value	Description
QUEUE	Not supported
RACK	Not supported
SCHED	Not supported
SYTEM	Not supported
TASK	Not supported
VC	Not supported
XFACTOR	Not supported

API version 2

AclRule

This class represents a rule that can be in Moab's access control list (ACL) mechanism.

The basic AclRule information is the object's name and type. The type directly maps to an AclType value. The default mechanism Moab uses to check the ACL for a particular item is if the user or object coming in has ANY of the values in the ACL, then the user or object is given access. If no values match the user or object in question, the user or object is rejected access.

Field Name	Type	PUT	Description
affinity	**AclAffinity**	Yes	Reservation ACLs allow or deny access to reserved resources but they may also be configured to affect a job's affinity for a particular reservation. By default, jobs gravitate toward reservations through a mechanism known as positive affinity. This mechanism allows jobs to run on the most constrained resources leaving other, unreserved resources free for use by other jobs that may not be able to access the reserved resources. Normally this is a desired behavior. However, sometimes, it is desirable to reserve resources for use only as a last resort-using the reserved resources only when there are no other resources available. This last resort behavior is known as negative affinity. Defaults to AclAffinity.POSITIVE.
comparator	**ComparisonOperator**	Yes	The type of comparison to make against the ACL object. Defaults to ComparisonOperator.EQUAL.
type	**AclType**	Yes	The type of the object that is being granted (or denied) access.
value	**String**	Yes	The name of the object that is being granted (or denied) access.

AclAffinity

This enumeration describes the values available for describing how a rule is used in establishing access to an object in Moab. Currently, these ACL affinities are used only for granting access to reservations.

Value	Description
NEGATIVE	Access to the object is repelled using this rule until access is the last choice.
NEUTRAL	Access to the object is not affected by affinity.
POSITIVE	Access to the object is looked at as the first choice.
PREEMPTIBLE	Access to the object given the rule gives preemptible status to the accessor. Supported only during GET.
REQUIRED	The rule in question must be satisified in order to gain access to the object. Supported only during GET.
UNAVAILABLE	The rule does not have its affinity available. Supported only during GET.

ComparisonOperator

This enumeration is used when Moab needs to compare items. One such use is in Access Control Lists (ACLs).

Value	Description
GREATER_THAN	Valid values: ">", "gt"
GREATER_THAN_OR_EQUAL	Valid values: ">=", "ge"
LESS_THAN	Valid values: "<", "lt"
LESS_THAN_OR_EQUAL	Valid values: "<=", "le"
EQUAL	Valid values: "==", "eq", "="
NOT_EQUAL	Valid values: "!=", "ne", "<>"
LEXIGRAPHIC_SUBSTRING	Valid value: "%<"
LEXIGRAPHIC_NOT_EQUAL	Valid value: "%!"
LEXIGRAPHIC_EQUAL	Valid value: "%="

AclType

This enumeration describes the values available for the type of an ACL Rule.

Value	Description
USER	User
GROUP	Group
ACCOUNT	Account or Project
CLASS	Class or Queue
QOS	Quality of Service
CLUSTER	Cluster
JOB_ID	Job ID
RESERVATION_ID	Reservation ID
JOB_TEMPLATE	Job Template
JOB_ATTRIBUTE	Job Attribute
DURATION	Duration in Seconds
PROCESSOR_SECONDS	Processor Seconds
JPRIORITY	Not supported
MEMORY	Not supported
NODE	Not supported
PAR	Not supported
PROC	Not supported
QTIME	Not supported

Value	Description
QUEUE	Not supported
RACK	Not supported
SCHED	Not supported
SYTEM	Not supported
TASK	Not supported
VC	Not supported
XFACTOR	Not supported

Related topics

- **Access control lists (ACLs)** on page 1426

Accounting

Fields: Accounts

See the associated **Accounting Accounts** on page 1429 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`accounting/accounts`	**Permissions** on page 1571
Hooks filename	`accounting.accounts.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

Account

Users may be designated as members of an account and may be allowed to share its allocations. The user members may be designated as active or inactive, and as an account admin or not an account admin. Default account properties include the description, the organization it is part of, and whether or not it is active. An account's user membership can also be adjusted. By default, a standard user may only query accounts they belong to.

Field Name	Type	Description
id	**String**	The unique account identifier
active	**Boolean**	A boolean indicating whether this account is active or not
creationTime	**Date**	The time this account was created
deleted	**Boolean**	A boolean indicating whether this account is deleted or not
description	**String**	The account description
modificationTime	**Date**	The time this account was last modified
organization	**String**	The organization to which the account belongs
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction
users	**Set<AccountUser>**	The users associated with this account

AccountUser

An account user is a person authorized to use an account.

Field Name	Type	Description
id	**String**	The unique user identifier
active	**Boolean**	A boolean indicating whether this user is active or not
admin	**Boolean**	A boolean indicating wheter this user is an admin or not

API version 2

Account

Users may be designated as members of an account and may be allowed to share its allocations. The user members may be designated as active or inactive, and as an account admin or not an account admin. Default account properties include the description, the organization it is part of, and whether or not it is active. An account's user membership can also be adjusted. By default, a standard user may only query accounts they belong to.

Field Name	Type	Description
id	**String**	The unique account identifier
active	**Boolean**	A boolean indicating whether this account is active or not
creationTime	**Date**	The time this account was created
deleted	**Boolean**	A boolean indicating whether this account is deleted or not
description	**String**	The account description
modificationTime	**Date**	The time this account was last modified
organization	**String**	The organization to which the account belongs
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction
users	**Set<AccountUser>**	The users associated with this account

AccountUser

An account user is a person authorized to use an account.

Field Name	Type	Description
id	**String**	The unique user identifier
active	**Boolean**	A boolean indicating whether this user is active or not
admin	**Boolean**	A boolean indicating wheter this user is an admin or not

Related topics

- **Accounting Accounts** on page 1429

Fields: Allocations

> See the associated **Accounting Allocations** on page 1433 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`accounting/allocations`	**Permissions** on page 1571
Hooks filename	`accounting.allocations.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

Allocation

An allocation is a time-bounded pool of resource or service credits associated with an fund. An fund may have multiple allocations, each for use during a different time period.

An allocation has a start time and an end time that defines the time period during which the allocation may be used. By default an allocation is created with an unbounded time period (-infinity to infinity). An active flag is automatically updated to true if the fund is within its valid timeframe or false if it is not. An allocation may also have a credit limit representing the amount by which it can go negative. Thus, by having a positive balance in the Amount field, the fund is like a debit fund, implementing a pay-first use-later model. By establishing a credit limit instead of depositing an initial balance, the fund will be like a credit fund, implementing a use-first pay-later model. These strategies can be combined by depositing some amount of funds coupled with a credit limit, implementing a form of overdraft protection where the funds will be used down to the negative of the credit limit.

Field Name	Type	Description
id	**String**	The unique identifier for this allocation
active	**Boolean**	Indicates whether this allocation is active or not
allocated	**BigDecimal**	Adjusted allocation. This value stores the effective allocated amount based on the initial deposit and subsequent allocation adjustments via deposits, withdrawals or transfers.
amount	**BigDecimal**	The amount of this allocation
creationTime	**Date**	The date this allocation was created
creditLimit	**BigDecimal**	Determines how far in the negative this allocation is permitted to be used (enforced in quotes and liens)
deleted	**Boolean**	A boolean indicating whether this allocation is deleted or not
description	**String**	The description of this allocation
endTime	**Date**	The date this allocation becomes inactive
fund	**String**	The fund Id associated with this allocation
modificationTime	**Date**	The date this allocation was last modified

Field Name	Type	Description
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The date this allocation becomes active
transactionId	**Long**	The id of the last modifying transaction

API version 2

Allocation

An allocation is a time-bounded pool of resource or service credits associated with an fund. An fund may have multiple allocations, each for use during a different time period.

An allocation has a start time and an end time that defines the time period during which the allocation may be used. By default an allocation is created with an unbounded time period (-infinity to infinity). An active flag is automatically updated to true if the fund is within its valid timeframe or false if it is not. An allocation may also have a credit limit representing the amount by which it can go negative. Thus, by having a positive balance in the Amount field, the fund is like a debit fund, implementing a pay-first use-later model. By establishing a credit limit instead of depositing an initial balance, the fund will be like a credit fund, implementing a use-first pay-later model. These strategies can be combined by depositing some amount of funds coupled with a credit limit, implementing a form of overdraft protection where the funds will be used down to the negative of the credit limit.

Field Name	Type	Description
id	**String**	The unique identifier for this allocation
active	**Boolean**	Indicates whether this allocation is active or not
allocated	**BigDecimal**	Adjusted allocation. This value stores the effective allocated amount based on the initial deposit and subsequent allocation adjustments via deposits, withdrawals or transfers.
amount	**BigDecimal**	The amount of this allocation
creationTime	**Date**	The date this allocation was created
creditLimit	**BigDecimal**	Determines how far in the negative this allocation is permitted to be used (enforced in quotes and liens)
deleted	**Boolean**	A boolean indicating whether this allocation is deleted or not
description	**String**	The description of this allocation
endTime	**Date**	The date this allocation becomes inactive
fund	**String**	The fund Id associated with this allocation
modificationTime	**Date**	The date this allocation was last modified

Field Name	Type	Description
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The date this allocation becomes active
transactionId	**Long**	The id of the last modifying transaction

Related topics

- **Accounting Allocations** on page 1433

Fields: Charge Rates

> See the associated **Accounting Charge rates** on page 1437 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`accounting/charge-rates`	**Permissions** on page 1571
Hooks filename	`accounting.charge-rates.-groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-sup-ported	No	**Distinct** on page 1504

API version 3

ChargeRate

Charge rates establish how much to charge for usage. A charge rate consists of its name, an optional value and the amount. Both name and value are primary keys and a charge rate is uniquely defined by both its name and its value. A charge rate value that is null designates the default charge rate.

Field Name	Type	Description
id	**Long**	
amount	**String**	The charge rate amount
creationTime	**Date**	The date this charge rate was created
deleted	**Boolean**	A boolean indicating whether this charge rate is deleted or not
description	**String**	The charge rate description
modificationTime	**Date**	The date this charge rate was last modified
name	**String**	The charge rate name
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction
value	**String**	The charge rate value. This will be null for default charge rates.

API version 2

ChargeRate

Charge rates establish how much to charge for usage. A charge rate consists of its name, an optional value and the amount. Both name and value are primary keys and a charge rate is uniquely defined by both its name and its value. A charge rate value that is null designates the default charge rate.

Field Name	Type	Description
id	**Long**	
amount	**String**	The charge rate amount
creationTime	**Date**	The date this charge rate was created
deleted	**Boolean**	A boolean indicating whether this charge rate is deleted or not
description	**String**	The charge rate description
modificationTime	**Date**	The date this charge rate was last modified
name	**String**	The charge rate name
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction
value	**String**	The charge rate value. This will be null for default charge rates.

Related topics

- **Accounting Charge rates** on page 1437

Fields: Fund Balances

See the associated **Accounting Funds** on page 1441 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`accounting/funds/balances`	**Permissions** on page 1571
Hooks filename	`accounting.funds.balances.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

FundBalance

Represents a report of fund balance.

Field Name	Type	Description
id	**Long**	The unique fund identifier
allocated	**BigDecimal**	The total adjusted allocations. This value is affected positively by deposits, activations and destination transfers and affected negatively by withdrawals, deactivations and source transfers that have occurred since the last reset.
allocations	**Set<Allocation>**	Allocations associated with this fund
amount	**BigDecimal**	The sum of active allocation amounts within this fund. It does not take into fund current liens.
available	**BigDecimal**	The total amount available for charging. amount - reserved + creditLimit
balance	**BigDecimal**	The allocation total not blocked by liens. amount - reserved
capacity	**BigDecimal**	The total amount allocated via deposits and credit limits. allocated + creditLimit
creationTime	**Date**	Date this fund was created
creditLimit	**BigDecimal**	The sum of active credit limits within this fund
description	**String**	The fund description
fundConstraints	**Set<FundConstraint>**	Constraints on fund usage.
modificationTime	**Date**	The date this fund was last modified
name	**String**	The name of this fund
percentRemaining	**Double**	The percentage of allocation remaining. amount * 100 / allocated

Field Name	Type	Description
percentUsed	**Double**	The percentage of allocated used. used * 100 / allocated
reserved	**BigDecimal**	The sum of active lien amounts against this fund
used	**BigDecimal**	The total amount used this allocation cycle. allocated - amount

Allocation

An allocation is a time-bounded pool of resource or service credits associated with an fund. An fund may have multiple allocations, each for use during a different time period.

An allocation has a start time and an end time that defines the time period during which the allocation may be used. By default an allocation is created with an unbounded time period (-infinity to infinity). An active flag is automatically updated to true if the fund is within its valid timeframe or false if it is not. An allocation may also have a credit limit representing the amount by which it can go negative. Thus, by having a positive balance in the Amount field, the fund is like a debit fund, implementing a pay-first use-later model. By establishing a credit limit instead of depositing an initial balance, the fund will be like a credit fund, implementing a use-first pay-later model. These strategies can be combined by depositing some amount of funds coupled with a credit limit, implementing a form of overdraft protection where the funds will be used down to the negative of the credit limit.

Field Name	Type	Description
id	**String**	The unique identifier for this allocation
active	**Boolean**	Indicates whether this allocation is active or not
allocated	**BigDecimal**	Adjusted allocation. This value stores the effective allocated amount based on the initial deposit and subsequent allocation adjustments via deposits, withdrawals or transfers.
amount	**BigDecimal**	The amount of this allocation
creationTime	**Date**	The date this allocation was created
creditLimit	**BigDecimal**	Determines how far in the negative this allocation is permitted to be used (enforced in quotes and liens)

Field Name	Type	Description
deleted	**Boolean**	A boolean indicating whether this allocation is deleted or not
description	**String**	The description of this allocation
endTime	**Date**	The date this allocation becomes inactive
fund	**String**	The fund Id associated with this allocation
modificationTime	**Date**	The date this allocation was last modified
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The date this allocation becomes active
transactionId	**Long**	The id of the last modifying transaction

FundConstraint

Constraints designate which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the encapsulated credits in a fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, etc.).

Field Name	Type	Description
id	**String**	The unique identifier of this constraint.
fund	**String**	The fund ID that this constraint is associated with.
name	**String**	The name of the constraint.
value	**String**	The value of the constraint. The constraint may be negated by the used of an exclamation point leading the value.

API version 2

FundBalance

Represents a report of fund balance.

Field Name	Type	Description
id	**Long**	The unique fund identifier
allocated	**BigDecimal**	The total adjusted allocations. This value is affected positively by deposits, activations and destination transfers and affected negatively by withdrawals, deactivations and source transfers that have occurred since the last reset.
allocations	**Set<Allocation>**	Allocations associated with this fund
amount	**BigDecimal**	The sum of active allocation amounts within this fund. It does not take into fund current liens.
available	**BigDecimal**	The total amount available for charging. amount - reserved + creditLimit
balance	**BigDecimal**	The allocation total not blocked by liens. amount - reserved
capacity	**BigDecimal**	The total amount allocated via deposits and credit limits. allocated + creditLimit
creationTime	**Date**	Date this fund was created
creditLimit	**BigDecimal**	The sum of active credit limits within this fund
description	**String**	The fund description
fundConstraints	**Set<FundConstraint>**	Constraints on fund usage.
modificationTime	**Date**	The date this fund was last modified
name	**String**	The name of this fund
percentRemaining	**Double**	The percentage of allocation remaining. amount * 100 / allocated

Field Name	Type	Description
percentUsed	**Double**	The percentage of allocated used. used * 100 / allocated
reserved	**BigDecimal**	The sum of active lien amounts against this fund
used	**BigDecimal**	The total amount used this allocation cycle. allocated - amount

Allocation

An allocation is a time-bounded pool of resource or service credits associated with an fund. An fund may have multiple allocations, each for use during a different time period.

An allocation has a start time and an end time that defines the time period during which the allocation may be used. By default an allocation is created with an unbounded time period (-infinity to infinity). An active flag is automatically updated to true if the fund is within its valid timeframe or false if it is not. An allocation may also have a credit limit representing the amount by which it can go negative. Thus, by having a positive balance in the Amount field, the fund is like a debit fund, implementing a pay-first use-later model. By establishing a credit limit instead of depositing an initial balance, the fund will be like a credit fund, implementing a use-first pay-later model. These strategies can be combined by depositing some amount of funds coupled with a credit limit, implementing a form of overdraft protection where the funds will be used down to the negative of the credit limit.

Field Name	Type	Description
id	**String**	The unique identifier for this allocation
active	**Boolean**	Indicates whether this allocation is active or not
allocated	**BigDecimal**	Adjusted allocation. This value stores the effective allocated amount based on the initial deposit and subsequent allocation adjustments via deposits, withdrawals or transfers.
amount	**BigDecimal**	The amount of this allocation
creationTime	**Date**	The date this allocation was created
creditLimit	**BigDecimal**	Determines how far in the negative this allocation is permitted to be used (enforced in quotes and liens)

Field Name	Type	Description
deleted	**Boolean**	A boolean indicating whether this allocation is deleted or not
description	**String**	The description of this allocation
endTime	**Date**	The date this allocation becomes inactive
fund	**String**	The fund Id associated with this allocation
modificationTime	**Date**	The date this allocation was last modified
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The date this allocation becomes active
transactionId	**Long**	The id of the last modifying transaction

FundConstraint

Constraints designate which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the encapsulated credits in a fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, etc.).

Field Name	Type	Description
id	**String**	The unique identifier of this constraint.
fund	**String**	The fund ID that this constraint is associated with.
name	**String**	The name of the constraint.
value	**String**	The value of the constraint. The constraint may be negated by the used of an exclamation point leading the value.

Related topics

- **Accounting Funds** on page 1441

Fields: Fund Statement Summary

> See the associated **Accounting Funds** on page 1441 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`accounting/funds/reports/statement`	**Permissions** on page 1571
Hooks filename	`accounting.funds.reports.statement.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

FundStatementSummary

An fund statement summary is related to and quite similar to the FundStatement report, but differs in the transactions field by using the FundTransactionSummary.

Field Name	Type	Description
id	**Long**	
endBalance	**BigDecimal**	The balance of the funds at the endTime of the statement
endTime	**Date**	The ending time that the statement covers
funds	**Set<Fund>**	The funds that this statement covers. Only a sub-set of the full fund fields are available from this property. This includes id, name, priority, description, and creationTime.
generationTime	**Date**	The date that the statement report was generated
startBalance	**BigDecimal**	The balance of the funds at the startTime of the statement
startTime	**Date**	The starting time that the statement covers
totalCredits	**BigDecimal**	The total number of credits that occurred during the time period that the statement covers
totalDebits	**BigDecimal**	The total number of debits that occurred during the time period that the statement covers
transactions	**Set<FundTransactionSummary>**	Summaries of the specific transactions which occurred during the time period that this statement covers.

Fund

A fund is a container for a time-bounded reference currency called credits for which the usage is restricted by constraints that define how the credits must be used. Much like with a bank, an fund is a repository for these resource or service credits which are added through deposits and debited through withdrawals and charges. Each fund has a set of constraints designating which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, Feature, etc.). Fund constraints may also be negated with an exclamation point leading the constraint value.

When credits are deposited into an fund, they are associated with a time period within which they are valid. These time-bounded pools of credits are known as allocations. (An allocation is a pool of billable units associated with an fund for use during a particular time period.) By using multiple allocations that expire in regular intervals it is possible to implement a use-it-or-lose-it policy and establish an allocation cycle.

Funds may be nested. Hierarchically nested funds may be useful for the delegation of management roles and responsibilities. Deposit shares may be established that assist to automate a trickle-down effect for funds deposited at higher level funds. Additionally, an optional overflow feature allows charges against lower level funds to trickle up the hierarchy.

Funds may have an arbitrary name which is not necessarily unique for the fund. Funds may also have a priority which will influence the order of fund selection when charging.

Field Name	Type	Description
id	**Long**	The unique fund identifier
allocated	**BigDecimal**	Total Adjusted allocations. This value is affected positively by deposits, activations and destination transfers and affected negatively by withdrawals, deactivations and source transfers that have occurred since the last reset.
allocations	**Set<Allocation>**	The allocations associated with this fund.
amount	**BigDecimal**	The sum of active allocation amounts within this fund. It does not take into fund current liens.
creationTime	**Date**	Date this fund was created
creditLimit	**BigDecimal**	The sum of active credit limits within this fund
defaultDeposit	**String**	The default deposit amount
deleted	**Boolean**	A boolean indicating whether this fund is deleted or not
description	**String**	The fund description
fundConstraints	**Set<FundConstraint>**	Constraints on fund usage.

Field Name	Type	Description
initialDeposit	**BigDecimal**	The initial deposit amount
modificationTime	**Date**	The date this fund was last modified
name	**String**	The name of this fund
priority	**Integer**	The fund priority
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction

Allocation

An allocation is a time-bounded pool of resource or service credits associated with an fund. An fund may have multiple allocations, each for use during a different time period.

An allocation has a start time and an end time that defines the time period during which the allocation may be used. By default an allocation is created with an unbounded time period (-infinity to infinity). An active flag is automatically updated to true if the fund is within its valid timeframe or false if it is not. An allocation may also have a credit limit representing the amount by which it can go negative. Thus, by having a positive balance in the Amount field, the fund is like a debit fund, implementing a pay-first use-later model. By establishing a credit limit instead of depositing an initial balance, the fund will be like a credit fund, implementing a use-first pay-later model. These strategies can be combined by depositing some amount of funds coupled with a credit limit, implementing a form of overdraft protection where the funds will be used down to the negative of the credit limit.

Field Name	Type	Description
id	**String**	The unique identifier for this allocation
active	**Boolean**	Indicates whether this allocation is active or not
allocated	**BigDecimal**	Adjusted allocation. This value stores the effective allocated amount based on the initial deposit and subsequent allocation adjustments via deposits, withdrawals or transfers.
amount	**BigDecimal**	The amount of this allocation

Field Name	Type	Description
creationTime	**Date**	The date this allocation was created
creditLimit	**BigDecimal**	Determines how far in the negative this allocation is permitted to be used (enforced in quotes and liens)
deleted	**Boolean**	A boolean indicating whether this allocation is deleted or not
description	**String**	The description of this allocation
endTime	**Date**	The date this allocation becomes inactive
fund	**String**	The fund Id associated with this allocation
modificationTime	**Date**	The date this allocation was last modified
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The date this allocation becomes active
transactionId	**Long**	The id of the last modifying transaction

FundConstraint

Constraints designate which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the encapsulated credits in a fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, etc.).

Field Name	Type	Description
id	**String**	The unique identifier of this constraint.
fund	**String**	The fund ID that this constraint is associated with.
name	**String**	The name of the constraint.
value	**String**	The value of the constraint. The constraint may be negated by the used of an exclamation point leading the value.

FundTransactionSummary

Represents a Moab Accounting Manager transaction summary, which is a consolidated view of multiple transactions. The transactions are grouped by object and action, and a total count is given for the summary.

Field Name	Type	Description
id	**Long**	
count	**Long**	The number of transactions in this grouping of object and action
action	**String**	Action name for the transaction
amount	**BigDecimal**	Amount of the transaction. A positive or amount signifies a credit, while a negative or zero amount signifies a debit.
object	**String**	Object's name associated with the transaction

API version 2

FundStatementSummary

An fund statement summary is related to and quite similar to the FundStatement report, but differs in the transactions field by using the FundTransactionSummary.

Field Name	Type	Description
id	**Long**	
endBalance	**BigDecimal**	The balance of the funds at the endTime of the statement
endTime	**Date**	The ending time that the statement covers
funds	**Set<Fund>**	The funds that this statement covers. Only a sub-set of the full fund fields are available from this property. This includes id, name, priority, description, and creationTime.
generationTime	**Date**	The date that the statement report was generated
startBalance	**BigDecimal**	The balance of the funds at the startTime of the statement
startTime	**Date**	The starting time that the statement covers
totalCredits	**BigDecimal**	The total number of credits that occurred during the time period that the statement covers
totalDebits	**BigDecimal**	The total number of debits that occurred during the time period that the statement covers
transactions	**Set<FundTransactionSummary>**	Summaries of the specific transactions which occurred during the time period that this statement covers.

Fund

A fund is a container for a time-bounded reference currency called credits for which the usage is restricted by constraints that define how the credits must be used. Much like with a bank, an fund is a repository for these resource or service credits which are added through deposits and debited through withdrawals and charges. Each fund has a set of constraints designating which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, Feature, etc.). Fund constraints may also be negated with an exclamation point leading the constraint value.

When credits are deposited into an fund, they are associated with a time period within which they are valid. These time-bounded pools of credits are known as allocations. (An allocation is a pool of billable units associated with an fund for use during a particular time period.) By using multiple allocations that expire in regular intervals it is possible to implement a use-it-or-lose-it policy and establish an allocation cycle.

Funds may be nested. Hierarchically nested funds may be useful for the delegation of management roles and responsibilities. Deposit shares may be established that assist to automate a trickle-down effect for funds deposited at higher level funds. Additionally, an optional overflow feature allows charges against lower level funds to trickle up the hierarchy.

Funds may have an arbitrary name which is not necessarily unique for the fund. Funds may also have a priority which will influence the order of fund selection when charging.

Field Name	Type	Description
id	**Long**	The unique fund identifier
allocated	**BigDecimal**	Total Adjusted allocations. This value is affected positively by deposits, activations and destination transfers and affected negatively by withdrawals, deactivations and source transfers that have occurred since the last reset.
allocations	**Set<Allocation>**	The allocations associated with this fund.
amount	**BigDecimal**	The sum of active allocation amounts within this fund. It does not take into fund current liens.
creationTime	**Date**	Date this fund was created
creditLimit	**BigDecimal**	The sum of active credit limits within this fund
defaultDeposit	**String**	The default deposit amount
deleted	**Boolean**	A boolean indicating whether this fund is deleted or not
description	**String**	The fund description
fundConstraints	**Set<FundConstraint>**	Constraints on fund usage.

Field Name	Type	Description
initialDeposit	**BigDecimal**	The initial deposit amount
modificationTime	**Date**	The date this fund was last modified
name	**String**	The name of this fund
priority	**Integer**	The fund priority
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction

Allocation

An allocation is a time-bounded pool of resource or service credits associated with an fund. An fund may have multiple allocations, each for use during a different time period.

An allocation has a start time and an end time that defines the time period during which the allocation may be used. By default an allocation is created with an unbounded time period (-infinity to infinity). An active flag is automatically updated to true if the fund is within its valid timeframe or false if it is not. An allocation may also have a credit limit representing the amount by which it can go negative. Thus, by having a positive balance in the Amount field, the fund is like a debit fund, implementing a pay-first use-later model. By establishing a credit limit instead of depositing an initial balance, the fund will be like a credit fund, implementing a use-first pay-later model. These strategies can be combined by depositing some amount of funds coupled with a credit limit, implementing a form of overdraft protection where the funds will be used down to the negative of the credit limit.

Field Name	Type	Description
id	**String**	The unique identifier for this allocation
active	**Boolean**	Indicates whether this allocation is active or not
allocated	**BigDecimal**	Adjusted allocation. This value stores the effective allocated amount based on the initial deposit and subsequent allocation adjustments via deposits, withdrawals or transfers.
amount	**BigDecimal**	The amount of this allocation

Field Name	Type	Description
creationTime	**Date**	The date this allocation was created
creditLimit	**BigDecimal**	Determines how far in the negative this allocation is permitted to be used (enforced in quotes and liens)
deleted	**Boolean**	A boolean indicating whether this allocation is deleted or not
description	**String**	The description of this allocation
endTime	**Date**	The date this allocation becomes inactive
fund	**String**	The fund Id associated with this allocation
modificationTime	**Date**	The date this allocation was last modified
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The date this allocation becomes active
transactionId	**Long**	The id of the last modifying transaction

FundConstraint

Constraints designate which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the encapsulated credits in a fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, etc.).

Field Name	Type	Description
id	**String**	The unique identifier of this constraint.
fund	**String**	The fund ID that this constraint is associated with.
name	**String**	The name of the constraint.
value	**String**	The value of the constraint. The constraint may be negated by the used of an exclamation point leading the value.

FundTransactionSummary

Represents a Moab Accounting Manager transaction summary, which is a consolidated view of multiple transactions. The transactions are grouped by object and action, and a total count is given for the summary.

Field Name	Type	Description
id	**Long**	
count	**Long**	The number of transactions in this grouping of object and action
action	**String**	Action name for the transaction
amount	**BigDecimal**	Amount of the transaction. A positive or amount signifies a credit, while a negative or zero amount signifies a debit.
object	**String**	Object's name associated with the transaction

Related topics

- **Accounting Funds** on page 1441

Fields: Fund Statements

> See the associated **Accounting Funds** on page 1441 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`accounting/funds/reports/statement`	**Permissions** on page 1571
Hooks filename	`accounting.funds.reports.statement.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

FundStatement

An fund statement is a report generated from Moab Accounting Manager fund, allocation, and transaction data. It contains fields detailing the specific time period covered, the starting and ending balances, the total of the transactions, and fund and transaction details.

Field Name	Type	Description
id	**Long**	
endBalance	**BigDecimal**	The balance of the funds at the endTime of the statement
endTime	**Date**	The ending time that the statement covers
funds	**Set<Fund>**	The funds that this statement covers. Only a sub-set of the full fund fields are available from this property. This includes id, name, priority, description, and creationTime.
generationTime	**Date**	The date that the statement report was generated
startBalance	**BigDecimal**	The balance of the funds at the startTime of the statement
startTime	**Date**	The starting time that the statement covers
totalCredits	**BigDecimal**	The total number of credits that occurred during the time period that the statement covers
totalDebits	**BigDecimal**	The total number of debits that occurred during the time period that the statement covers
transactions	**Set<FundTransaction>**	Details of each specific transaction which occurred during the time period that this statement covers.

Fund

A fund is a container for a time-bounded reference currency called credits for which the usage is restricted by constraints that define how the credits must be used. Much like with a bank, an fund is a repository for these resource or service credits which are added through deposits and debited through withdrawals and charges. Each fund has a set of constraints designating which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, Feature, etc.). Fund constraints may also be negated with an exclamation point leading the constraint value.

When credits are deposited into an fund, they are associated with a time period within which they are valid. These time-bounded pools of credits are known as allocations. (An allocation is a pool of billable units associated with an fund for use during a particular time period.) By using multiple allocations that expire in regular intervals it is possible to implement a use-it-or-lose-it policy and establish an allocation cycle.

Funds may be nested. Hierarchically nested funds may be useful for the delegation of management roles and responsibilities. Deposit shares may be established that assist to automate a trickle-down effect for funds deposited at higher level funds. Additionally, an optional overflow feature allows charges against lower level funds to trickle up the hierarchy.

Funds may have an arbitrary name which is not necessarily unique for the fund. Funds may also have a priority which will influence the order of fund selection when charging.

Field Name	Type	Description
id	**Long**	The unique fund identifier
allocated	**BigDecimal**	Total Adjusted allocations. This value is affected positively by deposits, activations and destination transfers and affected negatively by withdrawals, deactivations and source transfers that have occurred since the last reset.
allocations	**Set<Allocation>**	The allocations associated with this fund.
amount	**BigDecimal**	The sum of active allocation amounts within this fund. It does not take into fund current liens.
creationTime	**Date**	Date this fund was created
creditLimit	**BigDecimal**	The sum of active credit limits within this fund
defaultDeposit	**String**	The default deposit amount
deleted	**Boolean**	A boolean indicating whether this fund is deleted or not
description	**String**	The fund description
fundConstraints	**Set<FundConstraint>**	Constraints on fund usage.
initialDeposit	**BigDecimal**	The initial deposit amount

Field Name	Type	Description
modificationTime	**Date**	The date this fund was last modified
name	**String**	The name of this fund
priority	**Integer**	The fund priority
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction

Allocation

An allocation is a time-bounded pool of resource or service credits associated with an fund. An fund may have multiple allocations, each for use during a different time period.

An allocation has a start time and an end time that defines the time period during which the allocation may be used. By default an allocation is created with an unbounded time period (-infinity to infinity). An active flag is automatically updated to true if the fund is within its valid timeframe or false if it is not. An allocation may also have a credit limit representing the amount by which it can go negative. Thus, by having a positive balance in the Amount field, the fund is like a debit fund, implementing a pay-first use-later model. By establishing a credit limit instead of depositing an initial balance, the fund will be like a credit fund, implementing a use-first pay-later model. These strategies can be combined by depositing some amount of funds coupled with a credit limit, implementing a form of overdraft protection where the funds will be used down to the negative of the credit limit.

Field Name	Type	Description
id	**String**	The unique identifier for this allocation
active	**Boolean**	Indicates whether this allocation is active or not
allocated	**BigDecimal**	Adjusted allocation. This value stores the effective allocated amount based on the initial deposit and subsequent allocation adjustments via deposits, withdrawals or transfers.
amount	**BigDecimal**	The amount of this allocation
creationTime	**Date**	The date this allocation was created

Field Name	Type	Description
creditLimit	**BigDecimal**	Determines how far in the negative this allocation is permitted to be used (enforced in quotes and liens)
deleted	**Boolean**	A boolean indicating whether this allocation is deleted or not
description	**String**	The description of this allocation
endTime	**Date**	The date this allocation becomes inactive
fund	**String**	The fund Id associated with this allocation
modificationTime	**Date**	The date this allocation was last modified
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The date this allocation becomes active
transactionId	**Long**	The id of the last modifying transaction

FundConstraint

Constraints designate which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the encapsulated credits in a fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, etc.).

Field Name	Type	Description
id	**String**	The unique identifier of this constraint.
fund	**String**	The fund ID that this constraint is associated with.
name	**String**	The name of the constraint.
value	**String**	The value of the constraint. The constraint may be negated by the used of an exclamation point leading the value.

FundTransaction

Represents a Moab Accounting Manager transaction.

Field Name	Type	Description
id	**Long**	
account	**String**	The account associated with the transaction. For a credit this will likely be zero
action	**String**	Action name for the transaction
amount	**BigDecimal**	Amount of the transaction. A positive or amount signifies a credit, while a negative or zero amount signifies a debit.
instance	**String**	Instance name
machine	**String**	The machine associated with the transaction. For a credit this will likely be zero. This field is not available in the Cloud context.
object	**String**	Object's name associated with the transaction
time	**Date**	The date at which the transaction occurred
user	**String**	The user associated with the transaction. For a credit this will likely be zero

API version 2

FundStatement

An fund statement is a report generated from Moab Accounting Manager fund, allocation, and transaction data. It contains fields detailing the specific time period covered, the starting and ending balances, the total of the transactions, and fund and transaction details.

Field Name	Type	Description
id	**Long**	
endBalance	**BigDecimal**	The balance of the funds at the endTime of the statement
endTime	**Date**	The ending time that the statement covers
funds	**Set<Fund>**	The funds that this statement covers. Only a sub-set of the full fund fields are available from this property. This includes id, name, priority, description, and creationTime.
generationTime	**Date**	The date that the statement report was generated
startBalance	**BigDecimal**	The balance of the funds at the startTime of the statement
startTime	**Date**	The starting time that the statement covers
totalCredits	**BigDecimal**	The total number of credits that occurred during the time period that the statement covers
totalDebits	**BigDecimal**	The total number of debits that occurred during the time period that the statement covers
transactions	**Set<FundTransaction>**	Details of each specific transaction which occurred during the time period that this statement covers.

Fund

A fund is a container for a time-bounded reference currency called credits for which the usage is restricted by constraints that define how the credits must be used. Much like with a bank, an fund is a repository for these resource or service credits which are added through deposits and debited through withdrawals and charges. Each fund has a set of constraints designating which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, Feature, etc.). Fund constraints may also be negated with an exclamation point leading the constraint value.

When credits are deposited into an fund, they are associated with a time period within which they are valid. These time-bounded pools of credits are known as allocations. (An allocation is a pool of billable units associated with an fund for use during a particular time period.) By using multiple allocations that expire in regular intervals it is possible to implement a use-it-or-lose-it policy and establish an allocation cycle.

Funds may be nested. Hierarchically nested funds may be useful for the delegation of management roles and responsibilities. Deposit shares may be established that assist to automate a trickle-down effect for funds deposited at higher level funds. Additionally, an optional overflow feature allows charges against lower level funds to trickle up the hierarchy.

Funds may have an arbitrary name which is not necessarily unique for the fund. Funds may also have a priority which will influence the order of fund selection when charging.

Field Name	Type	Description
id	**Long**	The unique fund identifier
allocated	**BigDecimal**	Total Adjusted allocations. This value is affected positively by deposits, activations and destination transfers and affected negatively by withdrawals, deactivations and source transfers that have occurred since the last reset.
allocations	**Set<Allocation>**	The allocations associated with this fund.
amount	**BigDecimal**	The sum of active allocation amounts within this fund. It does not take into fund current liens.
creationTime	**Date**	Date this fund was created
creditLimit	**BigDecimal**	The sum of active credit limits within this fund
defaultDeposit	**String**	The default deposit amount
deleted	**Boolean**	A boolean indicating whether this fund is deleted or not
description	**String**	The fund description
fundConstraints	**Set<FundConstraint>**	Constraints on fund usage.
initialDeposit	**BigDecimal**	The initial deposit amount

Field Name	Type	Description
modificationTime	**Date**	The date this fund was last modified
name	**String**	The name of this fund
priority	**Integer**	The fund priority
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction

Allocation

An allocation is a time-bounded pool of resource or service credits associated with an fund. An fund may have multiple allocations, each for use during a different time period.

An allocation has a start time and an end time that defines the time period during which the allocation may be used. By default an allocation is created with an unbounded time period (-infinity to infinity). An active flag is automatically updated to true if the fund is within its valid timeframe or false if it is not. An allocation may also have a credit limit representing the amount by which it can go negative. Thus, by having a positive balance in the Amount field, the fund is like a debit fund, implementing a pay-first use-later model. By establishing a credit limit instead of depositing an initial balance, the fund will be like a credit fund, implementing a use-first pay-later model. These strategies can be combined by depositing some amount of funds coupled with a credit limit, implementing a form of overdraft protection where the funds will be used down to the negative of the credit limit.

Field Name	Type	Description
id	**String**	The unique identifier for this allocation
active	**Boolean**	Indicates whether this allocation is active or not
allocated	**BigDecimal**	Adjusted allocation. This value stores the effective allocated amount based on the initial deposit and subsequent allocation adjustments via deposits, withdrawals or transfers.
amount	**BigDecimal**	The amount of this allocation
creationTime	**Date**	The date this allocation was created

Field Name	Type	Description
creditLimit	**BigDecimal**	Determines how far in the negative this allocation is permitted to be used (enforced in quotes and liens)
deleted	**Boolean**	A boolean indicating whether this allocation is deleted or not
description	**String**	The description of this allocation
endTime	**Date**	The date this allocation becomes inactive
fund	**String**	The fund Id associated with this allocation
modificationTime	**Date**	The date this allocation was last modified
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The date this allocation becomes active
transactionId	**Long**	The id of the last modifying transaction

FundConstraint

Constraints designate which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the encapsulated credits in a fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, etc.).

Field Name	Type	Description
id	**String**	The unique identifier of this constraint.
fund	**String**	The fund ID that this constraint is associated with.
name	**String**	The name of the constraint.
value	**String**	The value of the constraint. The constraint may be negated by the used of an exclamation point leading the value.

FundTransaction

Represents a Moab Accounting Manager transaction.

Field Name	Type	Description
id	**Long**	
account	**String**	The account associated with the transaction. For a credit this will likely be zero
action	**String**	Action name for the transaction
amount	**BigDecimal**	Amount of the transaction. A positive or amount signifies a credit, while a negative or zero amount signifies a debit.
instance	**String**	Instance name
machine	**String**	The machine associated with the transaction. For a credit this will likely be zero. This field is not available in the Cloud context.
object	**String**	Object's name associated with the transaction
time	**Date**	The date at which the transaction occurred
user	**String**	The user associated with the transaction. For a credit this will likely be zero

Related topics

- **Accounting Funds** on page 1441

Fields: Funds

See the associated **Accounting Funds** on page 1441 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`accounting/funds`	**Permissions** on page 1571
Hooks filename	`accounting.funds.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

Fund

A fund is a container for a time-bounded reference currency called credits for which the usage is restricted by constraints that define how the credits must be used. Much like with a bank, an fund is a repository for these resource or service credits which are added through deposits and debited through withdrawals and charges. Each fund has a set of constraints designating which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, Feature, etc.). Fund constraints may also be negated with an exclamation point leading the constraint value.

When credits are deposited into an fund, they are associated with a time period within which they are valid. These time-bounded pools of credits are known as allocations. (An allocation is a pool of billable units associated with an fund for use during a particular time period.) By using multiple allocations that expire in regular intervals it is possible to implement a use-it-or-lose-it policy and establish an allocation cycle.

Funds may be nested. Hierarchically nested funds may be useful for the delegation of management roles and responsibilities. Deposit shares may be established that assist to automate a trickle-down effect for funds deposited at higher level funds. Additionally, an optional overflow feature allows charges against lower level funds to trickle up the hierarchy.

Funds may have an arbitrary name which is not necessarily unique for the fund. Funds may also have a priority which will influence the order of fund selection when charging.

Field Name	Type	Description
id	**Long**	The unique fund identifier
allocated	**BigDecimal**	Total Adjusted allocations. This value is affected positively by deposits, activations and destination transfers and affected negatively by withdrawals, deactivations and source transfers that have occurred since the last reset.
allocations	**Set<Allocation>**	The allocations associated with this fund.
amount	**BigDecimal**	The sum of active allocation amounts within this fund. It does not take into fund current liens.
creationTime	**Date**	Date this fund was created
creditLimit	**BigDecimal**	The sum of active credit limits within this fund
defaultDeposit	**String**	The default deposit amount

Field Name	Type	Description
deleted	**Boolean**	A boolean indicating whether this fund is deleted or not
description	**String**	The fund description
fundConstraints	**Set<FundConstraint>**	Constraints on fund usage.
initialDeposit	**BigDecimal**	The initial deposit amount
modificationTime	**Date**	The date this fund was last modified
name	**String**	The name of this fund
priority	**Integer**	The fund priority
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction

Allocation

An allocation is a time-bounded pool of resource or service credits associated with an fund. An fund may have multiple allocations, each for use during a different time period.

An allocation has a start time and an end time that defines the time period during which the allocation may be used. By default an allocation is created with an unbounded time period (-infinity to infinity). An active flag is automatically updated to true if the fund is within its valid timeframe or false if it is not. An allocation may also have a credit limit representing the amount by which it can go negative. Thus, by having a positive balance in the Amount field, the fund is like a debit fund, implementing a pay-first use-later model. By establishing a credit limit instead of depositing an initial balance, the fund will be like a credit fund, implementing a use-first pay-later model. These strategies can be combined by depositing some amount of funds coupled with a credit limit, implementing a form of overdraft protection where the funds will be used down to the negative of the credit limit.

Field Name	Type	Description
id	**String**	The unique identifier for this allocation
active	**Boolean**	Indicates whether this allocation is active or not

Field Name	Type	Description
allocated	**BigDecimal**	Adjusted allocation. This value stores the effective allocated amount based on the initial deposit and subsequent allocation adjustments via deposits, withdrawals or transfers.
amount	**BigDecimal**	The amount of this allocation
creationTime	**Date**	The date this allocation was created
creditLimit	**BigDecimal**	Determines how far in the negative this allocation is permitted to be used (enforced in quotes and liens)
deleted	**Boolean**	A boolean indicating whether this allocation is deleted or not
description	**String**	The description of this allocation
endTime	**Date**	The date this allocation becomes inactive
fund	**String**	The fund Id associated with this allocation
modificationTime	**Date**	The date this allocation was last modified
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The date this allocation becomes active
transactionId	**Long**	The id of the last modifying transaction

FundConstraint

Constraints designate which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the encapsulated credits in a fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, etc.).

Field Name	Type	Description
id	**String**	The unique identifier of this constraint.
fund	**String**	The fund ID that this constraint is associated with.

Field Name	Type	Description
name	**String**	The name of the constraint.
value	**String**	The value of the constraint. The constraint may be negated by the used of an exclamation point leading the value.

API version 2

Fund

A fund is a container for a time-bounded reference currency called credits for which the usage is restricted by constraints that define how the credits must be used. Much like with a bank, an fund is a repository for these resource or service credits which are added through deposits and debited through withdrawals and charges. Each fund has a set of constraints designating which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, Feature, etc.). Fund constraints may also be negated with an exclamation point leading the constraint value.

When credits are deposited into an fund, they are associated with a time period within which they are valid. These time-bounded pools of credits are known as allocations. (An allocation is a pool of billable units associated with an fund for use during a particular time period.) By using multiple allocations that expire in regular intervals it is possible to implement a use-it-or-lose-it policy and establish an allocation cycle.

Funds may be nested. Hierarchically nested funds may be useful for the delegation of management roles and responsibilities. Deposit shares may be established that assist to automate a trickle-down effect for funds deposited at higher level funds. Additionally, an optional overflow feature allows charges against lower level funds to trickle up the hierarchy.

Funds may have an arbitrary name which is not necessarily unique for the fund. Funds may also have a priority which will influence the order of fund selection when charging.

Field Name	Type	Description
id	**Long**	The unique fund identifier
allocated	**BigDecimal**	Total Adjusted allocations. This value is affected positively by deposits, activations and destination transfers and affected negatively by withdrawals, deactivations and source transfers that have occurred since the last reset.
allocations	**Set<Allocation>**	The allocations associated with this fund.
amount	**BigDecimal**	The sum of active allocation amounts within this fund. It does not take into fund current liens.
creationTime	**Date**	Date this fund was created
creditLimit	**BigDecimal**	The sum of active credit limits within this fund
defaultDeposit	**String**	The default deposit amount

Field Name	Type	Description
deleted	**Boolean**	A boolean indicating whether this fund is deleted or not
description	**String**	The fund description
fundConstraints	**Set<FundConstraint>**	Constraints on fund usage.
initialDeposit	**BigDecimal**	The initial deposit amount
modificationTime	**Date**	The date this fund was last modified
name	**String**	The name of this fund
priority	**Integer**	The fund priority
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction

Allocation

An allocation is a time-bounded pool of resource or service credits associated with an fund. An fund may have multiple allocations, each for use during a different time period.

An allocation has a start time and an end time that defines the time period during which the allocation may be used. By default an allocation is created with an unbounded time period (-infinity to infinity). An active flag is automatically updated to true if the fund is within its valid timeframe or false if it is not. An allocation may also have a credit limit representing the amount by which it can go negative. Thus, by having a positive balance in the Amount field, the fund is like a debit fund, implementing a pay-first use-later model. By establishing a credit limit instead of depositing an initial balance, the fund will be like a credit fund, implementing a use-first pay-later model. These strategies can be combined by depositing some amount of funds coupled with a credit limit, implementing a form of overdraft protection where the funds will be used down to the negative of the credit limit.

Field Name	Type	Description
id	**String**	The unique identifier for this allocation
active	**Boolean**	Indicates whether this allocation is active or not

Field Name	Type	Description
allocated	**BigDecimal**	Adjusted allocation. This value stores the effective allocated amount based on the initial deposit and subsequent allocation adjustments via deposits, withdrawals or transfers.
amount	**BigDecimal**	The amount of this allocation
creationTime	**Date**	The date this allocation was created
creditLimit	**BigDecimal**	Determines how far in the negative this allocation is permitted to be used (enforced in quotes and liens)
deleted	**Boolean**	A boolean indicating whether this allocation is deleted or not
description	**String**	The description of this allocation
endTime	**Date**	The date this allocation becomes inactive
fund	**String**	The fund Id associated with this allocation
modificationTime	**Date**	The date this allocation was last modified
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The date this allocation becomes active
transactionId	**Long**	The id of the last modifying transaction

FundConstraint

Constraints designate which entities (such as Users, Accounts, Machines, Classes, Organizations, etc.) may access the encapsulated credits in a fund or for which aspects of usage the funds are intended (QualityOfService, GeographicalArea, etc.).

Field Name	Type	Description
id	**String**	The unique identifier of this constraint.
fund	**String**	The fund ID that this constraint is associated with.

Field Name	Type	Description
name	**String**	The name of the constraint.
value	**String**	The value of the constraint. The constraint may be negated by the used of an exclamation point leading the value.

Related topics

- **Accounting Funds** on page 1441

Fields: Liens

See the associated **Accounting Liens** on page 1451 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`accounting/liens`	**Permissions** on page 1571
Hooks filename	`accounting.liens.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

Lien

A lien is a reservation or hold placed against an allocation. Before usage of a resource or service begins, a lien is placed against one or more allocations within the requesting user's applicable funds. Subsequent usage requests will also post liens while the available balance (active allocations minus liens) allows. When the usage ends, the lien is removed and the actual charge is made to the allocation(s). This procedure ensures that usage will only be permitted so long as the requestors have sufficient funds.

Field Name	Type	Description
id	**Long**	The unique lien identifier
allocations	**Set<LienAllocation>**	The allocation amounts reserved with this lien.
creationTime	**Date**	The date this lien was created
deleted	**Boolean**	A boolean indicating whether this lien is deleted or not
description	**String**	The lien description
duration	**Long**	The expected duration of the reserved usage in seconds
endTime	**Date**	The time the lien becomes inactive
instance	**String**	The lien is against the specified instance (i.e. job id)
modificationTime	**Date**	The date this lien was last modified
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The time the lien becomes active
transactionId	**Long**	The id of the last modifying transaction
usageRecord	**Long**	The id of the usage record associated with the lien and containing the usage properties

LienAllocation

Amounts of the allocations that the lien has holds against

Field Name	Type	Description
id	**String**	The child allocation id
amount	**Long**	The amount reserved against the allocation by this lien
fund	**Long**	The fund that the allocation is in
lien	**String**	The parent lien id

API version 2

Lien

A lien is a reservation or hold placed against an allocation. Before usage of a resource or service begins, a lien is placed against one or more allocations within the requesting user's applicable funds. Subsequent usage requests will also post liens while the available balance (active allocations minus liens) allows. When the usage ends, the lien is removed and the actual charge is made to the allocation(s). This procedure ensures that usage will only be permitted so long as the requestors have sufficient funds.

Field Name	Type	Description
id	**Long**	The unique lien identifier
allocations	**Set<LienAllocation>**	The allocation amounts reserved with this lien.
creationTime	**Date**	The date this lien was created
deleted	**Boolean**	A boolean indicating whether this lien is deleted or not
description	**String**	The lien description
duration	**Long**	The expected duration of the reserved usage in seconds
endTime	**Date**	The time the lien becomes inactive
instance	**String**	The lien is against the specified instance (i.e. job id)
modificationTime	**Date**	The date this lien was last modified
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The time the lien becomes active
transactionId	**Long**	The id of the last modifying transaction
usageRecord	**Long**	The id of the usage record associated with the lien and containing the usage properties

LienAllocation

Amounts of the allocations that the lien has holds against

Field Name	Type	Description
id	**String**	The child allocation id
amount	**Long**	The amount reserved against the allocation by this lien
fund	**Long**	The fund that the allocation is in
lien	**String**	The parent lien id

Related topics

- **Accounting Liens** on page 1451

Fields: Organizations

See the associated **Accounting Organizations** on page 1455 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`accounting/organizations`	**Permissions** on page 1571
Hooks filename	`accounting.organizations.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

Organization

An organization is a virtual organization in which accounts are grouped. An account may only belong to a single organization while an organization may have multiple accounts. For example, an account may represent a project or cost-center while an organization may represent an institutional department or business division. The purpose of defining organizations is to support the ability to produce reporting for higher-order organizational entities beyond the individual account. Default organization properties include an id (name in MAM) and a description. An organization can be created, queried, modified, and deleted.

Field Name	Type	Description
id	**String**	The unique organization identifier
creationTime	**Date**	The date this organization was created
deleted	**Boolean**	A boolean indicating whether this organization is deleted or not
description	**String**	The organization description
modificationTime	**Date**	The date this organization was last modified
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction

API version 2

Organization

An organization is a virtual organization in which accounts are grouped. An account may only belong to a single organization while an organization may have multiple accounts. For example, an account may represent a project or cost-center while an organization may represent an institutional department or business division. The purpose of defining organizations is to support the ability to produce reporting for higher-order organizational entities beyond the individual account. Default organization properties include an id (name in MAM) and a description. An organization can be created, queried, modified, and deleted.

Field Name	Type	Description
id	**String**	The unique organization identifier
creationTime	**Date**	The date this organization was created
deleted	**Boolean**	A boolean indicating whether this organization is deleted or not
description	**String**	The organization description
modificationTime	**Date**	The date this organization was last modified
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction

Related topics

- **Accounting Organizations** on page 1455

Fields: Quotes

See the associated **Accounting Quotes** on page 1458 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`accounting/quotes`	**Permissions** on page 1571

Type	Value	Additional information
Hooks filename	`accounting.quotes.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

Quote

Quotes can be used to determine how much it will cost to use a resource or service. Provided the cost-only option is not specified, this step will additionally verify that the submitter has sufficient funds and meets all the allocation policy requirements for the usage, and can be used at the submission of the usage request as an early filter to prevent the usage from getting blocked when it tries to obtain a lien to start later. If a guaranteed quote is requested, a quote id is returned and can be used in the subsequent charge to guarantee the rates that were used to form the original quote. A guaranteed quote has the side effect of creating a quote record and a permanent usage record. A quote id will be returned which can be used with the lien and charge to claim the quoted charge rates. A cost-only quote can be used to determine how much would be charged for usage without verifying sufficient funds or checking to see if the charge could succeed.

Field Name	Type	Description
id	**Long**	The unique quote identifier
amount	**BigDecimal**	The total amount of the quote
chargeRates	**Set<QuoteChargeRate>**	The applied charges that make up this quote.
creationTime	**Date**	The date this quote was created
deleted	**Boolean**	A boolean indicating whether this quote is deleted or not
description	**String**	The quote description
duration	**Long**	The expected duration of the quoted usage in seconds
endTime	**Date**	The time the quote becomes inactive
instance	**String**	The quote instance name. (i.e. job id)
modificationTime	**Date**	The date this quote was last modified
pinned	**Boolean**	Boolean indicating whether the quote is pinned or not
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The time the quote becomes active

Field Name	Type	Description
transactionId	**Long**	The id of the last modifying transaction
usageRecord	**Long**	The usage record id associated with this quote

QuoteChargeRate

Saved charge rates to be used when the quote is referenced

Field Name	Type	Description
id	**Long**	
amount	**String**	The charge rate amount
name	**String**	The child charge rate name
quote	**String**	The parent quote id
value	**String**	The child charge rate value

API version 2

Quote

Quotes can be used to determine how much it will cost to use a resource or service. Provided the cost-only option is not specified, this step will additionally verify that the submitter has sufficient funds and meets all the allocation policy requirements for the usage, and can be used at the submission of the usage request as an early filter to prevent the usage from getting blocked when it tries to obtain a lien to start later. If a guaranteed quote is requested, a quote id is returned and can be used in the subsequent charge to guarantee the rates that were used to form the original quote. A guaranteed quote has the side effect of creating a quote record and a permanent usage record. A quote id will be returned which can be used with the lien and charge to claim the quoted charge rates. A cost-only quote can be used to determine how much would be charged for usage without verifying sufficient funds or checking to see if the charge could succeed.

Field Name	Type	Description
id	**Long**	The unique quote identifier
amount	**BigDecimal**	The total amount of the quote
chargeRates	**Set<QuoteChargeRate>**	The applied charges that make up this quote.
creationTime	**Date**	The date this quote was created
deleted	**Boolean**	A boolean indicating whether this quote is deleted or not
description	**String**	The quote description
duration	**Long**	The expected duration of the quoted usage in seconds
endTime	**Date**	The time the quote becomes inactive
instance	**String**	The quote instance name. (i.e. job id)
modificationTime	**Date**	The date this quote was last modified
pinned	**Boolean**	Boolean indicating whether the quote is pinned or not
requestId	**Long**	The id of the last modifying request
startTime	**Date**	The time the quote becomes active

Field Name	Type	Description
transactionId	**Long**	The id of the last modifying transaction
usageRecord	**Long**	The usage record id associated with this quote

QuoteChargeRate

Saved charge rates to be used when the quote is referenced

Field Name	Type	Description
id	**Long**	
amount	**String**	The charge rate amount
name	**String**	The child charge rate name
quote	**String**	The parent quote id
value	**String**	The child charge rate value

Related topics

- **Accounting Quotes** on page 1458

Fields: Transactions

See the associated **Accounting Transactions** on page 1461 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`accounting/transactions`	**Permissions** on page 1571
Hooks filename	`accounting.transactions.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

Transaction

Moab Accounting Manager logs all modifying transactions in a detailed transaction journal (queries are not recorded). Previous transactions can be queried but not modified or deleted. By default, a standard user may only query transactions performed by them.

Field Name	Type	Description
id	**Long**	The unique transaction identifier
account	**String**	The account name associated with the transaction
action	**String**	The transaction action name
actor	**String**	The authenticated user that performed the action
allocation	**Long**	The allocation id associated with the transaction
amount	**BigDecimal**	The amount
child	**String**	If the transaction object is an association, this is the value of the child
creationTime	**Date**	The date this transaction was created
deleted	**Boolean**	A boolean indicating whether this transaction is deleted or not
delta	**BigDecimal**	The effective change (positive or negative) to the balance of an allocation
description	**String**	The description for the transaction
duration	**Long**	The duration associated with the transaction in seconds
fund	**Long**	The fund id associated with the transaction
instance	**String**	The instance name (e.g. the job id)
key	**String**	The object primary key value

Field Name	Type	Description
machine	**String**	The machine name associated with the transaction (e.g. the cluster name)
modificationTime	**Date**	The date this transaction was last modified
object	**String**	The transaction object name
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction
usageRecord	**Long**	The usage record id associated with the transaction
user	**String**	The user name associated with the transaction

API version 2

Transaction

Moab Accounting Manager logs all modifying transactions in a detailed transaction journal (queries are not recorded). Previous transactions can be queried but not modified or deleted. By default, a standard user may only query transactions performed by them.

Field Name	Type	Description
id	**Long**	The unique transaction identifier
account	**String**	The account name associated with the transaction
action	**String**	The transaction action name
actor	**String**	The authenticated user that performed the action
allocation	**Long**	The allocation id associated with the transaction
amount	**BigDecimal**	The amount
child	**String**	If the transaction object is an association, this is the value of the child
creationTime	**Date**	The date this transaction was created
deleted	**Boolean**	A boolean indicating whether this transaction is deleted or not
delta	**BigDecimal**	The effective change (positive or negative) to the balance of an allocation
description	**String**	The description for the transaction
duration	**Long**	The duration associated with the transaction in seconds
fund	**Long**	The fund id associated with the transaction
instance	**String**	The instance name (e.g. the job id)
key	**String**	The object primary key value

Field Name	Type	Description
machine	**String**	The machine name associated with the transaction (e.g. the cluster name)
modificationTime	**Date**	The date this transaction was last modified
object	**String**	The transaction object name
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction
usageRecord	**Long**	The usage record id associated with the transaction
user	**String**	The user name associated with the transaction

Related topics

- **Accounting Transactions** on page 1461

Fields: Usage Records

> See the associated **Accounting Usage records** on page 1466 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`accounting/usage-records`	**Permissions** on page 1571
Hooks filename	`accounting.usage-record-s.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-sup-ported	No	**Distinct** on page 1504

API version 3

UsageRecord

A usage record tracks the usage of resources and services on your system, recording the charge and the details of the usage in a usage record.

Usage Record quotes can be used to determine how much it will cost to use a resource or service. Provided the cost-only option is not specified, this step will additionally verify that the submitter has sufficient funds and meets all the allocation policy requirements for the usage, and can be used at the submission of the usage request as an early filter to prevent the usage from getting blocked when it tries to obtain a lien to start later. If a guaranteed quote is requested, a quote id is returned and can be used in the subsequent charge to guarantee the rates that were used to form the original quote. A guaranteed quote has the side effect of creating a quote record and a permanent usage record. A quote id will be returned which can be used with the lien and charge to claim the quoted charge rates. A cost-only quote can be used to determine how much would be charged for usage without verifying sufficient funds or checking to see if the charge could succeed.

A usage lien can be used to place a hold on the user's fund before usage starts to ensure that the credits will be there when it completes. The replace option may be specified if you want the new lien to replace existing liens of the same instance name (associated with the same usage record). The modify option may be specified to dynamically extend any existing lien with the same instance name with the specified characteristics instead of creating a new one.

A usage charge debits the appropriate allocations based on the attributes of the usage. The charge is calculated based on factors including the resources and services used, the usage time, and other quality-based factors. By default, any liens associated with the charge will be removed. The incremental option may be specified if you want associated liens to be reduced instead of removed. If a usage record already exists for the instance being charged it will be updated with the data properties passed in with the charge request, otherwise a new usage record will be created.

Field Name	Type	POST	Description
id	**Long**	No	The unique usage record identifier
charge	**String**	No	The cumulative amount charged
creationTime	**Date**	No	The date this usage record was created
deleted	**Boolean**	No	A boolean indicating whether this usage record is deleted or not
instance	**String**	No	The usage record instance name (i.e. job id)
modificationTime	**Date**	No	The date this usage record was last modified

Field Name	Type	POST	Description
qualityOfService	**String**	No	The quality of service associated with the usage
quote	**Long**	No	The associated quote id
requestId	**Long**	No	The id of the last modifying request
stage	**String**	No	The last affecting action (i.e. Create, Quote, Reserve, Query)
transactionId	**Long**	No	The id of the last modifying transaction
type	**String**	No	The usage record type
user	**String**	No	The user name associated with the usage

API version 2

UsageRecord

A usage record tracks the usage of resources and services on your system, recording the charge and the details of the usage in a usage record.

Usage Record quotes can be used to determine how much it will cost to use a resource or service. Provided the cost-only option is not specified, this step will additionally verify that the submitter has sufficient funds and meets all the allocation policy requirements for the usage, and can be used at the submission of the usage request as an early filter to prevent the usage from getting blocked when it tries to obtain a lien to start later. If a guaranteed quote is requested, a quote id is returned and can be used in the subsequent charge to guarantee the rates that were used to form the original quote. A guaranteed quote has the side effect of creating a quote record and a permanent usage record. A quote id will be returned which can be used with the lien and charge to claim the quoted charge rates. A cost-only quote can be used to determine how much would be charged for usage without verifying sufficient funds or checking to see if the charge could succeed.

A usage lien can be used to place a hold on the user's fund before usage starts to ensure that the credits will be there when it completes. The replace option may be specified if you want the new lien to replace existing liens of the same instance name (associated with the same usage record). The modify option may be specified to dynamically extend any existing lien with the same instance name with the specified characteristics instead of creating a new one.

A usage charge debits the appropriate allocations based on the attributes of the usage. The charge is calculated based on factors including the resources and services used, the usage time, and other quality-based factors. By default, any liens associated with the charge will be removed. The incremental option may be specified if you want associated liens to be reduced instead of removed. If a usage record already exists for the instance being charged it will be updated with the data properties passed in with the charge request, otherwise a new usage record will be created.

Field Name	Type	POST	Description
id	**Long**	No	The unique usage record identifier
charge	**String**	No	The cumulative amount charged
creationTime	**Date**	No	The date this usage record was created
deleted	**Boolean**	No	A boolean indicating whether this usage record is deleted or not
instance	**String**	No	The usage record instance name (i.e. job id)
modificationTime	**Date**	No	The date this usage record was last modified

Field Name	Type	POST	Description
qualityOfService	**String**	No	The quality of service associated with the usage
quote	**Long**	No	The associated quote id
requestId	**Long**	No	The id of the last modifying request
stage	**String**	No	The last affecting action (i.e. Create, Quote, Reserve, Query)
transactionId	**Long**	No	The id of the last modifying transaction
type	**String**	No	The usage record type
user	**String**	No	The user name associated with the usage

Related topics

- **Accounting Usage records** on page 1466

Fields: Users

See the associated **Accounting Users** on page 1480 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`accounting/users`	**Permissions** on page 1571
Hooks filename	`accounting.users.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

User

A user is a person authorized to use a resource or service. Default user properties include the common name, phone number, email address, default account, and description for that person.

Field Name	Type	Description
id	**String**	The unique user identifier
active	**Boolean**	A boolean indicating whether this user is active or not
creationTime	**Date**	The date this user was created
defaultAccount	**String**	The default account for this user
deleted	**Boolean**	A boolean indicating whether this user is deleted or not
description	**String**	The user description
emailAddress	**String**	The user's email address
modificationTime	**Date**	The date this user was last modified
phoneNumber	**String**	The user's phone number
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction

API version 2

User

A user is a person authorized to use a resource or service. Default user properties include the common name, phone number, email address, default account, and description for that person.

Field Name	Type	Description
id	**String**	The unique user identifier
active	**Boolean**	A boolean indicating whether this user is active or not
creationTime	**Date**	The date this user was created
defaultAccount	**String**	The default account for this user
deleted	**Boolean**	A boolean indicating whether this user is deleted or not
description	**String**	The user description
emailAddress	**String**	The user's email address
modificationTime	**Date**	The date this user was last modified
phoneNumber	**String**	The user's phone number
requestId	**Long**	The id of the last modifying request
transactionId	**Long**	The id of the last modifying transaction

Related topics

- **Accounting Users** on page 1480

Fields: Credentials

See the associated **Credentials** on page 1484 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`credentials`	**Permissions** on page 1571
Hooks filename	`credentials.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

Credential

A credential is an entity, such as a user or a group, that has access to resources. Credentials allow specification of job ownership, tracking of resource usage, enforcement of policies, and many other features.

Field Name	Type	PUT	Description
name	**String**	No	The name of the credential.

API version 2

Credential

A credential is an entity, such as a user or a group, that has access to resources. Credentials allow specification of job ownership, tracking of resource usage, enforcement of policies, and many other features.

Field Name	Type	PUT	Description
name	**String**	No	The name of the credential.

Related topics

- **Credentials** on page 1484

Fields: Events

See the associated **Events** on page 1506 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`events`	**Permissions** on page 1571
Hooks filename	`events.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	Yes	**Distinct** on page 1504

API version 3

Event

Represents an event originating from any component in the system (MWM, MWS, MAM, etc). Events are related to, but not the same as, Notifications. See NotificationCondition for an explanation of when to use an event vs a notification.

Field Name	Type	POST	Description
id	**String**	No	The unique ID for this event
arguments	**List<String>**	Yes	The event's arguments
associatedObjects	**Set<AssociatedObject>**	Yes	Objects relating to the event
code	**int**	Yes	This is a positive, 32-bit numeric value. Source code that needs to take action on events based on which event (error) occurred can switch based on this value. The top 16 bits are determined by the severity of the event and the component that emits it. The bottom 16 bits are assigned by any arbitrary mechanism convenient to a component. Each component thus has 64k unique event codes that it can assign. Once assigned, event codes are immutable; it can never be the case that error 12345 means one thing in release A, and a different thing in release B.
eventDate	**Date**	Yes	The date and time the event occurred, not the date and time MWS received the event. It is up to the reporting component to report this time accurately. Required during POST.
eventType	**String**	Yes	Signifies what type of event. Cannot contain single quotes(') or double quotes(").
message	**String**	Yes	A summary of what happened that caused this event
origin	**String**	Yes	The origin of this event. Cannot contain single quotes(') or double quotes(").
severity	**EventSeverity**	Yes	Signifies the severity of an event.

Field Name	Type	POST	Description
tenant	**Map<String, String>**	No	The event's tenant (contains tenant id and name)

AssociatedObject

Represents and uniquely identifies an object associated with an event. (e.g node, job, reservation, trigger)

Field Name	Type	POST	Description
id	**String**	Yes	The object id (e.g. reservation.1, job.21, vm3). Cannot contain single quotes(') or double quotes(").
type	**String**	Yes	The type of object (e.g. node, job, reservation). Cannot contain single quotes(') or double quotes(").

EventSeverity

Value	Description
INFO	
WARN	
ERROR	
FATAL	

API version 2

EventVersion2

Field Name	Type	POST	Description
id	**String**	No	The unique ID for this event
details	**Map<String, Map>**	Yes	A map where detail name maps to detail value. (e.g. "sourceHypervisor" => "blade256", "destinationHypervisor" => "blade257", "os" => "centos-5.5-stateless)
errorMessage	**ErrorMessageVersion2**	Yes	Details about any errors associated with the event. If this event was not associated with any errors this field will be null
eventCategory	**String**	Yes	Signifies what category of event.
eventTime	**Date**	Yes	The time the event occurred, not the time MWS received the event. It is up to the reporting component to report this time accurately. Corresponds to eventDate in API Version 3. Required during POST.
eventType	**String**	Yes	Signifies what type of event.
facility	**String**	Yes	A categorization of how this event fits in with other events.
initiatedBy	**UserDetailsVersion2**	Yes	Details about the user that initiated this event
primaryObject	**MoabObjectVersion2**	Yes	Most events will have a "primary object" associated with it. An event can have at most ONE primary object. For example, a JobStart event will have a primary job object, so the type would be "job" and the object ID would be the ID of the job. Primary objects are, however, optional, depending on the type of event. For example, a "SchedulerCommand" event does not have a primary object.

Field Name	Type	POST	Description
relatedObjects	Set<MoabObjectVersion2>	Yes	Objects relating to the event that are not the primary object. Corresponds to associatedObjects in API Version 3.
severity	**String**	Yes	Signifies the severity of an event. Severity can be "FATAL", "ERROR", "WARN", "INFO"
sourceComponent	**String**	Yes	What Adaptive Computing component reported this event. Examples: "MWM", "MWS", "MAM", etc. Corresponds to origin in API Version 3.
status	**String**	Yes	The status of the reported event.

ErrorMessageVersion2

Field Name	Type	POST	Description
errorCode	**String**	Yes	The original error code generated or detected by the originator.
message	**String**	Yes	If an event has a status of "failure" or other non-successful operation, this field should provide a human-friendly error message Corresponds to Event.message in API Version 3 and above.
originator	**String**	Yes	The software component or entity that generated or detected the error (e.g. Moab, Torque, MWS, Viewpoint, RM, Database, etc).

UserDetailsVersion2

Field Name	Type	POST	Description
proxyUser	**String**	Yes	The proxy user that initiated the event.
user	**String**	Yes	The user that initiated the event.

MoabObjectVersion2

Field Name	Type	POST	Description
id	**String**	Yes	The moab object id (e.g. reservation.1, job.21, vm3)
serialization	**String**	Yes	A serialized representation of the object
type	**String**	Yes	The moab object type (e.g. node, job, reservation)

Related topics

- **Events** on page 1506

Fields: Images

See the associated **Images** on page 1514 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`images`	**Permissions** on page 1571
Hooks filename	`images.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	Yes	**Distinct** on page 1504

API version 3

Image

An image is used to track the different types of operating systems and hypervisors available in a data center. If the image is a hypervisor, it can contain other images which are the available virtual machines of the hypervisor.

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of this image.
active	**Boolean**	Yes	Yes	If false, the image is flagged as inactive and should not be used. Defaults to true.

Field Name	Type	POST	PUT	Description
extensions	**Map<String, Map>**	Yes	Yes	A map containing maps which represent settings for provisioning managers. Only one extension may be present on an image at a time currently. Valid default provisioning manager specific extensions include 'xcat'. Required properties for 'xcat' when hypervisor is false: • os - The name of the operating system according to xCAT • architecture - The architecture, such as x86_64 • profile - The xCAT profile to use for the image Required properties for 'xcat' when hypervisor is true: • os - The name of the operating system according to xCAT • architecture - The architecture, such as x86_64 • profile - The xCAT profile to use for the image • hvGroupName - The name of the xCAT hypervisor group • vmGroupName - The name of the xCAT VM group
features	**Set<String>**	Yes	Yes	The set of features used by the provisioning manager.

Field Name	Type	POST	PUT	Description
hypervisor	**Boolean**	Yes	Yes	Whether or not the image is a hypervisor. Required during POST. Note that this is related to, but not the same as, supportsPhysicalMachine. Also, when this is false, no virtualizedImages may be specified for an image.
hypervisorType	**String**	Yes	Yes	The type of the hypervisor, which is indicative of the hypervisor technology used in this image. Required if this image is a hypervisor image.
name	**String**	Yes	Yes	The unique human-readable name of this image. Required during POST.
osType	**String**	Yes	Yes	The type of the operating system such as 'Linux' or 'Windows'. Required during POST.
supportsPhysicalMachine	**Boolean**	Yes	Yes	Specifies whether the image can be used to provision a physical machine, defaults to false. Either this or supportsVirtualMachine must be set to true. Note that this is related to, but not the same as, hypervisor. Some images may not be hypervisors but can be provisioned on a physical machine.
supportsVirtualMachine	**Boolean**	Yes	Yes	Specifies whether the image can be used to provision a virtual machine, defaults to false. Either this or supportsPhysicalMachine must be set to true.
templateName	**String**	Yes	Yes	The VM template to use for this image. Only valid if the type is set to a valid template type such as 'ImageType.LINKED_CLONE'.

Field Name	Type	POST	PUT	Description
type	**ImageType**	Yes	Yes	The type of the image. This property may affect the valid values to use for other fields. See ImageType for more information. (See also: templateName.)
virtualizedImages	**Set<Image>**	Yes	Yes	The set of images available on this hypervisor.

ImageType

Represents an image type, such as stateful or stateless. This is used by provisioning managers and applications to correctly provision and represent the image.

Certain types are only valid for images configured as templates using the Image.templateName field. This currently includes ImageType.LINKED_CLONE and ImageType.FULL_CLONE.

Value	Description
STATEFUL	
STATELESS	
STATELITE	
LINKED_ CLONE	Template type. When this image type is used, the Image.hypervisor field must be set to false, Image.supportsVirtualMachine must be true, and Image.supportsPhysicalMachine must be false.
FULL_ CLONE	Template type. When this image type is used, the Image.hypervisor field must be set to false, Image.supportsVirtualMachine must be true, and Image.supportsPhysicalMachine must be false.

API version 2

Image

An image is used to track the different types of operating systems and hypervisors available in a data center. If the image is a hypervisor, it can contain other images which are the available virtual machines of the hypervisor.

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of this image.
active	**Boolean**	Yes	Yes	If false, the image is flagged as inactive and should not be used. Defaults to true.

Field Name	Type	POST	PUT	Description
extensions	**Map<String, Map>**	Yes	Yes	A map containing maps which represent settings for provisioning managers. Only one extension may be present on an image at a time currently. Valid default provisioning manager specific extensions include 'xcat'. Required properties for 'xcat' when hypervisor is false: • os - The name of the operating system according to xCAT • architecture - The architecture, such as x86_64 • profile - The xCAT profile to use for the image Required properties for 'xcat' when hypervisor is true: • os - The name of the operating system according to xCAT • architecture - The architecture, such as x86_64 • profile - The xCAT profile to use for the image • hvGroupName - The name of the xCAT hypervisor group • vmGroupName - The name of the xCAT VM group
features	**Set<String>**	Yes	Yes	The set of features used by the provisioning manager.

Field Name	Type	POST	PUT	Description
hypervisor	**Boolean**	Yes	Yes	Whether or not the image is a hypervisor. Required during POST. Note that this is related to, but not the same as, supportsPhysicalMachine. Also, when this is false, no virtualizedImages may be specified for an image.
hypervisorType	**String**	Yes	Yes	The type of the hypervisor, which is indicative of the hypervisor technology used in this image. Required if this image is a hypervisor image.
name	**String**	Yes	Yes	The unique human-readable name of this image. Required during POST.
osType	**String**	Yes	Yes	The type of the operating system such as 'Linux' or 'Windows'. Required during POST.
supportsPhysicalMachine	**Boolean**	Yes	Yes	Specifies whether the image can be used to provision a physical machine, defaults to false. Either this or supportsVirtualMachine must be set to true. Note that this is related to, but not the same as, hypervisor. Some images may not be hypervisors but can be provisioned on a physical machine.
supportsVirtualMachine	**Boolean**	Yes	Yes	Specifies whether the image can be used to provision a virtual machine, defaults to false. Either this or supportsPhysicalMachine must be set to true.
templateName	**String**	Yes	Yes	The VM template to use for this image. Only valid if the type is set to a valid template type such as 'ImageType.LINKED_CLONE'.

Field Name	Type	POST	PUT	Description
type	**ImageType**	Yes	Yes	The type of the image. This property may affect the valid values to use for other fields. See ImageType for more information. (See also: templateName.)
virtualizedImages	**Set<Image>**	Yes	Yes	The set of images available on this hypervisor.

ImageType

Represents an image type, such as stateful or stateless. This is used by provisioning managers and applications to correctly provision and represent the image.

Certain types are only valid for images configured as templates using the Image.templateName field. This currently includes ImageType.LINKED_CLONE and ImageType.FULL_CLONE.

Value	Description
STATEFUL	
STATELESS	
STATELITE	
LINKED_ CLONE	Template type. When this image type is used, the Image.hypervisor field must be set to false, Image.supportsVirtualMachine must be true, and Image.supportsPhysicalMachine must be false.
FULL_ CLONE	Template type. When this image type is used, the Image.hypervisor field must be set to false, Image.supportsVirtualMachine must be true, and Image.supportsPhysicalMachine must be false.

Related topics

- **Images** on page 1514

Fields: Job Arrays

See the associated **Job arrays** on page 1523 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`job-arrays`	**Permissions** on page 1571
Hooks filename	`job-arrays.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

JobArray

Job arrays are an easy way to submit many sub-jobs that perform the same work using the same script, but operate on different sets of data. Sub-jobs are the jobs created by an array job and are identified by the array job ID and an index; for example, if 235[1] is an identifier, the number 235 is a job array ID, and 1 is the sub-job.

Field Name	Type	POST	Description
cancellationPolicy	**CancellationPolicyInformation**	Yes	Represents the cancellation policy to use for the job array.
indexRanges	**List<JobArrayIndexRange>**	Yes	The index ranges used to generate the sub-job indices. To use hard-coded values, see indexValues.
indexValues	**List<Long>**	Yes	The index values to use for the sub-jobs. To use ranges, see indexRanges.
jobPrototype	**Job**	Yes	The definition of the job to use for each sub-job.
name	**String**	Yes	The name of the job array. In MWS API version 1, this is stored in the `name` field of the created jobs. In MWS API version 2, this is stored in the `customName` field of the created jobs.
slotLimit	**Long**	Yes	(Optional) The number of sub-jobs in the array that can run at a time.

CancellationPolicyInformation

Job arrays can be canceled based on the success or failure of the first or any sub-job. This class represents the failure policies.

Field Name	Type	POST	Description
anyJob	**CancellationPolicy**	Yes	The cancellation policy based on the result of any sub-job. May be used in combination with firstJob.
firstJob	**CancellationPolicy**	Yes	The cancellation policy based on the result of the first sub-job (array index 1). May be used in combination with anyJob.

CancellationPolicy

This enumeration represents job array cancellation policies, and is to be used in combination with CancellationPolicyInformation.

Value	Description
SUCCESS	Cancels the job array if the specified sub-job succeeds.
FAILURE	Cancels the job array if the specified sub-job fails.

JobArrayIndexRange

Represents information about a job index expression. This is used when creating job arrays only.

Field Name	Type	POST	Description
endIndex	**Long**	Yes	The end of the index range. i.e. `10` for `1-10`.
increment	**Long**	Yes	The increment of the index range, defaults to 1 and must be greater than 0. For a range of 1-10 with an increment of 2, the list of indices will be [1, 3, 5, 7, 9].
startIndex	**Long**	Yes	The start of the index range. i.e. `1` for `1-10`.

Job

This class represents a job in the Moab Workload Manager. A job is a request for compute resources (CPUs, memory, storage) with which the requester can do work for a given amount of time. In an HPC environment, this might be a batch script to perform a Monte Carlo simulation. In a cloud environment, this would be a virtual machine and its associated storage. Moab will evaluate the request and assign the requested resources to the requester based on policies, current demand, and other factors in the data center. A job will also usually have some process that Moab starts automatically at the assigned start time. In an HPC environment, this can be starting a batch script on the assigned nodes. In a cloud environment, this can be starting provisioning processes to create the virtual machine and storage and install software on it.

Field Name	Type	POST	Description
id	**String**	No	The unique identifier of this job. Note: this field is not user-assigned and is generated by the database.

Field Name	Type	POS T	Description
arrayIndex	**Long**	No	If this job is a sub-job of a JobArray, this field contains the index of this job in the array. For example, if this job is `Moab.1 [2]`, the array index would be `2`.
arrayMasterName	**String**	No	If this job is a sub-job of a JobArray, this field contains the name of the job array master. For example, if this job is `Moab.1[2]`, the array master name would be `Moab.1`.
attributes	**Set<String>**	Yes	The list of generic attributes associated with this job.
blocks	**Set<JobBlock>**	No	Reasons the job is blocked from running.
bypassCount	**Integer**	No	The number of times the job has been backfilled.
cancelCount	**Integer**	No	The number of times a job has received a cancel request.
commandFile	**String**	Yes	The name of the job script file (absolute path). If commandFile is set and commandScript is not set, then MWS must have read access to the file. If commandFile and commandScript are both set, then MWS does not read the contents of the file, but it does provide the name of the file to Moab.
commandLineArguments	**String**	Yes	The command line arguments passed to the job script specified by commandFile or commandScript.

Field Name	Type	POST	Description
commandScript	**String**	Yes	The contents of the job script. This field must be Base64-encoded.
completionCode	**Integer**	No	The exit code from this job.
cpuTime	**Long**	No	CPU usage time in seconds as reported by the resource manager.
credentials	**JobCredentials**	Yes	The credentials (user and group, for example) associated with this job.
customName	**String**	Yes	The user-specified name of this job.
dates	**JobDates**	Yes	Various dates associated with this job.
deferCount	**Integer**	No	The number of times a job has been deferred.
dependencies	**Set<JobDependency>**	Yes	Dependencies that must be fulfilled before the job can start.
description	**String**	No	The description of the job. Can be set only in a job template.
duration	**Long**	Yes	The length of time in seconds requested for the job. Note that it is possible to set duration to "INFINITY" if the AllowInfiniteJobs flag is set on the scheduler in the moab.cfg.
durationActive	**Long**	No	The length of time in seconds the job has been active or running.

Field Name	Type	POST	Description
durationQueued	**Long**	No	The length of time in seconds the job has been eligible to run in the queue.
durationRemaining	**Long**	No	An estimate of the time remaining, in seconds, before the job will complete.
durationSuspended	**Long**	No	The length of time in seconds the job has been suspended.
emailNotifyAddresses	**Set<String>**	Yes	The list of addresses to whom email is sent by the execution server.
emailNotifyTypes	**Set<JobEmailNotifyType>**	Yes	The list of email notify types attached to the job.
environmentRequested	**Boolean**	Yes	Setting this field to true tells the Moab Workload Manager to set various variables, if populated, in the job's environment.
environmentVariables	**Map<String, Map>**	Yes	The list of environment variables for this job.
epilogScript	**String**	Yes	The path to the TORQUE epilog script.
flags	**Set<JobFlag>**	Yes	The flags that are set on this job.
holdDate	**Date**	No	The date the most recent hold was placed on the job.
holdReason	**JobHoldReason**	No	The reason the job is on hold.
holds	**Set<JobHoldType>**	Yes	The holds that are set on the job. The "User" hold type is valid during POST.

Field Name	Type	POST	Description
initialWorkingDirectory	**String**	Yes	The path to the directory in which the job will be started.
isActive	**Boolean**	No	True if the job is active, false if the job is complete.
jobGroup	**String**	Yes	The job group to which this job belongs (different from credentials.group).
masterNode	**DomainProxy**	No	The first node in the list of allocated nodes for this job. For TORQUE jobs, this represents the "mother superior."
memorySecondsDedicated	**Double**	No	The memory seconds dedicated to the job as reported by its resource manager. Not all resource managers provide this information.
memorySecondsUtilized	**Double**	No	The memory seconds utilized by the job as reported by its resource manager. Not all resource managers provide this information.
messages	**Set<Message>**	No	The list of messages associated with the job. The "message" field is valid during PUT.
migrateCount	**Integer**	No	The number of times the job has been migrated.
minimumPreemptTime	**Long**	No	The minimum length of time, in seconds, an active job must be running before it is eligible for preemption.

Field Name	Type	POST	Description
mwmName	**String**	No	The name of the Moab Workload Manager instance that owns this job.
name	**String**	No	The name of this job. This name is unique *per instance* of Moab Workload Manager (i.e. not globally).
nodesExcluded	**Set<DomainProxy>**	Yes	The list of nodes that should not be considered for this job.
nodesRequested	**Set<DomainProxy>**	Yes	The exact set, superset, or subset of nodes on which this job must run. (See also: nodesRequestedPolicy.)
nodesRequestedPolicy	**JobHostListMode**	Yes	Indicates an exact set, superset, or subset of nodes on which the job must run. Only relevant if nodesRequested is provided. (See also: nodesRequested.)
partitionAccessList	**Set<String>**	No	The list of partitions that this job can access.
partitionAccessListRequested	**Set<String>**	Yes	The list of partitions that this job has requested.
preemptCount	**Integer**	No	The number of times the job has been preempted.
priorities	**JobPriority**	Yes	The list of priorities for the job.
processorSecondsDedicated	**Double**	No	The processor seconds dedicated to the job as reported by its resource manager. Not all resource managers provide this information.

Field Name	Type	POST	Description
processorSecondsLimit	**Double**	No	The limit for processorSecondsUtilized.
processorSecondsUtilized	**Double**	No	The processor seconds utilized by the job as reported by its resource manager. Not all resource managers provide this information.
prologScript	**String**	Yes	The path to the TORQUE prolog script.
queueStatus	**JobQueueStatus**	No	The status of the job in its queue.
rejectPolicies	**Set<JobRejectPolicy>**	No	The list of policies enabled when a job is rejected.
requirements	**Set<JobRequirement>**	Yes	The list of items required for this job to run. Only JobRequirement.features is valid during PUT.
reservationRequested	**DomainProxy**	Yes	The reservation that the job requested.
resourceFailPolicy	**JobResourceFailPolicyType**	Yes	The policy that dictates what should happen to the job if it is running and at least one of the resources it is using fails.
resourceManagerExtension	**String**	Yes	If provided during POST, this string will be added to the resource manager extension section of the job submission. Example: "bandwidth=120;queuejob=false" Note that the delimiter between resourceManagerExtension elements is the semicolon.

Field Name	Type	POST	Description
resourceManagers	Set<ResourceManager>	No	The list of resource managers associated with this job.
rmStandardErrorFilePath	**String**	No	The path to the remote file containing the standard error of the job.
rmStandardOutputFilePath	**String**	No	The path to the remote file containing the standard output of the job.
standardErrorFilePath	**String**	Yes	The path to the file containing the standard error of the job.
standardOutputFilePath	**String**	Yes	The path to the file containing the standard output of the job.
startCount	**Integer**	No	The number of times the job has been started.
states	JobStateInformation	No	Information about the state of the job.
submitHost	**String**	Yes	The host from which the job was submitted.
systemJobAction	**String**	No	The action the system job will take.
systemJobType	JobSystemJobType	No	The type of system job. In the Moab Cloud Suite, this will usually be "vmtracking" or "generic."
targetedJobAction	JobActionType	No	The action that this job is performing on another job.

Field Name	Type	POST	Description
targetedJobName	**String**	No	The name of the job on which this job is performing the targetedJobAction.
templates	**Set<DomainProxy>**	Yes	The list of all job templates to be set on this job.
triggers	**Set<String>**	No	The list of triggers associated with this job.
variables	**Map<String, Map>**	Yes	The list of variables that this job owns or sets on completion.
virtualContainers	**Set<DomainProxy>**	Yes	When submitting this job, add it to the specified existing virtual container. Valid during POST, but only one virtual container can be specified.
virtualMachines	**Set<DomainProxy>**	No	The list of virtual machines that are allocated to this job.
vmUsagePolicy	**VMUsagePolicy**	Yes	The requested Virtual Machine Usage Policy for this job.

JobBlock

Field Name	Type	POST	Description
category	**JobBlockCategory**	No	
message	**String**	No	
type	**JobBlockType**	No	

JobBlockCategory

Value	Description
depend	
jobBlock	
migrate	

JobBlockType

Value	Description
ActivePolicy	
BadUser	
Dependency	
EState	
FairShare	
Hold	
IdlePolicy	
LocalPolicy	
NoClass	
NoData	
NoResource	
NoTime	
PartitionAccess	

Value	Description
Priority	
RMSubmissionFailure	
StartDate	
State	
SysLimits	

JobCredentials

Moab Workload Manager supports the concept of credentials, which provide a means of attributing policy and resource access to entities such as users and groups. These credentials allow specification of job ownership, tracking of resource usage, enforcement of policies, and many other features.

Field Name	Type	POST	Description
account	**String**	Yes	The account credential is also referred to as the project. This credential is generally associated with a group of users along the lines of a particular project for accounting and billing purposes.
group	**String**	Yes	The group credential represents an aggregation of users. User-to-group mappings are often specified by the operating system or resource manager and typically map to a user's UNIX group ID. However, user-to-group mappings may also be provided by a security and identity management service, or you can specify such directly within Moab.
jobClass	**String**	Yes	The concept of the class credential is derived from the resource manager class or queue object. Classes differ from other credentials in that they more directly impact job attributes. In standard HPC usage, a user submits a job to a class and this class imposes a number of factors on the job. The attributes of a class may be specified within the resource manager or directly within Moab.

Field Name	Type	POST	Description
qos	**String**	No	The quality of service assigned to this job. The concept of a quality of service (QoS) credential is unique to Moab and is not derived from any underlying concept or peer service. In most cases, the QoS credential is used to allow a site to set up a selection of service levels for end-users to choose from on a long-term or job-by-job basis. QoS's differ from other credentials in that they are centered around special access where this access may allow use of additional services, additional resources, or improved responsiveness. Unique to this credential, organizations may also choose to apply different charge rates to the varying levels of service available within each QoS. As QoS is an internal credential, all QoS configuration occurs within Moab.
qosRequested	**String**	Yes	The quality of service requested for this job.
user	**String**	Yes	The user credential is the fundamental credential within a workload manager; each job requires an association with exactly one user. In fact, the user credential is the only required credential in Moab; all others are optional. In most cases, the job's user credential is configured within or managed by the operating system itself, although Moab may be configured to obtain this information from an independent security and identity management service.

JobDates

Field Name	Type	POST	Description
completedDate	**Date**	No	
createdDate	**Date**	No	
deadlineDate	**Date**	Yes	The deadline for completion of the job.
dispatchedDate	**Date**	No	
earliestRequestedStartDate	**Date**	Yes	The job will start no sooner than this date.
earliestStartDate	**Date**	No	

Field Name	Type	POST	Description
eligibleDate	**Date**	No	
lastCanceledDate	**Date**	No	
lastChargedDate	**Date**	No	
lastPreemptedDate	**Date**	No	
lastUpdatedDate	**Date**	No	
startDate	**Date**	No	
submitDate	**Date**	No	
terminationDate	**Date**	No	

JobDependency

Field Name	Type	POST	Description
name	**String**	Yes	The name of the object on on which the job is dependent.
type	**JobDependencyType**	Yes	The type of job dependency. Only set is valid for POST.
value	**String**	No	

JobDependencyType

Represents the type of a job dependency. For now, only the "set" type is supported.

Value	Description
set	

JobEmailNotifyType

Value	Description
JobStart	
JobEnd	
JobFail	
All	

JobFlag

This enumeration specifies the flag types of a job.

Value	Description
NONE	
BACKFILL	The job is using backfill to run.
COALLOC	The job can use resources from multiple resource managers and partitions.
ADVRES	The job requires the use of a reservation.
NOQUEUE	The job will attempt to execute immediately or fail.
ARRAYJOB	The job is part of a job array.
ARRAYJOBPARLOCK	This array job will only run in one partition.
ARRAYJOBPARSPAN	This array job will span partitions (default).
ARRAYMASTER	This job is the master of a job array.
BESTEFFORT	The job will succeed if even partial resources are available.
RESTARTABLE	The job is restartable.
SUSPENDABLE	The job is suspendable.

Value	Description
HASPREEMPTED	This job preempted other jobs to start.
PREEMPTEE	The job is a preemptee and therefore can be preempted by other jobs.
PREEMPTOR	The job is a preemptor and therefore can preempt other jobs.
RSVMAP	The job is based on a reservation.
SPVIOLATION	The job was started with a soft policy violation.
IGNNODEPOLICIES	The job will ignore node policies.
IGNPOLICIES	The job will ignore idle, active, class, partition, and system policies.
IGNNODESTATE	The job will ignore node state in order to run.
IGNIDLEJOBRSV	The job can ignore idle job reservations. The job granted access to all idle job reservations.
INTERACTIVE	The job needs to interactive input from the user to run.
FSVIOLATION	The job was started with a fairshare violation.
GLOBALQUEUE	The job is directly submitted without doing any authentication.
NORESOURCES	The job is a system job that does not need any resources.
NORMSTART	The job will not query a resource manager to run.
CLUSTERLOCKED	The job is locked into the current cluster and cannot be migrated elsewhere. This is for grid mode.
FRAGMENT	The job can be run across multiple nodes in individual chunks.
SYSTEMJOB	The job is a system job which simply runs on the same node that Moab is running on. This is usually used for running scripts and other executables in workflows.

Value	Description
ADMINSETIGNPOLICIES	The IGNPOLICIES flag was set by an administrator.
EXTENDSTARTWALLTIME	The job duration (walltime) was extended at job start.
SHAREDMEM	The job will share its memory across nodes.
BLOCKEDBYGRES	The job's generic resource requirement caused the job to start later.
GRESONLY	The job is requesting only generic resources, no compute resources.
TEMPLATESAPPLIED	The job has had all applicable templates applied to it.
META	META job, just a container around resources.
WIDERSVSEARCHALGO	This job prefers the wide search algorithm.
VMTRACKING	The job is a VMTracking job for an externally-created VM (via job template).
DESTROYTEMPLATESUBMITTED	A destroy job has already been created from the template for this job.
PROCSPECIFIED	The job requested processors on the command line.
CANCELONFIRSTFAILURE	Cancel job array on first array job failure.
CANCELONFIRSTSUCCESS	Cancel job array on first array job success.
CANCELONANYFAILURE	Cancel job array on any array job failure.
CANCELONANYSUCCESS	Cancel job array on any array job success.
CANCELONEXITCODE	Cancel job array on a specific exit code.
NOVMMIGRATE	Do not migrate the virtual machine that this job sets up.
PURGEONSUCCESSONLY	Only purge the job if it completed successfully

JobHoldReason

Value	Description
Admin	
NoResources	
SystemLimitsExceeded	
BankFailure	
CannotDebitAccount	
InvalidAccount	
RMFailure	
RMReject	
PolicyViolation	
CredAccess	
CredHold	
PreReq	
Data	
Security	
MissingDependency	

JobHoldType

Value	Description
User	

Value	Description
System	
Batch	
Defer	
All	

DomainProxy

A reference to an object contained within an object. For example, a Virtual Machine object contains a reference to the Node on which it is running. That reference is represented by this class.

Field Name	Type	POST	Description
name	**String**	Yes	The name of the object.

Message

Field Name	Type	POST	Description
count	**Integer**	No	The number of times this message has occurred.
createdDate	**Date**	No	The date this message was created.
expireDate	**Date**	No	The date this message expires.
message	**String**	No	The message itself.

JobHostListMode

Value	Description
superset	
subset	

Value	Description
exactset	

JobPriority

Field Name	Type	POST	Description
run	**Long**	No	
start	**Long**	No	
system	**Long**	No	
user	**Long**	Yes	The user-requested priority for the job. By default, the range is between -1024 and 0. To enable priority range from -1024 to +1023, set `ENABLEPOSUSERPRIORITY` in the `moab.cfg` file.

JobQueueStatus

Value	Description
active	
blocked	
completed	
eligible	

JobRejectPolicy

Value	Description
CANCEL	
HOLD	

Value	Description
IGNORE	
MAIL	
RETRY	

JobRequirement

Field Name	Type	POST	Description
architecture	**String**	Yes	The architecture required by the job.
attributes	**Map<String, JobRequirementAttribute>**	Yes	Required node attributes with version number support.
features	**Set<String>**	No	The list of node features the job is scheduled against.
featuresExcluded	**Set<String>**	Yes	Excluded node features. That is, do not select nodes with these features. (See also: featuresExcludedMode.)
featuresExcludedMode	**JobRequirementFeaturesMode**	Yes	Indicates whether excluded features should be AND'ed or OR'd. The default is AND. Only relevant if featuresExcluded is provided. (See also: featuresExcluded.)
featuresRequested	**Set<String>**	Yes	Requested node features. (See also: featuresRequestedMode.)
featuresRequestedMode	**JobRequirementFeaturesMode**	Yes	Indicates whether requested features should be AND'ed or OR'd. The default is AND. Only relevant if featuresRequested is provided. (See also: featuresRequested.)
image	**String**	Yes	The image required by the job.

Field Name	Type	POST	Description
index	**Integer**	No	The index of the requirement, starting with 0.
metrics	**Map<String, Double>**	No	Generic metrics associated with the job as reported by the resource manager.
nodeAccessPolicy	[NodeAccessPolicy]()	Yes	Specifies how node resources should be accessed. Note: If the job requirements array has more than one element that contains nodeAccessPolicy, only the first occurrence will be used.
nodeAllocationPolicy	[NodeAllocationPolicy]()	Yes	Specifies how node resources should be selected and allocated to the job. Note: If the job requirements array has more than one element that contains nodeAllocationPolicy, only the first occurrence will be used.
nodeCount	**Integer**	Yes	The number of nodes required by the job.
nodeSet	**String**	Yes	The requested node set of the job. This must follow the format `SETSELECTION:SETTYPE[:SETLIST]` • SETSELECTION - `ANYOF`, `ONEOF`, or `FIRSTOF` • SETTYPE - `FEATURE` or `VARATTR` • SETLIST - For `FEATURE`, a comma-separated list of features. For `VARATTR`, a key=value pair. Examples: • `ONEOF:FEATURE:fastos,hiprio,bigmem` • `FIRSTOF:VARATTR:datacenter=Provo:datacenter=SaltLake`
nodes	[Set<AllocatedNode>]()	No	Nodes that have been allocated to meet this requirement.

Field Name	Type	POST	Description
reservation	[DomainProxy]()	No	The allocated reservation (assigned after the job has a reservation).
resourcesPerTask	[Map<String, JobResource>]()	Yes	Contains requirements for disk, memory, processors, swap, and generic resources. For disk, memory, and swap, the unit is MB. For each resource, the "dedicated" field can be set during POST.
taskCount	**Integer**	Yes	The number of tasks (processors) required by this job.
tasksPerNode	**Integer**	Yes	The number of tasks to map to each node.

JobRequirementAttribute

Field Name	Type	POST	Description
comparator	**String**	Yes	The comparison operator. Valid values: • >= - Greater than or equal to • > - Greater than • <= - Less than • < - Less than • %= - Equals • %! - Not equals • Null - Defaults to %= • = - (Deprecated) Equivalent to %=

Field Name	Type	POST	Description
displayValue	**String**	Yes	The display value for the required attribute.
restriction	**JobRequirementAttributeRestriction**	Yes	The restriction of this attribute. May be null, but defaults to JobRequirementAttributeRestriction.must.
value	**String**	Yes	The value of the required attribute. During POST, if value is missing, blank, or null, do not provide a comparator.

JobRequirementAttributeRestriction

Represents a restriction for a job requirement attribute.

Value	Description
must	

JobRequirementFeaturesMode

Value	Description
OR	
AND	

NodeAccessPolicy

This enumeration describes how node resources will be shared by various tasks.

Value	Description
NONE	
SHARED	Tasks from any combination of jobs may utilize available resources.

Value	Description
SHAREDONLY	Only jobs requesting shared node access may utilize available resources.
SINGLEJOB	Tasks from a single job may utilize available resourceses.
SINGLETASK	A single task from a single job may run on the node.
SINGLEUSER	Tasks from any jobs owned by the same user may utilize available resources.
UNIQUEUSER	Any number of tasks from a single job may allocate resources from a node but only if the user has no other jobs running on that node.
SINGLEGROUP	Any number of tasks from the same group may utilize node.
SINGLEACCOUNT	Any number of tasks from the same account may utilize node.

NodeAllocationPolicy

Node Allocation enumeration.

Value	Description
FIRSTSET	
MINGLOBAL	
MINLOCAL	
PLUGIN	
NONE	No node allocation policy is specified. Moab defaults to MINRESOURCE when this is the case.
FIRSTAVAILABLE	Simple first come, first served algorithm where nodes are allocated in the order they are presented by the resource manager. This is a very simple, very fast algorithm.

Value	Description
LASTAVAILABLE	This algorithm selects resources so as to minimize the amount of time after the job and before the trailing reservation. This algorithm is a best fit in time algorithm which minimizes the impact of reservation based node-time fragmentation. It is useful in systems where a large number of reservations (job, standing, or administrative) are in place.
MINRESOURCE	This algorithm prioritizes nodes according to the configured resources on each node. Those nodes with the fewest configured resources which still meet the job's resource constraints are selected.
CPULOAD	Nodes are selected which have the maximum amount of available, unused cpu power, i.e. [# of CPU's] - [CPU load]. Good algorithm for timesharing node systems. This algorithm is only applied to jobs starting immediately. For the purpose of future reservations, the **MINRESOURCE** algorithm is used.
LOCAL	This will call the locally created **contrib** node allocation algorithm.
CONTIGUOUS	This algorithm will allocate nodes in contiguous (linear) blocks as required by the Compaq RMS system.
MAXBALANCE	This algorithm will attempt to allocate the most 'balanced' set of nodes possible to a job. In most cases, but not all, the metric for balance of the nodes is node speed. Thus, if possible, nodes with identical speeds will be allocated to the job. If identical speed nodes cannot be found, the algorithm will allocate the set of nodes with the minimum node speed 'span' or range.
PRIORITY	This algorithm allows a site to specify the priority of various static and dynamic aspects of compute nodes and allocate them with preference for higher priority nodes. It is highly flexible allowing node attribute and usage information to be combined with reservation affinity.
FASTEST	This algorithm will select nodes in 'fastest node first' order. Nodes will be selected by node speed if specified. If node speed is not specified, nodes will be selected by processor speed. If neither is specified, nodes will be selected in a random order.
PROCESSORLOAD	Alias for CPULOAD.

Value	Description
NODESPEED	Alias for FASTEST.
INREPORTEDORDER	Alias for FIRSTAVAILABLE.
INREVERSEREPORTEDORDER	Alias for LASTAVAILABLE.
CUSTOMPRIORITY	Alias for PRIORITY.
PROCESSORSPEEDBALANCE	Alias for MAXBALANCE.
MINIMUMCONFIGUREDRESOURCES	Alias for MINRESOURCE.

AllocatedNode

Field Name	Type	POST	Description
name	**String**	No	
taskCount	**Integer**	No	

JobResource

Represents counts of dedicated and utilized resources.

Field Name	Type	POST	Description
dedicated	**Integer**	No	The amount of this resource that has been allocated for running workload.
utilized	**Integer**	No	The amount of this resource that is currently reported as utilized by resource managers.

JobResourceFailPolicyType

Value	Description
CANCEL	
FAIL	
HOLD	
IGNORE	
NOTIFY	
REQUEUE	

ResourceManager

Field Name	Type	POST	Description
isDestination	Boolean	No	
isSource	Boolean	No	
jobName	String	No	
name	String	No	

JobStateInformation

Field Name	Type	POST	Description
state	JobState	No	
stateExpected	JobState	No	
stateLastUpdatedDate	Date	No	
subState	JobSubState	No	

JobState

Value	Description
Idle	
Starting	
Running	
Removed	
Completed	
Hold	
Deferred	
Vacated	
NotQueued	
Unknown	
Staging	
Suspended	
Blocked	

JobSubState

Value	Description
Epilogue	
Migrated	
Preempted	

Value	Description
Prologue	

JobSystemJobType

Value	Description
generic	
osprovision	
osprovision2	
poweroff	
poweron	
reset	
storage	
vmmap	
vmmigrate	
vmtracking	

JobActionType

Value	Description
DESTROY	
MIGRATE	
MODIFY	

VMUsagePolicy

This enumeration describes the virtual machine requirements of a job

Value	Description
REQUIREPM	Requires a physical machine.
PREFPM	Prefers a physical machine.
CREATEVM	Creates a virtual machine.
CREATEPERSISTENTVM	Creates a virtual machine that doesn't go away after the job is done.
REQUIREVM	Requires a virtual machine.
PREFVM	Prefers a virtual machine.

API version 2

JobArray

Job arrays are an easy way to submit many sub-jobs that perform the same work using the same script, but operate on different sets of data. Sub-jobs are the jobs created by an array job and are identified by the array job ID and an index; for example, if 235[1] is an identifier, the number 235 is a job array ID, and 1 is the sub-job.

Field Name	Type	POST	Description
cancellationPolicy	CancellationPolicyInformation	Yes	Represents the cancellation policy to use for the job array.
indexRanges	List<JobArrayIndexRange>	Yes	The index ranges used to generate the sub-job indices. To use hard-coded values, see indexValues.
indexValues	**List<Long>**	Yes	The index values to use for the sub-jobs. To use ranges, see indexRanges.
jobPrototype	Job	Yes	The definition of the job to use for each sub-job.
name	**String**	Yes	The name of the job array. In MWS API version 1, this is stored in the `name` field of the created jobs. In MWS API version 2, this is stored in the `customName` field of the created jobs.
slotLimit	**Long**	Yes	(Optional) The number of sub-jobs in the array that can run at a time.

CancellationPolicyInformation

Job arrays can be canceled based on the success or failure of the first or any sub-job. This class represents the failure policies.

Field Name	Type	POST	Description
anyJob	CancellationPolicy	Yes	The cancellation policy based on the result of any sub-job. May be used in combination with firstJob.
firstJob	CancellationPolicy	Yes	The cancellation policy based on the result of the first sub-job (array index 1). May be used in combination with anyJob.

CancellationPolicy

This enumeration represents job array cancellation policies, and is to be used in combination with CancellationPolicyInformation.

Value	Description
SUCCESS	Cancels the job array if the specified sub-job succeeds.
FAILURE	Cancels the job array if the specified sub-job fails.

JobArrayIndexRange

Represents information about a job index expression. This is used when creating job arrays only.

Field Name	Type	POST	Description
endIndex	**Long**	Yes	The end of the index range. i.e. `10` for `1-10`.
increment	**Long**	Yes	The increment of the index range, defaults to 1 and must be greater than 0. For a range of 1-10 with an increment of 2, the list of indices will be [1, 3, 5, 7, 9].
startIndex	**Long**	Yes	The start of the index range. i.e. `1` for `1-10`.

Job

This class represents a job in the Moab Workload Manager. A job is a request for compute resources (CPUs, memory, storage) with which the requester can do work for a given amount of time. In an HPC environment, this might be a batch script to perform a Monte Carlo simulation. In a cloud environment, this would be a virtual machine and its associated storage. Moab will evaluate the request and assign the requested resources to the requester based on policies, current demand, and other factors in the data center. A job will also usually have some process that Moab starts automatically at the assigned start time. In an HPC environment, this can be starting a batch script on the assigned nodes. In a cloud environment, this can be starting provisioning processes to create the virtual machine and storage and install software on it.

Field Name	Type	POST	Description
id	**String**	No	The unique identifier of this job. Note: this field is not user-assigned and is generated by the database.

Field Name	Type	POST	Description
arrayIndex	**Long**	No	If this job is a sub-job of a JobArray, this field contains the index of this job in the array. For example, if this job is `Moab.1[2]`, the array index would be `2`.
arrayMasterName	**String**	No	If this job is a sub-job of a JobArray, this field contains the name of the job array master. For example, if this job is `Moab.1[2]`, the array master name would be `Moab.1`.
attributes	**Set<String>**	Yes	The list of generic attributes associated with this job.
blocks	**Set<JobBlock>**	No	Reasons the job is blocked from running.
bypassCount	**Integer**	No	The number of times the job has been backfilled.
cancelCount	**Integer**	No	The number of times a job has received a cancel request.
commandFile	**String**	Yes	The name of the job script file (absolute path). If commandFile is set and commandScript is not set, then MWS must have read access to the file. If commandFile and commandScript are both set, then MWS does not read the contents of the file, but it does provide the name of the file to Moab.
commandLineArguments	**String**	Yes	The command line arguments passed to the job script specified by commandFile or commandScript.

Field Name	Type	POST	Description
commandScript	**String**	Yes	The contents of the job script. This field must be Base64-encoded.
completionCode	**Integer**	No	The exit code from this job.
cpuTime	**Long**	No	CPU usage time in seconds as reported by the resource manager.
credentials	**JobCredentials**	Yes	The credentials (user and group, for example) associated with this job.
customName	**String**	Yes	The user-specified name of this job.
dates	**JobDates**	Yes	Various dates associated with this job.
deferCount	**Integer**	No	The number of times a job has been deferred.
dependencies	**Set<JobDependency>**	Yes	Dependencies that must be fulfilled before the job can start.
description	**String**	No	The description of the job. Can be set only in a job template.
duration	**Long**	Yes	The length of time in seconds requested for the job. Note that it is possible to set duration to "INFINITY" if the AllowInfiniteJobs flag is set on the scheduler in the moab.cfg.
durationActive	**Long**	No	The length of time in seconds the job has been active or running.

Field Name	Type	POST	Description
durationQueued	**Long**	No	The length of time in seconds the job has been eligible to run in the queue.
durationRemaining	**Long**	No	An estimate of the time remaining, in seconds, before the job will complete.
durationSuspended	**Long**	No	The length of time in seconds the job has been suspended.
emailNotifyAddresses	**Set<String>**	Yes	The list of addresses to whom email is sent by the execution server.
emailNotifyTypes	**Set<JobEmailNotifyType>**	Yes	The list of email notify types attached to the job.
environmentRequested	**Boolean**	Yes	Setting this field to true tells the Moab Workload Manager to set various variables, if populated, in the job's environment.
environmentVariables	**Map<String, Map>**	Yes	The list of environment variables for this job.
epilogScript	**String**	Yes	The path to the TORQUE epilog script.
flags	**Set<JobFlag>**	Yes	The flags that are set on this job.
holdDate	**Date**	No	The date the most recent hold was placed on the job.
holdReason	**JobHoldReason**	No	The reason the job is on hold.
holds	**Set<JobHoldType>**	Yes	The holds that are set on the job. The "User" hold type is valid during POST.

Field Name	Type	POST	Description
initialWorkingDirectory	**String**	Yes	The path to the directory in which the job will be started.
isActive	**Boolean**	No	True if the job is active, false if the job is complete.
jobGroup	**String**	Yes	The job group to which this job belongs (different from credentials.group).
masterNode	**DomainProxy**	No	The first node in the list of allocated nodes for this job. For TORQUE jobs, this represents the "mother superior."
memorySecondsDedicated	**Double**	No	The memory seconds dedicated to the job as reported by its resource manager. Not all resource managers provide this information.
memorySecondsUtilized	**Double**	No	The memory seconds utilized by the job as reported by its resource manager. Not all resource managers provide this information.
messages	**Set<Message>**	No	The list of messages associated with the job. The "message" field is valid during PUT.
migrateCount	**Integer**	No	The number of times the job has been migrated.
minimumPreemptTime	**Long**	No	The minimum length of time, in seconds, an active job must be running before it is eligible for preemption.

Field Name	Type	POST	Description
mwmName	**String**	No	The name of the Moab Workload Manager instance that owns this job.
name	**String**	No	The name of this job. This name is unique *per instance* of Moab Workload Manager (i.e. not globally).
nodesExcluded	**Set<DomainProxy>**	Yes	The list of nodes that should not be considered for this job.
nodesRequested	**Set<DomainProxy>**	Yes	The exact set, superset, or subset of nodes on which this job must run. (See also: nodesRequestedPolicy.)
nodesRequestedPolicy	**JobHostListMode**	Yes	Indicates an exact set, superset, or subset of nodes on which the job must run. Only relevant if nodesRequested is provided. (See also: nodesRequested.)
partitionAccessList	**Set<String>**	No	The list of partitions that this job can access.
partitionAccessListRequested	**Set<String>**	Yes	The list of partitions that this job has requested.
preemptCount	**Integer**	No	The number of times the job has been preempted.
priorities	**JobPriority**	Yes	The list of priorities for the job.
processorSecondsDedicated	**Double**	No	The processor seconds dedicated to the job as reported by its resource manager. Not all resource managers provide this information.

Field Name	Type	POST	Description
processorSecondsLimit	**Double**	No	The limit for processorSecondsUtilized.
processorSecondsUtilized	**Double**	No	The processor seconds utilized by the job as reported by its resource manager. Not all resource managers provide this information.
prologScript	**String**	Yes	The path to the TORQUE prolog script.
queueStatus	**JobQueueStatus**	No	The status of the job in its queue.
rejectPolicies	**Set<JobRejectPolicy>**	No	The list of policies enabled when a job is rejected.
requirements	**Set<JobRequirement>**	Yes	The list of items required for this job to run. Only JobRequirement.features is valid during PUT.
reservationRequested	**DomainProxy**	Yes	The reservation that the job requested.
resourceFailPolicy	**JobResourceFailPolicyType**	Yes	The policy that dictates what should happen to the job if it is running and at least one of the resources it is using fails.
resourceManagerExtension	**String**	Yes	If provided during POST, this string will be added to the resource manager extension section of the job submission. Example: "bandwidth=120;queuejob=false" Note that the delimiter between resourceManagerExtension elements is the semicolon.

Field Name	Type	POST	Description
resourceManagers	Set<ResourceManager>	No	The list of resource managers associated with this job.
rmStandardErrorFilePath	**String**	No	The path to the remote file containing the standard error of the job.
rmStandardOutputFilePath	**String**	No	The path to the remote file containing the standard output of the job.
standardErrorFilePath	**String**	Yes	The path to the file containing the standard error of the job.
standardOutputFilePath	**String**	Yes	The path to the file containing the standard output of the job.
startCount	**Integer**	No	The number of times the job has been started.
states	JobStateInformation	No	Information about the state of the job.
submitHost	**String**	Yes	The host from which the job was submitted.
systemJobAction	**String**	No	The action the system job will take.
systemJobType	JobSystemJobType	No	The type of system job. In the Moab Cloud Suite, this will usually be "vmtracking" or "generic."
targetedJobAction	JobActionType	No	The action that this job is performing on another job.

Field Name	Type	POST	Description
targetedJobName	**String**	No	The name of the job on which this job is performing the targetedJobAction.
templates	**Set<DomainProxy>**	Yes	The list of all job templates to be set on this job.
triggers	**Set<String>**	No	The list of triggers associated with this job.
variables	**Map<String, Map>**	Yes	The list of variables that this job owns or sets on completion.
virtualContainers	**Set<DomainProxy>**	Yes	When submitting this job, add it to the specified existing virtual container. Valid during POST, but only one virtual container can be specified.
virtualMachines	**Set<DomainProxy>**	No	The list of virtual machines that are allocated to this job.
vmUsagePolicy	**VMUsagePolicy**	Yes	The requested Virtual Machine Usage Policy for this job.

JobBlock

Field Name	Type	POST	Description
category	**JobBlockCategory**	No	
message	**String**	No	
type	**JobBlockType**	No	

JobBlockCategory

Value	Description
depend	
jobBlock	
migrate	

JobBlockType

Value	Description
ActivePolicy	
BadUser	
Dependency	
EState	
FairShare	
Hold	
IdlePolicy	
LocalPolicy	
NoClass	
NoData	
NoResource	
NoTime	
PartitionAccess	

Value	Description
Priority	
RMSubmissionFailure	
StartDate	
State	
SysLimits	

JobCredentials

Moab Workload Manager supports the concept of credentials, which provide a means of attributing policy and resource access to entities such as users and groups. These credentials allow specification of job ownership, tracking of resource usage, enforcement of policies, and many other features.

Field Name	Type	POST	Description
account	**String**	Yes	The account credential is also referred to as the project. This credential is generally associated with a group of users along the lines of a particular project for accounting and billing purposes.
group	**String**	Yes	The group credential represents an aggregation of users. User-to-group mappings are often specified by the operating system or resource manager and typically map to a user's UNIX group ID. However, user-to-group mappings may also be provided by a security and identity management service, or you can specify such directly within Moab.
jobClass	**String**	Yes	The concept of the class credential is derived from the resource manager class or queue object. Classes differ from other credentials in that they more directly impact job attributes. In standard HPC usage, a user submits a job to a class and this class imposes a number of factors on the job. The attributes of a class may be specified within the resource manager or directly within Moab.

Field Name	Type	POST	Description
qos	**String**	No	The quality of service assigned to this job. The concept of a quality of service (QoS) credential is unique to Moab and is not derived from any underlying concept or peer service. In most cases, the QoS credential is used to allow a site to set up a selection of service levels for end-users to choose from on a long-term or job-by-job basis. QoS's differ from other credentials in that they are centered around special access where this access may allow use of additional services, additional resources, or improved responsiveness. Unique to this credential, organizations may also choose to apply different charge rates to the varying levels of service available within each QoS. As QoS is an internal credential, all QoS configuration occurs within Moab.
qosRequested	**String**	Yes	The quality of service requested for this job.
user	**String**	Yes	The user credential is the fundamental credential within a workload manager; each job requires an association with exactly one user. In fact, the user credential is the only required credential in Moab; all others are optional. In most cases, the job's user credential is configured within or managed by the operating system itself, although Moab may be configured to obtain this information from an independent security and identity management service.

JobDates

Field Name	Type	POST	Description
completedDate	**Date**	No	
createdDate	**Date**	No	
deadlineDate	**Date**	Yes	The deadline for completion of the job.
dispatchedDate	**Date**	No	
earliestRequestedStartDate	**Date**	Yes	The job will start no sooner than this date.
earliestStartDate	**Date**	No	

Field Name	Type	POST	Description
eligibleDate	**Date**	No	
lastCanceledDate	**Date**	No	
lastChargedDate	**Date**	No	
lastPreemptedDate	**Date**	No	
lastUpdatedDate	**Date**	No	
startDate	**Date**	No	
submitDate	**Date**	No	
terminationDate	**Date**	No	

JobDependency

Field Name	Type	POST	Description
name	**String**	Yes	The name of the object on on which the job is dependent.
type	**JobDependencyType**	Yes	The type of job dependency. Only set is valid for POST.
value	**String**	No	

JobDependencyType

Represents the type of a job dependency. For now, only the "set" type is supported.

Value	Description
set	

JobEmailNotifyType

Value	Description
JobStart	
JobEnd	
JobFail	
All	

JobFlag

This enumeration specifies the flag types of a job.

Value	Description
NONE	
BACKFILL	The job is using backfill to run.
COALLOC	The job can use resources from multiple resource managers and partitions.
ADVRES	The job requires the use of a reservation.
NOQUEUE	The job will attempt to execute immediately or fail.
ARRAYJOB	The job is part of a job array.
ARRAYJOBPARLOCK	This array job will only run in one partition.
ARRAYJOBPARSPAN	This array job will span partitions (default).
ARRAYMASTER	This job is the master of a job array.
BESTEFFORT	The job will succeed if even partial resources are available.
RESTARTABLE	The job is restartable.
SUSPENDABLE	The job is suspendable.

Value	Description
HASPREEMPTED	This job preempted other jobs to start.
PREEMPTEE	The job is a preemptee and therefore can be preempted by other jobs.
PREEMPTOR	The job is a preemptor and therefore can preempt other jobs.
RSVMAP	The job is based on a reservation.
SPVIOLATION	The job was started with a soft policy violation.
IGNNODEPOLICIES	The job will ignore node policies.
IGNPOLICIES	The job will ignore idle, active, class, partition, and system policies.
IGNNODESTATE	The job will ignore node state in order to run.
IGNIDLEJOBRSV	The job can ignore idle job reservations. The job granted access to all idle job reservations.
INTERACTIVE	The job needs to interactive input from the user to run.
FSVIOLATION	The job was started with a fairshare violation.
GLOBALQUEUE	The job is directly submitted without doing any authentication.
NORESOURCES	The job is a system job that does not need any resources.
NORMSTART	The job will not query a resource manager to run.
CLUSTERLOCKED	The job is locked into the current cluster and cannot be migrated elsewhere. This is for grid mode.
FRAGMENT	The job can be run across multiple nodes in individual chunks.
SYSTEMJOB	The job is a system job which simply runs on the same node that Moab is running on. This is usually used for running scripts and other executables in workflows.

Value	Description
ADMINSETIGNPOLICIES	The IGNPOLICIES flag was set by an administrator.
EXTENDSTARTWALLTIME	The job duration (walltime) was extended at job start.
SHAREDMEM	The job will share its memory across nodes.
BLOCKEDBYGRES	The job's generic resource requirement caused the job to start later.
GRESONLY	The job is requesting only generic resources, no compute resources.
TEMPLATESAPPLIED	The job has had all applicable templates applied to it.
META	META job, just a container around resources.
WIDERSVSEARCHALGO	This job prefers the wide search algorithm.
VMTRACKING	The job is a VMTracking job for an externally-created VM (via job template).
DESTROYTEMPLATESUBMITTED	A destroy job has already been created from the template for this job.
PROCSPECIFIED	The job requested processors on the command line.
CANCELONFIRSTFAILURE	Cancel job array on first array job failure.
CANCELONFIRSTSUCCESS	Cancel job array on first array job success.
CANCELONANYFAILURE	Cancel job array on any array job failure.
CANCELONANYSUCCESS	Cancel job array on any array job success.
CANCELONEXITCODE	Cancel job array on a specific exit code.
NOVMMIGRATE	Do not migrate the virtual machine that this job sets up.
PURGEONSUCCESSONLY	Only purge the job if it completed successfully

JobHoldReason

Value	Description
Admin	
NoResources	
SystemLimitsExceeded	
BankFailure	
CannotDebitAccount	
InvalidAccount	
RMFailure	
RMReject	
PolicyViolation	
CredAccess	
CredHold	
PreReq	
Data	
Security	
MissingDependency	

JobHoldType

Value	Description
User	

Value	Description
System	
Batch	
Defer	
All	

DomainProxy

A reference to an object contained within an object. For example, a Virtual Machine object contains a reference to the Node on which it is running. That reference is represented by this class.

Field Name	Type	POST	Description
name	**String**	Yes	The name of the object.

Message

Field Name	Type	POST	Description
count	**Integer**	No	The number of times this message has occurred.
createdDate	**Date**	No	The date this message was created.
expireDate	**Date**	No	The date this message expires.
message	**String**	No	The message itself.

JobHostListMode

Value	Description
superset	
subset	

Value	Description
exactset	

JobPriority

Field Name	Type	POST	Description
run	**Long**	No	
start	**Long**	No	
system	**Long**	No	
user	**Long**	Yes	The user-requested priority for the job. By default, the range is between -1024 and 0. To enable priority range from -1024 to +1023, set `ENABLEPOSUSERPRIORITY` in the `moab.cfg` file.

JobQueueStatus

Value	Description
active	
blocked	
completed	
eligible	

JobRejectPolicy

Value	Description
CANCEL	
HOLD	

Value	Description
IGNORE	
MAIL	
RETRY	

JobRequirement

Field Name	Type	POST	Description
architecture	**String**	Yes	The architecture required by the job.
attributes	**Map<String, JobRequirementAttribute>**	Yes	Required node attributes with version number support.
features	**Set<String>**	No	The list of node features the job is scheduled against.
featuresExcluded	**Set<String>**	Yes	Excluded node features. That is, do not select nodes with these features. (See also: featuresExcludedMode.)
featuresExcludedMode	**JobRequirementFeaturesMode**	Yes	Indicates whether excluded features should be AND'ed or OR'd. The default is AND. Only relevant if featuresExcluded is provided. (See also: featuresExcluded.)
featuresRequested	**Set<String>**	Yes	Requested node features. (See also: featuresRequestedMode.)
featuresRequestedMode	**JobRequirementFeaturesMode**	Yes	Indicates whether requested features should be AND'ed or OR'd. The default is AND. Only relevant if featuresRequested is provided. (See also: featuresRequested.)
image	**String**	Yes	The image required by the job.

Field Name	Type	POST	Description
index	**Integer**	No	The index of the requirement, starting with 0.
metrics	**Map<String, Double>**	No	Generic metrics associated with the job as reported by the resource manager.
nodeAccessPolicy	[NodeAccessPolicy]()	Yes	Specifies how node resources should be accessed. Note: If the job requirements array has more than one element that contains nodeAccessPolicy, only the first occurrence will be used.
nodeAllocationPolicy	[NodeAllocationPolicy]()	Yes	Specifies how node resources should be selected and allocated to the job. Note: If the job requirements array has more than one element that contains nodeAllocationPolicy, only the first occurrence will be used.
nodeCount	**Integer**	Yes	The number of nodes required by the job.
nodeSet	**String**	Yes	The requested node set of the job. This must follow the format `SETSELECTION:SETTYPE[:SETLIST]` • SETSELECTION - `ANYOF`, `ONEOF`, or `FIRSTOF` • SETTYPE - `FEATURE` or `VARATTR` • SETLIST - For `FEATURE`, a comma-separated list of features. For `VARATTR`, a key=value pair. Examples: • `ONEOF:FEATURE:fastos,hiprio,bigmem` • `FIRSTOF:VARATTR:datacenter=Provo:datacenter=SaltLake`
nodes	[Set<AllocatedNode>]()	No	Nodes that have been allocated to meet this requirement.

Field Name	Type	POST	Description
reservation	DomainProxy	No	The allocated reservation (assigned after the job has a reservation).
resourcesPerTask	Map<String, JobResource>	Yes	Contains requirements for disk, memory, processors, swap, and generic resources. For disk, memory, and swap, the unit is MB. For each resource, the "dedicated" field can be set during POST.
taskCount	**Integer**	Yes	The number of tasks (processors) required by this job.
tasksPerNode	**Integer**	Yes	The number of tasks to map to each node.

JobRequirementAttribute

Field Name	Type	POST	Description
comparator	**String**	Yes	The comparison operator. Valid values: • >= - Greater than or equal to • > - Greater than • <= - Less than • < - Less than • %= - Equals • %! - Not equals • Null - Defaults to %= • = - (Deprecated) Equivalent to %=

Field Name	Type	POST	Description
displayValue	**String**	Yes	The display value for the required attribute.
restriction	**JobRequirementAttributeRestriction**	Yes	The restriction of this attribute. May be null, but defaults to JobRequirementAttributeRestriction.must.
value	**String**	Yes	The value of the required attribute. During POST, if value is missing, blank, or null, do not provide a comparator.

JobRequirementAttributeRestriction

Represents a restriction for a job requirement attribute.

Value	Description
must	

JobRequirementFeaturesMode

Value	Description
OR	
AND	

NodeAccessPolicy

This enumeration describes how node resources will be shared by various tasks.

Value	Description
NONE	
SHARED	Tasks from any combination of jobs may utilize available resources.

Value	Description
SHAREDONLY	Only jobs requesting shared node access may utilize available resources.
SINGLEJOB	Tasks from a single job may utilize available resourceses.
SINGLETASK	A single task from a single job may run on the node.
SINGLEUSER	Tasks from any jobs owned by the same user may utilize available resources.
UNIQUEUSER	Any number of tasks from a single job may allocate resources from a node but only if the user has no other jobs running on that node.
SINGLEGROUP	Any number of tasks from the same group may utilize node.
SINGLEACCOUNT	Any number of tasks from the same account may utilize node.

NodeAllocationPolicy

Node Allocation enumeration.

Value	Description
FIRSTSET	
MINGLOBAL	
MINLOCAL	
PLUGIN	
NONE	No node allocation policy is specified. Moab defaults to MINRESOURCE when this is the case.
FIRSTAVAILABLE	Simple first come, first served algorithm where nodes are allocated in the order they are presented by the resource manager. This is a very simple, very fast algorithm.

Value	Description
LASTAVAILABLE	This algorithm selects resources so as to minimize the amount of time after the job and before the trailing reservation. This algorithm is a best fit in time algorithm which minimizes the impact of reservation based node-time fragmentation. It is useful in systems where a large number of reservations (job, standing, or administrative) are in place.
MINRESOURCE	This algorithm prioritizes nodes according to the configured resources on each node. Those nodes with the fewest configured resources which still meet the job's resource constraints are selected.
CPULOAD	Nodes are selected which have the maximum amount of available, unused cpu power, i.e. [# of CPU's] - [CPU load]. Good algorithm for timesharing node systems. This algorithm is only applied to jobs starting immediately. For the purpose of future reservations, the **MINRESOURCE** algorithm is used.
LOCAL	This will call the locally created **contrib** node allocation algorithm.
CONTIGUOUS	This algorithm will allocate nodes in contiguous (linear) blocks as required by the Compaq RMS system.
MAXBALANCE	This algorithm will attempt to allocate the most 'balanced' set of nodes possible to a job. In most cases, but not all, the metric for balance of the nodes is node speed. Thus, if possible, nodes with identical speeds will be allocated to the job. If identical speed nodes cannot be found, the algorithm will allocate the set of nodes with the minimum node speed 'span' or range.
PRIORITY	This algorithm allows a site to specify the priority of various static and dynamic aspects of compute nodes and allocate them with preference for higher priority nodes. It is highly flexible allowing node attribute and usage information to be combined with reservation affinity.
FASTEST	This algorithm will select nodes in 'fastest node first' order. Nodes will be selected by node speed if specified. If node speed is not specified, nodes will be selected by processor speed. If neither is specified, nodes will be selected in a random order.
PROCESSORLOAD	Alias for CPULOAD.

Value	Description
NODESPEED	Alias for FASTEST.
INREPORTEDORDER	Alias for FIRSTAVAILABLE.
INREVERSEREPORTEDORDER	Alias for LASTAVAILABLE.
CUSTOMPRIORITY	Alias for PRIORITY.
PROCESSORSPEEDBALANCE	Alias for MAXBALANCE.
MINIMUMCONFIGUREDRESOURCES	Alias for MINRESOURCE.

AllocatedNode

Field Name	Type	POST	Description
name	**String**	No	
taskCount	**Integer**	No	

JobResource

Represents counts of dedicated and utilized resources.

Field Name	Type	POST	Description
dedicated	**Integer**	No	The amount of this resource that has been allocated for running workload.
utilized	**Integer**	No	The amount of this resource that is currently reported as utilized by resource managers.

JobResourceFailPolicyType

Value	Description
CANCEL	
FAIL	
HOLD	
IGNORE	
NOTIFY	
REQUEUE	

ResourceManager

Field Name	Type	POST	Description
isDestination	Boolean	No	
isSource	Boolean	No	
jobName	String	No	
name	String	No	

JobStateInformation

Field Name	Type	POST	Description
state	JobState	No	
stateExpected	JobState	No	
stateLastUpdatedDate	Date	No	
subState	JobSubState	No	

JobState

Value	Description
Idle	
Starting	
Running	
Removed	
Completed	
Hold	
Deferred	
Vacated	
NotQueued	
Unknown	
Staging	
Suspended	
Blocked	

JobSubState

Value	Description
Epilogue	
Migrated	
Preempted	

Value	Description
Prologue	

JobSystemJobType

Value	Description
generic	
osprovision	
osprovision2	
poweroff	
poweron	
reset	
storage	
vmmap	
vmmigrate	
vmtracking	

JobActionType

Value	Description
DESTROY	
MIGRATE	
MODIFY	

VMUsagePolicy

This enumeration describes the virtual machine requirements of a job

Value	Description
REQUIREPM	Requires a physical machine.
PREFPM	Prefers a physical machine.
CREATEVM	Creates a virtual machine.
CREATEPERSISTENTVM	Creates a virtual machine that doesn't go away after the job is done.
REQUIREVM	Requires a virtual machine.
PREFVM	Prefers a virtual machine.

Related topics

- **Job arrays** on page 1523

Fields: Jobs

See the associated **Jobs** on page 1525 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`jobs`	**Permissions** on page 1571
Hooks filename	`jobs.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	Yes	**Distinct** on page 1504

API version 3

Job

This class represents a job in the Moab Workload Manager. A job is a request for compute resources (CPUs, memory, storage) with which the requester can do work for a given amount of time. In an HPC environment, this might be a batch script to perform a Monte Carlo simulation. In a cloud environment, this would be a virtual machine and its associated storage. Moab will evaluate the request and assign the requested resources to the requester based on policies, current demand, and other factors in the data center. A job will also usually have some process that Moab starts automatically at the assigned start time. In an HPC environment, this can be starting a batch script on the assigned nodes. In a cloud environment, this can be starting provisioning processes to create the virtual machine and storage and install software on it.

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique identifier of this job. Note: this field is not user-assigned and is generated by the database.
arrayIndex	**Long**	No	No	If this job is a sub-job of a JobArray, this field contains the index of this job in the array. For example, if this job is `Moab.1[2]`, the array index would be `2`.
arrayMasterName	**String**	No	No	If this job is a sub-job of a JobArray, this field contains the name of the job array master. For example, if this job is `Moab.1[2]`, the array master name would be `Moab.1`.
attributes	**Set<String>**	Yes	No	The list of generic attributes associated with this job.
blocks	**Set<JobBlock>**	No	No	Reasons the job is blocked from running.
bypassCount	**Integer**	No	No	The number of times the job has been backfilled.

Field Name	Type	POST	PUT	Description
cancelCount	**Integer**	No	No	The number of times a job has received a cancel request.
commandFile	**String**	Yes	No	The name of the job script file (absolute path). If commandFile is set and commandScript is not set, then MWS must have read access to the file. If commandFile and commandScript are both set, then MWS does not read the contents of the file, but it does provide the name of the file to Moab.
commandLineArguments	**String**	Yes	No	The command line arguments passed to the job script specified by commandFile or commandScript.
commandScript	**String**	Yes	No	The contents of the job script. This field must be Base64-encoded.
completionCode	**Integer**	No	No	The exit code from this job.
cpuTime	**Long**	No	No	CPU usage time in seconds as reported by the resource manager.
credentials	**JobCredentials**	Yes	Yes	The credentials (user and group, for example) associated with this job.
customName	**String**	Yes	Yes	The user-specified name of this job.
dates	**JobDates**	Yes	Yes	Various dates associated with this job.

Field Name	Type	POST	PUT	Description
deferCount	**Integer**	No	No	The number of times a job has been deferred.
dependencies	**Set<JobDependency>**	Yes	No	Dependencies that must be fulfilled before the job can start.
description	**String**	No	No	The description of the job. Can be set only in a job template.
duration	**Long**	Yes	Yes	The length of time in seconds requested for the job. Note that it is possible to set duration to "INFINITY" if the AllowInfiniteJobs flag is set on the scheduler in the moab.cfg.
durationActive	**Long**	No	No	The length of time in seconds the job has been active or running.
durationQueued	**Long**	No	No	The length of time in seconds the job has been eligible to run in the queue.
durationRemaining	**Long**	No	No	An estimate of the time remaining, in seconds, before the job will complete.
durationSuspended	**Long**	No	No	The length of time in seconds the job has been suspended.
emailNotifyAddresses	**Set<String>**	Yes	No	The list of addresses to whom email is sent by the execution server.
emailNotifyTypes	**Set<JobEmailNotifyType>**	Yes	No	The list of email notify types attached to the job.

Field Name	Type	POST	PUT	Description
environmentRequested	**Boolean**	Yes	No	Setting this field to true tells the Moab Workload Manager to set various variables, if populated, in the job's environment.
environmentVariables	**Map<String, Map>**	Yes	No	The list of environment variables for this job.
epilogScript	**String**	Yes	No	The path to the TORQUE epilog script.
flags	**Set<JobFlag>**	Yes	Yes	The flags that are set on this job.
holdDate	**Date**	No	No	The date the most recent hold was placed on the job.
holdReason	**JobHoldReason**	No	No	The reason the job is on hold.
holds	**Set<JobHoldType>**	Yes	Yes	The holds that are set on the job. The "User" hold type is valid during POST.
initialWorkingDirectory	**String**	Yes	No	The path to the directory in which the job will be started.
isActive	**Boolean**	No	No	True if the job is active, false if the job is complete.
jobGroup	**String**	Yes	No	The job group to which this job belongs (different from credentials.group).
masterNode	**DomainProxy**	No	No	The first node in the list of allocated nodes for this job. For TORQUE jobs, this represents the "mother superior."

Field Name	Type	POST	PUT	Description
memorySecondsDedicated	**Double**	No	No	The memory seconds dedicated to the job as reported by its resource manager. Not all resource managers provide this information.
memorySecondsUtilized	**Double**	No	No	The memory seconds utilized by the job as reported by its resource manager. Not all resource managers provide this information.
messages	**Set<Message>**	No	Yes	The list of messages associated with the job. The "message" field is valid during PUT.
migrateCount	**Integer**	No	No	The number of times the job has been migrated.
minimumPreemptTime	**Long**	No	No	The minimum length of time, in seconds, an active job must be running before it is eligible for preemption.
mwmName	**String**	No	No	The name of the Moab Workload Manager instance that owns this job.
name	**String**	No	No	The name of this job. This name is unique *per instance* of Moab Workload Manager (i.e. not globally).
nodesExcluded	**Set<DomainProxy>**	Yes	No	The list of nodes that should not be considered for this job.

Field Name	Type	POST	PUT	Description
nodesRequested	**Set<DomainProxy>**	Yes	No	The exact set, superset, or subset of nodes on which this job must run. (See also: nodesRequestedPolicy.)
nodesRequestedPolicy	**JobHostListMode**	Yes	No	Indicates an exact set, superset, or subset of nodes on which the job must run. Only relevant if nodesRequested is provided. (See also: nodesRequested.)
partitionAccessList	**Set<String>**	No	No	The list of partitions that this job can access.
partitionAccessListRequested	**Set<String>**	Yes	Yes	The list of partitions that this job has requested.
preemptCount	**Integer**	No	No	The number of times the job has been preempted.
priorities	**JobPriority**	Yes	Yes	The list of priorities for the job.
processorSecondsDedicated	**Double**	No	No	The processor seconds dedicated to the job as reported by its resource manager. Not all resource managers provide this information.
processorSecondsLimit	**Double**	No	No	The limit for processorSecondsUtilized.
processorSecondsUtilized	**Double**	No	No	The processor seconds utilized by the job as reported by its resource manager. Not all resource managers provide this information.

Field Name	Type	POST	PUT	Description
prologScript	**String**	Yes	No	The path to the TORQUE prolog script.
queueStatus	**JobQueueStatus**	No	No	The status of the job in its queue.
rejectPolicies	**Set<JobRejectPolicy>**	No	No	The list of policies enabled when a job is rejected.
requirements	**Set<JobRequirement>**	Yes	Yes	The list of items required for this job to run. Only JobRequirement.features is valid during PUT.
reservationRequested	**DomainProxy**	Yes	Yes	The reservation that the job requested.
resourceFailPolicy	**JobResourceFailPolicyType**	Yes	No	The policy that dictates what should happen to the job if it is running and at least one of the resources it is using fails.
resourceManagerExtension	**String**	Yes	No	If provided during POST, this string will be added to the resource manager extension section of the job submission. Example: "bandwidth=120;queuejob=false" Note that the delimiter between resourceManagerExtension elements is the semicolon.
resourceManagers	**Set<ResourceManager>**	No	No	The list of resource managers associated with this job.
rmStandardErrorFilePath	**String**	No	No	The path to the remote file containing the standard error of the job.

Field Name	Type	POST	PUT	Description
rmStandardOutputFilePath	**String**	No	No	The path to the remote file containing the standard output of the job.
standardErrorFilePath	**String**	Yes	No	The path to the file containing the standard error of the job.
standardOutputFilePath	**String**	Yes	No	The path to the file containing the standard output of the job.
startCount	**Integer**	No	No	The number of times the job has been started.
states	**JobStateInformation**	No	No	Information about the state of the job.
submitHost	**String**	Yes	No	The host from which the job was submitted.
systemJobAction	**String**	No	No	The action the system job will take.
systemJobType	**JobSystemJobType**	No	No	The type of system job. In the Moab Cloud Suite, this will usually be "vmtracking" or "generic."
targetedJobAction	**JobActionType**	No	No	The action that this job is performing on another job.
targetedJobName	**String**	No	No	The name of the job on which this job is performing the targetedJobAction.
templates	**Set<DomainProxy>**	Yes	No	The list of all job templates to be set on this job.

Field Name	Type	POST	PUT	Description
triggers	**Set<String>**	No	No	The list of triggers associated with this job.
variables	**Map<String, Map>**	Yes	Yes	The list of variables that this job owns or sets on completion.
virtualContainers	**Set<DomainProxy>**	Yes	No	When submitting this job, add it to the specified existing virtual container. Valid during POST, but only one virtual container can be specified.
virtualMachines	**Set<DomainProxy>**	No	No	The list of virtual machines that are allocated to this job.
vmUsagePolicy	**VMUsagePolicy**	Yes	No	The requested Virtual Machine Usage Policy for this job.

JobBlock

Field Name	Type	POST	PUT	Description
category	**JobBlockCategory**	No	No	
message	**String**	No	No	
type	**JobBlockType**	No	No	

JobBlockCategory

Value	Description
depend	
jobBlock	

Value	Description
migrate	

JobBlockType

Value	Description
ActivePolicy	
BadUser	
Dependency	
EState	
FairShare	
Hold	
IdlePolicy	
LocalPolicy	
NoClass	
NoData	
NoResource	
NoTime	
PartitionAccess	
Priority	
RMSubmissionFailure	

Value	Description
StartDate	
State	
SysLimits	

JobCredentials

Moab Workload Manager supports the concept of credentials, which provide a means of attributing policy and resource access to entities such as users and groups. These credentials allow specification of job ownership, tracking of resource usage, enforcement of policies, and many other features.

Field Name	Type	POST	PUT	Description
account	**String**	Yes	Yes	The account credential is also referred to as the project. This credential is generally associated with a group of users along the lines of a particular project for accounting and billing purposes.
group	**String**	Yes	No	The group credential represents an aggregation of users. User-to-group mappings are often specified by the operating system or resource manager and typically map to a user's UNIX group ID. However, user-to-group mappings may also be provided by a security and identity management service, or you can specify such directly within Moab.
jobClass	**String**	Yes	Yes	The concept of the class credential is derived from the resource manager class or queue object. Classes differ from other credentials in that they more directly impact job attributes. In standard HPC usage, a user submits a job to a class and this class imposes a number of factors on the job. The attributes of a class may be specified within the resource manager or directly within Moab.

Field Name	Type	POST	PUT	Description
qos	**String**	No	No	The quality of service assigned to this job. The concept of a quality of service (QoS) credential is unique to Moab and is not derived from any underlying concept or peer service. In most cases, the QoS credential is used to allow a site to set up a selection of service levels for end-users to choose from on a long-term or job-by-job basis. QoS's differ from other credentials in that they are centered around special access where this access may allow use of additional services, additional resources, or improved responsiveness. Unique to this credential, organizations may also choose to apply different charge rates to the varying levels of service available within each QoS. As QoS is an internal credential, all QoS configuration occurs within Moab.
qosRequested	**String**	Yes	Yes	The quality of service requested for this job.
user	**String**	Yes	No	The user credential is the fundamental credential within a workload manager; each job requires an association with exactly one user. In fact, the user credential is the only required credential in Moab; all others are optional. In most cases, the job's user credential is configured within or managed by the operating system itself, although Moab may be configured to obtain this information from an independent security and identity management service.

JobDates

Field Name	Type	POST	PUT	Description
completedDate	**Date**	No	No	
createdDate	**Date**	No	No	
deadlineDate	**Date**	Yes	No	The deadline for completion of the job.
dispatchedDate	**Date**	No	No	
earliestRequestedStartDate	**Date**	Yes	Yes	The job will start no sooner than this date.

Field Name	Type	POST	PUT	Description
earliestStartDate	**Date**	No	No	
eligibleDate	**Date**	No	No	
lastCanceledDate	**Date**	No	No	
lastChargedDate	**Date**	No	No	
lastPreemptedDate	**Date**	No	No	
lastUpdatedDate	**Date**	No	No	
startDate	**Date**	No	No	
submitDate	**Date**	No	No	
terminationDate	**Date**	No	No	

JobDependency

Field Name	Type	POST	PUT	Description
name	**String**	Yes	No	The name of the object on on which the job is dependent.
type	**JobDependencyType**	Yes	No	The type of job dependency. Only set is valid for POST.
value	**String**	No	No	

JobDependencyType

Represents the type of a job dependency. For now, only the "set" type is supported.

Value	Description
set	

JobEmailNotifyType

Value	Description
JobStart	
JobEnd	
JobFail	
All	

JobFlag

This enumeration specifies the flag types of a job.

Value	Description
NONE	
BACKFILL	The job is using backfill to run.
COALLOC	The job can use resources from multiple resource managers and partitions.
ADVRES	The job requires the use of a reservation.
NOQUEUE	The job will attempt to execute immediately or fail.
ARRAYJOB	The job is part of a job array.
ARRAYJOBPARLOCK	This array job will only run in one partition.
ARRAYJOBPARSPAN	This array job will span partitions (default).
ARRAYMASTER	This job is the master of a job array.
BESTEFFORT	The job will succeed if even partial resources are available.
RESTARTABLE	The job is restartable.

Value	Description
SUSPENDABLE	The job is suspendable.
HASPREEMPTED	This job preempted other jobs to start.
PREEMPTEE	The job is a preemptee and therefore can be preempted by other jobs.
PREEMPTOR	The job is a preemptor and therefore can preempt other jobs.
RSVMAP	The job is based on a reservation.
SPVIOLATION	The job was started with a soft policy violation.
IGNNODEPOLICIES	The job will ignore node policies.
IGNPOLICIES	The job will ignore idle, active, class, partition, and system policies.
IGNNODESTATE	The job will ignore node state in order to run.
IGNIDLEJOBRSV	The job can ignore idle job reservations. The job granted access to all idle job reservations.
INTERACTIVE	The job needs to interactive input from the user to run.
FSVIOLATION	The job was started with a fairshare violation.
GLOBALQUEUE	The job is directly submitted without doing any authentication.
NORESOURCES	The job is a system job that does not need any resources.
NORMSTART	The job will not query a resource manager to run.
CLUSTERLOCKED	The job is locked into the current cluster and cannot be migrated elsewhere. This is for grid mode.
FRAGMENT	The job can be run across multiple nodes in individual chunks.

Value	Description
SYSTEMJOB	The job is a system job which simply runs on the same node that Moab is running on. This is usually used for running scripts and other executables in workflows.
ADMINSETIGNPOLICIES	The IGNPOLICIES flag was set by an administrator.
EXTENDSTARTWALLTIME	The job duration (walltime) was extended at job start.
SHAREDMEM	The job will share its memory across nodes.
BLOCKEDBYGRES	The job's generic resource requirement caused the job to start later.
GRESONLY	The job is requesting only generic resources, no compute resources.
TEMPLATESAPPLIED	The job has had all applicable templates applied to it.
META	META job, just a container around resources.
WIDERSVSEARCHALGO	This job prefers the wide search algorithm.
VMTRACKING	The job is a VMTracking job for an externally-created VM (via job template).
DESTROYTEMPLATESUBMITTED	A destroy job has already been created from the template for this job.
PROCSPECIFIED	The job requested processors on the command line.
CANCELONFIRSTFAILURE	Cancel job array on first array job failure.
CANCELONFIRSTSUCCESS	Cancel job array on first array job success.
CANCELONANYFAILURE	Cancel job array on any array job failure.
CANCELONANYSUCCESS	Cancel job array on any array job success.
CANCELONEXITCODE	Cancel job array on a specific exit code.
NOVMMIGRATE	Do not migrate the virtual machine that this job sets up.

Value	Description
PURGEONSUCCESSONLY	Only purge the job if it completed successfully

JobHoldReason

Value	Description
Admin	
NoResources	
SystemLimitsExceeded	
BankFailure	
CannotDebitAccount	
InvalidAccount	
RMFailure	
RMReject	
PolicyViolation	
CredAccess	
CredHold	
PreReq	
Data	
Security	
MissingDependency	

JobHoldType

Value	Description
User	
System	
Batch	
Defer	
All	

DomainProxy

A reference to an object contained within an object. For example, a Virtual Machine object contains a reference to the Node on which it is running. That reference is represented by this class.

Field Name	Type	POST	PUT	Description
name	**String**	Yes	No	The name of the object.

Message

Field Name	Type	POST	PUT	Description
count	**Integer**	No	No	The number of times this message has occurred.
createdDate	**Date**	No	No	The date this message was created.
expireDate	**Date**	No	No	The date this message expires.
message	**String**	No	Yes	The message itself.

JobHostListMode

Value	Description
superset	

Value	Description
subset	
exactset	

JobPriority

Field Name	Type	POST	PUT	Description
run	**Long**	No	No	
start	**Long**	No	No	
system	**Long**	No	No	
user	**Long**	Yes	Yes	The user-requested priority for the job. By default, the range is between -1024 and 0. To enable priority range from -1024 to +1023, set `ENABLEPOSUSERPRIORITY` in the `moab.cfg` file.

JobQueueStatus

Value	Description
active	
blocked	
completed	
eligible	

JobRejectPolicy

Value	Description
CANCEL	

Value	Description
HOLD	
IGNORE	
MAIL	
RETRY	

JobRequirement

Field Name	Type	POST	PUT	Description
architecture	**String**	Yes	No	The architecture required by the job.
attributes	**Map<String, JobRequirementAttribute>**	Yes	No	Required node attributes with version number support.
features	**Set<String>**	No	Yes	The list of node features the job is scheduled against.
featuresExcluded	**Set<String>**	Yes	No	Excluded node features. That is, do not select nodes with these features. (See also: featuresExcludedMode.)
featuresExcludedMode	**JobRequirementFeaturesMode**	Yes	No	Indicates whether excluded features should be AND'ed or OR'd. The default is AND. Only relevant if featuresExcluded is provided. (See also: featuresExcluded.)
featuresRequested	**Set<String>**	Yes	No	Requested node features. (See also: featuresRequestedMode.)

Field Name	Type	POST	PUT	Description
featuresRequestedMode	**JobRequirementFeaturesMode**	Yes	No	Indicates whether requested features should be AND'ed or OR'd. The default is AND. Only relevant if featuresRequested is provided. (See also: featuresRequested.)
image	**String**	Yes	No	The image required by the job.
index	**Integer**	No	No	The index of the requirement, starting with 0.
metrics	**Map<String, Double>**	No	No	Generic metrics associated with the job as reported by the resource manager.
nodeAccessPolicy	**NodeAccessPolicy**	Yes	No	Specifies how node resources should be accessed. Note: If the job requirements array has more than one element that contains nodeAccessPolicy, only the first occurrence will be used.
nodeAllocationPolicy	**NodeAllocationPolicy**	Yes	No	Specifies how node resources should be selected and allocated to the job. Note: If the job requirements array has more than one element that contains nodeAllocationPolicy, only the first occurrence will be used.
nodeCount	**Integer**	Yes	No	The number of nodes required by the job.

Field Name	Type	POST	PUT	Description
nodeSet	**String**	Yes	No	The requested node set of the job. This must follow the format `SETSELECTION:SETTYPE [:SETLIST]` • SETSELECTION - `ANYOF`, `ONEOF`, or `FIRSTOF` • SETTYPE - `FEATURE` or `VARATTR` • SETLIST - For `FEATURE`, a comma-separated list of features. For `VARATTR`, a key=value pair. Examples: • `ONEOF:FEATURE:fastos,hiprio,bigmem` • `FIRSTOF:VARATTR:datacenter=Provo:datacenter=SaltLake`
nodes	**Set<AllocatedNode>**	No	No	Nodes that have been allocated to meet this requirement.
reservation	**DomainProxy**	No	No	The allocated reservation (assigned after the job has a reservation).
resourcesPerTask	**Map<String, JobResource>**	Yes	No	Contains requirements for disk, memory, processors, swap, and generic resources. For disk, memory, and swap, the unit is MB. For each resource, the "dedicated" field can be set during POST.
taskCount	**Integer**	Yes	No	The number of tasks (processors) required by this job.
tasksPerNode	**Integer**	Yes	No	The number of tasks to map to each node.

JobRequirementAttribute

Field Name	Type	POST	PUT	Description
comparator	**String**	Yes	No	The comparison operator. Valid values: • >= - Greater than or equal to • > - Greater than • <= - Less than • < - Less than • %= - Equals • %! - Not equals • Null - Defaults to %= • = - (Deprecated) Equivalent to %=
displayValue	**String**	Yes	No	The display value for the required attribute.
restriction	**JobRequirementAttributeRestriction**	Yes	No	The restriction of this attribute. May be null, but defaults to JobRequirementAttributeRestriction.must.
value	**String**	Yes	No	The value of the required attribute. During POST, if value is missing, blank, or null, do not provide a comparator.

JobRequirementAttributeRestriction

Represents a restriction for a job requirement attribute.

Value	Description
must	

JobRequirementFeaturesMode

Value	Description
OR	
AND	

NodeAccessPolicy

This enumeration describes how node resources will be shared by various tasks.

Value	Description
NONE	
SHARED	Tasks from any combination of jobs may utilize available resources.
SHAREDONLY	Only jobs requesting shared node access may utilize available resources.
SINGLEJOB	Tasks from a single job may utilize available resourceses.
SINGLETASK	A single task from a single job may run on the node.
SINGLEUSER	Tasks from any jobs owned by the same user may utilize available resources.
UNIQUEUSER	Any number of tasks from a single job may allocate resources from a node but only if the user has no other jobs running on that node.
SINGLEGROUP	Any number of tasks from the same group may utilize node.
SINGLEACCOUNT	Any number of tasks from the same account may utilize node.

NodeAllocationPolicy

Node Allocation enumeration.

Value	Description
FIRSTSET	
MINGLOBAL	
MINLOCAL	
PLUGIN	
NONE	No node allocation policy is specified. Moab defaults to MINRESOURCE when this is the case.
FIRSTAVAILABLE	Simple first come, first served algorithm where nodes are allocated in the order they are presented by the resource manager. This is a very simple, very fast algorithm.
LASTAVAILABLE	This algorithm selects resources so as to minimize the amount of time after the job and before the trailing reservation. This algorithm is a best fit in time algorithm which minimizes the impact of reservation based node-time fragmentation. It is useful in systems where a large number of reservations (job, standing, or administrative) are in place.
MINRESOURCE	This algorithm prioritizes nodes according to the configured resources on each node. Those nodes with the fewest configured resources which still meet the job's resource constraints are selected.
CPULOAD	Nodes are selected which have the maximum amount of available, unused cpu power, i.e. [# of CPU's] - [CPU load]. Good algorithm for timesharing node systems. This algorithm is only applied to jobs starting immediately. For the purpose of future reservations, the **MINRESOURCE** algorithm is used.
LOCAL	This will call the locally created **contrib** node allocation algorithm.
CONTIGUOUS	This algorithm will allocate nodes in contiguous (linear) blocks as required by the Compaq RMS system.

Value	Description
MAXBALANCE	This algorithm will attempt to allocate the most 'balanced' set of nodes possible to a job. In most cases, but not all, the metric for balance of the nodes is node speed. Thus, if possible, nodes with identical speeds will be allocated to the job. If identical speed nodes cannot be found, the algorithm will allocate the set of nodes with the minimum node speed 'span' or range.
PRIORITY	This algorithm allows a site to specify the priority of various static and dynamic aspects of compute nodes and allocate them with preference for higher priority nodes. It is highly flexible allowing node attribute and usage information to be combined with reservation affinity.
FASTEST	This algorithm will select nodes in 'fastest node first' order. Nodes will be selected by node speed if specified. If node speed is not specified, nodes will be selected by processor speed. If neither is specified, nodes will be selected in a random order.
PROCESSORLOAD	Alias for CPULOAD.
NODESPEED	Alias for FASTEST.
INREPORTEDORDER	Alias for FIRSTAVAILABLE.
INREVERSEREPORTEDORDER	Alias for LASTAVAILABLE.
CUSTOMPRIORITY	Alias for PRIORITY.
PROCESSORSPEEDBALANCE	Alias for MAXBALANCE.
MINIMUMCONFIGUREDRESOURCES	Alias for MINRESOURCE.

AllocatedNode

Field Name	Type	POST	PUT	Description
name	**String**	No	No	
taskCount	**Integer**	No	No	

JobResource

Represents counts of dedicated and utilized resources.

Field Name	Type	POST	PUT	Description
dedicated	**Integer**	No	No	The amount of this resource that has been allocated for running workload.
utilized	**Integer**	No	No	The amount of this resource that is currently reported as utilized by resource managers.

JobResourceFailPolicyType

Value	Description
CANCEL	
FAIL	
HOLD	
IGNORE	
NOTIFY	
REQUEUE	

ResourceManager

Field Name	Type	POST	PUT	Description
isDestination	**Boolean**	No	No	
isSource	**Boolean**	No	No	
jobName	**String**	No	No	
name	**String**	No	No	

JobStateInformation

Field Name	Type	POST	PUT	Description
state	JobState	No	No	
stateExpected	JobState	No	No	
stateLastUpdatedDate	**Date**	No	No	
subState	JobSubState	No	No	

JobState

Value	Description
Idle	
Starting	
Running	
Removed	
Completed	
Hold	
Deferred	
Vacated	
NotQueued	
Unknown	
Staging	
Suspended	

Value	Description
Blocked	

JobSubState

Value	Description
Epilogue	
Migrated	
Preempted	
Prologue	

JobSystemJobType

Value	Description
generic	
osprovision	
osprovision2	
poweroff	
poweron	
reset	
storage	
vmmap	
vmmigrate	
vmtracking	

JobActionType

Value	Description
DESTROY	
MIGRATE	
MODIFY	

VMUsagePolicy

This enumeration describes the virtual machine requirements of a job

Value	Description
REQUIREPM	Requires a physical machine.
PREFPM	Prefers a physical machine.
CREATEVM	Creates a virtual machine.
CREATEPERSISTENTVM	Creates a virtual machine that doesn't go away after the job is done.
REQUIREVM	Requires a virtual machine.
PREFVM	Prefers a virtual machine.

API version 2

Job

This class represents a job in the Moab Workload Manager. A job is a request for compute resources (CPUs, memory, storage) with which the requester can do work for a given amount of time. In an HPC environment, this might be a batch script to perform a Monte Carlo simulation. In a cloud environment, this would be a virtual machine and its associated storage. Moab will evaluate the request and assign the requested resources to the requester based on policies, current demand, and other factors in the data center. A job will also usually have some process that Moab starts automatically at the assigned start time. In an HPC environment, this can be starting a batch script on the assigned nodes. In a cloud environment, this can be starting provisioning processes to create the virtual machine and storage and install software on it.

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique identifier of this job. Note: this field is not user-assigned and is generated by the database.
arrayIndex	**Long**	No	No	If this job is a sub-job of a JobArray, this field contains the index of this job in the array. For example, if this job is `Moab.1[2]`, the array index would be `2`.
arrayMasterName	**String**	No	No	If this job is a sub-job of a JobArray, this field contains the name of the job array master. For example, if this job is `Moab.1[2]`, the array master name would be `Moab.1`.
attributes	**Set<String>**	Yes	No	The list of generic attributes associated with this job.
blocks	**Set<JobBlock>**	No	No	Reasons the job is blocked from running.
bypassCount	**Integer**	No	No	The number of times the job has been backfilled.

Field Name	Type	POST	PUT	Description
cancelCount	**Integer**	No	No	The number of times a job has received a cancel request.
commandFile	**String**	Yes	No	The name of the job script file (absolute path). If commandFile is set and commandScript is not set, then MWS must have read access to the file. If commandFile and commandScript are both set, then MWS does not read the contents of the file, but it does provide the name of the file to Moab.
commandLineArguments	**String**	Yes	No	The command line arguments passed to the job script specified by commandFile or commandScript.
commandScript	**String**	Yes	No	The contents of the job script. This field must be Base64-encoded.
completionCode	**Integer**	No	No	The exit code from this job.
cpuTime	**Long**	No	No	CPU usage time in seconds as reported by the resource manager.
credentials	**JobCredentials**	Yes	Yes	The credentials (user and group, for example) associated with this job.
customName	**String**	Yes	Yes	The user-specified name of this job.
dates	**JobDates**	Yes	Yes	Various dates associated with this job.

Field Name	Type	POST	PUT	Description
deferCount	**Integer**	No	No	The number of times a job has been deferred.
dependencies	**Set<JobDependency>**	Yes	No	Dependencies that must be fulfilled before the job can start.
description	**String**	No	No	The description of the job. Can be set only in a job template.
duration	**Long**	Yes	Yes	The length of time in seconds requested for the job. Note that it is possible to set duration to "INFINITY" if the AllowInfiniteJobs flag is set on the scheduler in the moab.cfg.
durationActive	**Long**	No	No	The length of time in seconds the job has been active or running.
durationQueued	**Long**	No	No	The length of time in seconds the job has been eligible to run in the queue.
durationRemaining	**Long**	No	No	An estimate of the time remaining, in seconds, before the job will complete.
durationSuspended	**Long**	No	No	The length of time in seconds the job has been suspended.
emailNotifyAddresses	**Set<String>**	Yes	No	The list of addresses to whom email is sent by the execution server.
emailNotifyTypes	**Set<JobEmailNotifyType>**	Yes	No	The list of email notify types attached to the job.

Field Name	Type	POST	PUT	Description
environmentRequested	**Boolean**	Yes	No	Setting this field to true tells the Moab Workload Manager to set various variables, if populated, in the job's environment.
environmentVariables	**Map<String, Map>**	Yes	No	The list of environment variables for this job.
epilogScript	**String**	Yes	No	The path to the TORQUE epilog script.
flags	**Set<JobFlag>**	Yes	Yes	The flags that are set on this job.
holdDate	**Date**	No	No	The date the most recent hold was placed on the job.
holdReason	**JobHoldReason**	No	No	The reason the job is on hold.
holds	**Set<JobHoldType>**	Yes	Yes	The holds that are set on the job. The "User" hold type is valid during POST.
initialWorkingDirectory	**String**	Yes	No	The path to the directory in which the job will be started.
isActive	**Boolean**	No	No	True if the job is active, false if the job is complete.
jobGroup	**String**	Yes	No	The job group to which this job belongs (different from credentials.group).
masterNode	**DomainProxy**	No	No	The first node in the list of allocated nodes for this job. For TORQUE jobs, this represents the "mother superior."

Field Name	Type	POST	PUT	Description
memorySecondsDedicated	**Double**	No	No	The memory seconds dedicated to the job as reported by its resource manager. Not all resource managers provide this information.
memorySecondsUtilized	**Double**	No	No	The memory seconds utilized by the job as reported by its resource manager. Not all resource managers provide this information.
messages	**Set<Message>**	No	Yes	The list of messages associated with the job. The "message" field is valid during PUT.
migrateCount	**Integer**	No	No	The number of times the job has been migrated.
minimumPreemptTime	**Long**	No	No	The minimum length of time, in seconds, an active job must be running before it is eligible for preemption.
mwmName	**String**	No	No	The name of the Moab Workload Manager instance that owns this job.
name	**String**	No	No	The name of this job. This name is unique *per instance* of Moab Workload Manager (i.e. not globally).
nodesExcluded	**Set<DomainProxy>**	Yes	No	The list of nodes that should not be considered for this job.

Field Name	Type	POST	PUT	Description
nodesRequested	**Set<DomainProxy>**	Yes	No	The exact set, superset, or subset of nodes on which this job must run. (See also: nodesRequestedPolicy.)
nodesRequestedPolicy	**JobHostListMode**	Yes	No	Indicates an exact set, superset, or subset of nodes on which the job must run. Only relevant if nodesRequested is provided. (See also: nodesRequested.)
partitionAccessList	**Set<String>**	No	No	The list of partitions that this job can access.
partitionAccessListRequested	**Set<String>**	Yes	Yes	The list of partitions that this job has requested.
preemptCount	**Integer**	No	No	The number of times the job has been preempted.
priorities	**JobPriority**	Yes	Yes	The list of priorities for the job.
processorSecondsDedicated	**Double**	No	No	The processor seconds dedicated to the job as reported by its resource manager. Not all resource managers provide this information.
processorSecondsLimit	**Double**	No	No	The limit for processorSecondsUtilized.
processorSecondsUtilized	**Double**	No	No	The processor seconds utilized by the job as reported by its resource manager. Not all resource managers provide this information.

Field Name	Type	POST	PUT	Description
prologScript	**String**	Yes	No	The path to the TORQUE prolog script.
queueStatus	**JobQueueStatus**	No	No	The status of the job in its queue.
rejectPolicies	**Set<JobRejectPolicy>**	No	No	The list of policies enabled when a job is rejected.
requirements	**Set<JobRequirement>**	Yes	Yes	The list of items required for this job to run. Only JobRequirement.features is valid during PUT.
reservationRequested	**DomainProxy**	Yes	Yes	The reservation that the job requested.
resourceFailPolicy	**JobResourceFailPolicyType**	Yes	No	The policy that dictates what should happen to the job if it is running and at least one of the resources it is using fails.
resourceManagerExtension	**String**	Yes	No	If provided during POST, this string will be added to the resource manager extension section of the job submission. Example: "bandwidth=120;queuejob=false" Note that the delimiter between resourceManagerExtension elements is the semicolon.
resourceManagers	**Set<ResourceManager>**	No	No	The list of resource managers associated with this job.
rmStandardErrorFilePath	**String**	No	No	The path to the remote file containing the standard error of the job.

Field Name	Type	POST	PUT	Description
rmStandardOutputFilePath	**String**	No	No	The path to the remote file containing the standard output of the job.
standardErrorFilePath	**String**	Yes	No	The path to the file containing the standard error of the job.
standardOutputFilePath	**String**	Yes	No	The path to the file containing the standard output of the job.
startCount	**Integer**	No	No	The number of times the job has been started.
states	**JobStateInformation**	No	No	Information about the state of the job.
submitHost	**String**	Yes	No	The host from which the job was submitted.
systemJobAction	**String**	No	No	The action the system job will take.
systemJobType	**JobSystemJobType**	No	No	The type of system job. In the Moab Cloud Suite, this will usually be "vmtracking" or "generic."
targetedJobAction	**JobActionType**	No	No	The action that this job is performing on another job.
targetedJobName	**String**	No	No	The name of the job on which this job is performing the targetedJobAction.
templates	**Set<DomainProxy>**	Yes	No	The list of all job templates to be set on this job.

Field Name	Type	POST	PUT	Description
triggers	**Set<String>**	No	No	The list of triggers associated with this job.
variables	**Map<String, Map>**	Yes	Yes	The list of variables that this job owns or sets on completion.
virtualContainers	**Set<DomainProxy>**	Yes	No	When submitting this job, add it to the specified existing virtual container. Valid during POST, but only one virtual container can be specified.
virtualMachines	**Set<DomainProxy>**	No	No	The list of virtual machines that are allocated to this job.
vmUsagePolicy	**VMUsagePolicy**	Yes	No	The requested Virtual Machine Usage Policy for this job.

JobBlock

Field Name	Type	POST	PUT	Description
category	**JobBlockCategory**	No	No	
message	**String**	No	No	
type	**JobBlockType**	No	No	

JobBlockCategory

Value	Description
depend	
jobBlock	

Value	Description
migrate	

JobBlockType

Value	Description
ActivePolicy	
BadUser	
Dependency	
EState	
FairShare	
Hold	
IdlePolicy	
LocalPolicy	
NoClass	
NoData	
NoResource	
NoTime	
PartitionAccess	
Priority	
RMSubmissionFailure	

Value	Description
StartDate	
State	
SysLimits	

JobCredentials

Moab Workload Manager supports the concept of credentials, which provide a means of attributing policy and resource access to entities such as users and groups. These credentials allow specification of job ownership, tracking of resource usage, enforcement of policies, and many other features.

Field Name	Type	POST	PUT	Description
account	**String**	Yes	Yes	The account credential is also referred to as the project. This credential is generally associated with a group of users along the lines of a particular project for accounting and billing purposes.
group	**String**	Yes	No	The group credential represents an aggregation of users. User-to-group mappings are often specified by the operating system or resource manager and typically map to a user's UNIX group ID. However, user-to-group mappings may also be provided by a security and identity management service, or you can specify such directly within Moab.
jobClass	**String**	Yes	Yes	The concept of the class credential is derived from the resource manager class or queue object. Classes differ from other credentials in that they more directly impact job attributes. In standard HPC usage, a user submits a job to a class and this class imposes a number of factors on the job. The attributes of a class may be specified within the resource manager or directly within Moab.

Field Name	Type	POST	PUT	Description
qos	**String**	No	No	The quality of service assigned to this job. The concept of a quality of service (QoS) credential is unique to Moab and is not derived from any underlying concept or peer service. In most cases, the QoS credential is used to allow a site to set up a selection of service levels for end-users to choose from on a long-term or job-by-job basis. QoS's differ from other credentials in that they are centered around special access where this access may allow use of additional services, additional resources, or improved responsiveness. Unique to this credential, organizations may also choose to apply different charge rates to the varying levels of service available within each QoS. As QoS is an internal credential, all QoS configuration occurs within Moab.
qosRequested	**String**	Yes	Yes	The quality of service requested for this job.
user	**String**	Yes	No	The user credential is the fundamental credential within a workload manager; each job requires an association with exactly one user. In fact, the user credential is the only required credential in Moab; all others are optional. In most cases, the job's user credential is configured within or managed by the operating system itself, although Moab may be configured to obtain this information from an independent security and identity management service.

JobDates

Field Name	Type	POST	PUT	Description
completedDate	**Date**	No	No	
createdDate	**Date**	No	No	
deadlineDate	**Date**	Yes	No	The deadline for completion of the job.
dispatchedDate	**Date**	No	No	
earliestRequestedStartDate	**Date**	Yes	Yes	The job will start no sooner than this date.

Field Name	Type	POST	PUT	Description
earliestStartDate	**Date**	No	No	
eligibleDate	**Date**	No	No	
lastCanceledDate	**Date**	No	No	
lastChargedDate	**Date**	No	No	
lastPreemptedDate	**Date**	No	No	
lastUpdatedDate	**Date**	No	No	
startDate	**Date**	No	No	
submitDate	**Date**	No	No	
terminationDate	**Date**	No	No	

JobDependency

Field Name	Type	POST	PUT	Description
name	**String**	Yes	No	The name of the object on on which the job is dependent.
type	**JobDependencyType**	Yes	No	The type of job dependency. Only set is valid for POST.
value	**String**	No	No	

JobDependencyType

Represents the type of a job dependency. For now, only the "set" type is supported.

Value	Description
set	

JobEmailNotifyType

Value	Description
JobStart	
JobEnd	
JobFail	
All	

JobFlag

This enumeration specifies the flag types of a job.

Value	Description
NONE	
BACKFILL	The job is using backfill to run.
COALLOC	The job can use resources from multiple resource managers and partitions.
ADVRES	The job requires the use of a reservation.
NOQUEUE	The job will attempt to execute immediately or fail.
ARRAYJOB	The job is part of a job array.
ARRAYJOBPARLOCK	This array job will only run in one partition.
ARRAYJOBPARSPAN	This array job will span partitions (default).
ARRAYMASTER	This job is the master of a job array.
BESTEFFORT	The job will succeed if even partial resources are available.
RESTARTABLE	The job is restartable.

Value	Description
SUSPENDABLE	The job is suspendable.
HASPREEMPTED	This job preempted other jobs to start.
PREEMPTEE	The job is a preemptee and therefore can be preempted by other jobs.
PREEMPTOR	The job is a preemptor and therefore can preempt other jobs.
RSVMAP	The job is based on a reservation.
SPVIOLATION	The job was started with a soft policy violation.
IGNNODEPOLICIES	The job will ignore node policies.
IGNPOLICIES	The job will ignore idle, active, class, partition, and system policies.
IGNNODESTATE	The job will ignore node state in order to run.
IGNIDLEJOBRSV	The job can ignore idle job reservations. The job granted access to all idle job reservations.
INTERACTIVE	The job needs to interactive input from the user to run.
FSVIOLATION	The job was started with a fairshare violation.
GLOBALQUEUE	The job is directly submitted without doing any authentication.
NORESOURCES	The job is a system job that does not need any resources.
NORMSTART	The job will not query a resource manager to run.
CLUSTERLOCKED	The job is locked into the current cluster and cannot be migrated elsewhere. This is for grid mode.
FRAGMENT	The job can be run across multiple nodes in individual chunks.

Value	Description
SYSTEMJOB	The job is a system job which simply runs on the same node that Moab is running on. This is usually used for running scripts and other executables in workflows.
ADMINSETIGNPOLICIES	The IGNPOLICIES flag was set by an administrator.
EXTENDSTARTWALLTIME	The job duration (walltime) was extended at job start.
SHAREDMEM	The job will share its memory across nodes.
BLOCKEDBYGRES	The job's generic resource requirement caused the job to start later.
GRESONLY	The job is requesting only generic resources, no compute resources.
TEMPLATESAPPLIED	The job has had all applicable templates applied to it.
META	META job, just a container around resources.
WIDERSVSEARCHALGO	This job prefers the wide search algorithm.
VMTRACKING	The job is a VMTracking job for an externally-created VM (via job template).
DESTROYTEMPLATESUBMITTED	A destroy job has already been created from the template for this job.
PROCSPECIFIED	The job requested processors on the command line.
CANCELONFIRSTFAILURE	Cancel job array on first array job failure.
CANCELONFIRSTSUCCESS	Cancel job array on first array job success.
CANCELONANYFAILURE	Cancel job array on any array job failure.
CANCELONANYSUCCESS	Cancel job array on any array job success.
CANCELONEXITCODE	Cancel job array on a specific exit code.
NOVMMIGRATE	Do not migrate the virtual machine that this job sets up.

Value	Description
PURGEONSUCCESSONLY	Only purge the job if it completed successfully

JobHoldReason

Value	Description
Admin	
NoResources	
SystemLimitsExceeded	
BankFailure	
CannotDebitAccount	
InvalidAccount	
RMFailure	
RMReject	
PolicyViolation	
CredAccess	
CredHold	
PreReq	
Data	
Security	
MissingDependency	

JobHoldType

Value	Description
User	
System	
Batch	
Defer	
All	

DomainProxy

A reference to an object contained within an object. For example, a Virtual Machine object contains a reference to the Node on which it is running. That reference is represented by this class.

Field Name	Type	POST	PUT	Description
name	**String**	Yes	No	The name of the object.

Message

Field Name	Type	POST	PUT	Description
count	**Integer**	No	No	The number of times this message has occurred.
createdDate	**Date**	No	No	The date this message was created.
expireDate	**Date**	No	No	The date this message expires.
message	**String**	No	Yes	The message itself.

JobHostListMode

Value	Description
superset	

Value	Description
subset	
exactset	

JobPriority

Field Name	Type	POST	PUT	Description
run	**Long**	No	No	
start	**Long**	No	No	
system	**Long**	No	No	
user	**Long**	Yes	Yes	The user-requested priority for the job. By default, the range is between -1024 and 0. To enable priority range from -1024 to +1023, set `ENABLEPOSUSERPRIORITY` in the `moab.cfg` file.

JobQueueStatus

Value	Description
active	
blocked	
completed	
eligible	

JobRejectPolicy

Value	Description
CANCEL	

Value	Description
HOLD	
IGNORE	
MAIL	
RETRY	

JobRequirement

Field Name	Type	POST	PUT	Description
architecture	String	Yes	No	The architecture required by the job.
attributes	Map<String, JobRequirementAttribute>	Yes	No	Required node attributes with version number support.
features	Set<String>	No	Yes	The list of node features the job is scheduled against.
featuresExcluded	Set<String>	Yes	No	Excluded node features. That is, do not select nodes with these features. (See also: featuresExcludedMode.)
featuresExcludedMode	JobRequirementFeaturesMode	Yes	No	Indicates whether excluded features should be AND'ed or OR'd. The default is AND. Only relevant if featuresExcluded is provided. (See also: featuresExcluded.)
featuresRequested	Set<String>	Yes	No	Requested node features. (See also: featuresRequestedMode.)

Field Name	Type	POST	PUT	Description
featuresRequestedMode	**JobRequirementFeaturesMode**	Yes	No	Indicates whether requested features should be AND'ed or OR'd. The default is AND. Only relevant if featuresRequested is provided. (See also: featuresRequested.)
image	**String**	Yes	No	The image required by the job.
index	**Integer**	No	No	The index of the requirement, starting with 0.
metrics	**Map<String, Double>**	No	No	Generic metrics associated with the job as reported by the resource manager.
nodeAccessPolicy	**NodeAccessPolicy**	Yes	No	Specifies how node resources should be accessed. Note: If the job requirements array has more than one element that contains nodeAccessPolicy, only the first occurrence will be used.
nodeAllocationPolicy	**NodeAllocationPolicy**	Yes	No	Specifies how node resources should be selected and allocated to the job. Note: If the job requirements array has more than one element that contains nodeAllocationPolicy, only the first occurrence will be used.
nodeCount	**Integer**	Yes	No	The number of nodes required by the job.

Field Name	Type	POST	PUT	Description
nodeSet	**String**	Yes	No	The requested node set of the job. This must follow the format `SETSELECTION:SETTYPE[:SETLIST]` • SETSELECTION - `ANYOF`, `ONEOF`, or `FIRSTOF` • SETTYPE - `FEATURE` or `VARATTR` • SETLIST - For `FEATURE`, a comma-separated list of features. For `VARATTR`, a key=value pair. Examples: • `ONEOF:FEATURE:fastos,hiprio,bigmem` • `FIRSTOF:VARATTR:datacenter=Provo:datacenter=SaltLake`
nodes	**Set<AllocatedNode>**	No	No	Nodes that have been allocated to meet this requirement.
reservation	**DomainProxy**	No	No	The allocated reservation (assigned after the job has a reservation).
resourcesPerTask	**Map<String, JobResource>**	Yes	No	Contains requirements for disk, memory, processors, swap, and generic resources. For disk, memory, and swap, the unit is MB. For each resource, the "dedicated" field can be set during POST.
taskCount	**Integer**	Yes	No	The number of tasks (processors) required by this job.
tasksPerNode	**Integer**	Yes	No	The number of tasks to map to each node.

JobRequirementAttribute

Field Name	Type	POST	PUT	Description
comparator	**String**	Yes	No	The comparison operator. Valid values: • >= - Greater than or equal to • > - Greater than • <= - Less than • < - Less than • %= - Equals • %! - Not equals • Null - Defaults to %= • = - (Deprecated) Equivalent to %=
displayValue	**String**	Yes	No	The display value for the required attribute.
restriction	**JobRequirementAttributeRestriction**	Yes	No	The restriction of this attribute. May be null, but defaults to JobRequirementAttributeRestriction.must.
value	**String**	Yes	No	The value of the required attribute. During POST, if value is missing, blank, or null, do not provide a comparator.

JobRequirementAttributeRestriction

Represents a restriction for a job requirement attribute.

Value	Description
must	

JobRequirementFeaturesMode

Value	Description
OR	
AND	

NodeAccessPolicy

This enumeration describes how node resources will be shared by various tasks.

Value	Description
NONE	
SHARED	Tasks from any combination of jobs may utilize available resources.
SHAREDONLY	Only jobs requesting shared node access may utilize available resources.
SINGLEJOB	Tasks from a single job may utilize available resourceses.
SINGLETASK	A single task from a single job may run on the node.
SINGLEUSER	Tasks from any jobs owned by the same user may utilize available resources.
UNIQUEUSER	Any number of tasks from a single job may allocate resources from a node but only if the user has no other jobs running on that node.
SINGLEGROUP	Any number of tasks from the same group may utilize node.
SINGLEACCOUNT	Any number of tasks from the same account may utilize node.

NodeAllocationPolicy

Node Allocation enumeration.

Value	Description
FIRSTSET	
MINGLOBAL	
MINLOCAL	
PLUGIN	
NONE	No node allocation policy is specified. Moab defaults to MINRESOURCE when this is the case.
FIRSTAVAILABLE	Simple first come, first served algorithm where nodes are allocated in the order they are presented by the resource manager. This is a very simple, very fast algorithm.
LASTAVAILABLE	This algorithm selects resources so as to minimize the amount of time after the job and before the trailing reservation. This algorithm is a best fit in time algorithm which minimizes the impact of reservation based node-time fragmentation. It is useful in systems where a large number of reservations (job, standing, or administrative) are in place.
MINRESOURCE	This algorithm prioritizes nodes according to the configured resources on each node. Those nodes with the fewest configured resources which still meet the job's resource constraints are selected.
CPULOAD	Nodes are selected which have the maximum amount of available, unused cpu power, i.e. [# of CPU's] - [CPU load]. Good algorithm for timesharing node systems. This algorithm is only applied to jobs starting immediately. For the purpose of future reservations, the **MINRESOURCE** algorithm is used.
LOCAL	This will call the locally created **contrib** node allocation algorithm.
CONTIGUOUS	This algorithm will allocate nodes in contiguous (linear) blocks as required by the Compaq RMS system.

Value	Description
MAXBALANCE	This algorithm will attempt to allocate the most 'balanced' set of nodes possible to a job. In most cases, but not all, the metric for balance of the nodes is node speed. Thus, if possible, nodes with identical speeds will be allocated to the job. If identical speed nodes cannot be found, the algorithm will allocate the set of nodes with the minimum node speed 'span' or range.
PRIORITY	This algorithm allows a site to specify the priority of various static and dynamic aspects of compute nodes and allocate them with preference for higher priority nodes. It is highly flexible allowing node attribute and usage information to be combined with reservation affinity.
FASTEST	This algorithm will select nodes in 'fastest node first' order. Nodes will be selected by node speed if specified. If node speed is not specified, nodes will be selected by processor speed. If neither is specified, nodes will be selected in a random order.
PROCESSORLOAD	Alias for CPULOAD.
NODESPEED	Alias for FASTEST.
INREPORTEDORDER	Alias for FIRSTAVAILABLE.
INREVERSEREPORTEDORDER	Alias for LASTAVAILABLE.
CUSTOMPRIORITY	Alias for PRIORITY.
PROCESSORSPEEDBALANCE	Alias for MAXBALANCE.
MINIMUMCONFIGUREDRESOURCES	Alias for MINRESOURCE.

AllocatedNode

Field Name	Type	POST	PUT	Description
name	**String**	No	No	
taskCount	**Integer**	No	No	

JobResource

Represents counts of dedicated and utilized resources.

Field Name	Type	POST	PUT	Description
dedicated	**Integer**	No	No	The amount of this resource that has been allocated for running workload.
utilized	**Integer**	No	No	The amount of this resource that is currently reported as utilized by resource managers.

JobResourceFailPolicyType

Value	Description
CANCEL	
FAIL	
HOLD	
IGNORE	
NOTIFY	
REQUEUE	

ResourceManager

Field Name	Type	POST	PUT	Description
isDestination	**Boolean**	No	No	
isSource	**Boolean**	No	No	
jobName	**String**	No	No	
name	**String**	No	No	

JobStateInformation

Field Name	Type	POST	PUT	Description
state	**JobState**	No	No	
stateExpected	**JobState**	No	No	
stateLastUpdatedDate	**Date**	No	No	
subState	**JobSubState**	No	No	

JobState

Value	Description
Idle	
Starting	
Running	
Removed	
Completed	
Hold	
Deferred	
Vacated	
NotQueued	
Unknown	
Staging	
Suspended	

Value	Description
Blocked	

JobSubState

Value	Description
Epilogue	
Migrated	
Preempted	
Prologue	

JobSystemJobType

Value	Description
generic	
osprovision	
osprovision2	
poweroff	
poweron	
reset	
storage	
vmmap	
vmmigrate	
vmtracking	

JobActionType

Value	Description
DESTROY	
MIGRATE	
MODIFY	

VMUsagePolicy

This enumeration describes the virtual machine requirements of a job

Value	Description
REQUIREPM	Requires a physical machine.
PREFPM	Prefers a physical machine.
CREATEVM	Creates a virtual machine.
CREATEPERSISTENTVM	Creates a virtual machine that doesn't go away after the job is done.
REQUIREVM	Requires a virtual machine.
PREFVM	Prefers a virtual machine.

Related topics

- **Jobs** on page 1525

Fields: Job Templates

See the associated **Job templates** on page 1547 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`job-templates`	**Permissions** on page 1571

Type	Value	Additional information
Hooks filename	`job-templates.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

JobTemplate

This class represents a job template in the Moab Workload Manager. Job templates are used for two primary purposes: (1) to provide a means of generically matching and categorizing jobs, and (2) to provide a means of setting arbitrary default or forced attributes for certain jobs.

Field Name	Type	Description
id	**String**	The unique identifier for this job template.
account	**String**	The account under which this job will run for billing purposes.
args	**String**	Command-line arguments that get passed to commandFile.
commandFile	**String**	The path to the file that is executed when the job runs. This is the script that will actually call all the work of the job. Can be null.
description	**String**	The description of the job.
durationRequested	**Long**	The amount of time (in seconds) requested for the job.
genericSystemJob	**Boolean**	True if this template will instantiate a generic system job.
inheritResources	**Boolean**	True if jobs instantiated from this template inherit resources.
jobDependencies	**Set<JobTemplateDependency>**	The list of dependencies for this job template.
jobFlags	**Set<JobFlag>**	Job flags for this template.
jobTemplateFlags	**Set<JobTemplateFlag>**	Job template flags for this template.
jobTemplateRequirements	**Set<JobTemplateRequirement>**	The requirements for this job template.

Field Name	Type	Description
priority	**Long**	Relative job priority.
qos	**String**	The Quality of Service for the job.
queue	**String**	The class or queue in which the job will run.
select	**Boolean**	True if job template can be directly requested by job at submission.
trigger	**Trigger**	The trigger that is typically assigned to generic system jobs.
vmUsagePolicy	**VMUsagePolicy**	The virtual machine usage policy.

JobTemplateDependency

Field Name	Type	Description
name	**String**	The name of the template on which this template depends.
type	**JobDependencyTypeVersion1**	The type of the dependency.

JobDependencyTypeVersion1

Value	Description
JOBSTART	Job may start at any time after specified jobs have started execution.
JOBSUCCESSFULCOMPLETE	Job may be start at any time after all specified jobs have successfully completed.
JOBFAILEDCOMPLETE	Job may start at any time after any specified jobs have completed unsuccessfully.

Value	Description
JOBCOMPLETE	Job may start at any time after all specified jobs have completed regardless of completion status.
BEFORE	Job may start at any time before specified jobs have started execution. NOTE: Only reported to Moab and then reported back. Moab currently cannot internally handle this type of dependency.
BEFOREANY	Job may start at any time before all specified jobs have completed regardless of completion status. NOTE: Only reported to Moab and then reported back. Moab currently cannot internally handle this type of dependency.
BEFOREOK	Job may start at any time before all specified jobs have successfully completed. NOTE: Only reported to Moab and then reported back. Moab currently cannot internally handle this type of dependency.
BEFORENOTOK	Job may start at any time before any specified jobs have completed unsuccessfully. NOTE: Only reported to Moab and then reported back. Moab currently cannot internally handle this type of dependency.
HIBERNATE	Job was set to Hibernate mode.
SYNCWITH	Job will wait until it can start simultaneously with a master job
SYNCCOUNT	This job will wait until it can start simultaneously with synccount jobs of type syncwith that have all specified this synccount job is their master job.
SET	Job will wait until a variable on a Moab object is set before starting.

JobFlag

This enumeration specifies the flag types of a job.

Value	Description
NONE	
BACKFILL	The job is using backfill to run.
COALLOC	The job can use resources from multiple resource managers and partitions.

Value	Description
ADVRES	The job requires the use of a reservation.
NOQUEUE	The job will attempt to execute immediately or fail.
ARRAYJOB	The job is part of a job array.
ARRAYJOBPARLOCK	This array job will only run in one partition.
ARRAYJOBPARSPAN	This array job will span partitions (default).
ARRAYMASTER	This job is the master of a job array.
BESTEFFORT	The job will succeed if even partial resources are available.
RESTARTABLE	The job is restartable.
SUSPENDABLE	The job is suspendable.
HASPREEMPTED	This job preempted other jobs to start.
PREEMPTEE	The job is a preemptee and therefore can be preempted by other jobs.
PREEMPTOR	The job is a preemptor and therefore can preempt other jobs.
RSVMAP	The job is based on a reservation.
SPVIOLATION	The job was started with a soft policy violation.
IGNNODEPOLICIES	The job will ignore node policies.
IGNPOLICIES	The job will ignore idle, active, class, partition, and system policies.
IGNNODESTATE	The job will ignore node state in order to run.
IGNIDLEJOBRSV	The job can ignore idle job reservations. The job granted access to all idle job reservations.

Value	Description
INTERACTIVE	The job needs to interactive input from the user to run.
FSVIOLATION	The job was started with a fairshare violation.
GLOBALQUEUE	The job is directly submitted without doing any authentication.
NORESOURCES	The job is a system job that does not need any resources.
NORMSTART	The job will not query a resource manager to run.
CLUSTERLOCKED	The job is locked into the current cluster and cannot be migrated elsewhere. This is for grid mode.
FRAGMENT	The job can be run across multiple nodes in individual chunks.
SYSTEMJOB	The job is a system job which simply runs on the same node that Moab is running on. This is usually used for running scripts and other executables in workflows.
ADMINSETIGNPOLICIES	The IGNPOLICIES flag was set by an administrator.
EXTENDSTARTWALLTIME	The job duration (walltime) was extended at job start.
SHAREDMEM	The job will share its memory across nodes.
BLOCKEDBYGRES	The job's generic resource requirement caused the job to start later.
GRESONLY	The job is requesting only generic resources, no compute resources.
TEMPLATESAPPLIED	The job has had all applicable templates applied to it.
META	META job, just a container around resources.
WIDERSVSEARCHALGO	This job prefers the wide search algorithm.
VMTRACKING	The job is a VMTracking job for an externally-created VM (via job template).

Value	Description
DESTROYTEMPLATESUBMITTED	A destroy job has already been created from the template for this job.
PROCSPECIFIED	The job requested processors on the command line.
CANCELONFIRSTFAILURE	Cancel job array on first array job failure.
CANCELONFIRSTSUCCESS	Cancel job array on first array job success.
CANCELONANYFAILURE	Cancel job array on any array job failure.
CANCELONANYSUCCESS	Cancel job array on any array job success.
CANCELONEXITCODE	Cancel job array on a specific exit code.
NOVMMIGRATE	Do not migrate the virtual machine that this job sets up.
PURGEONSUCCESSONLY	Only purge the job if it completed successfully

JobTemplateFlag

This enumeration specifies the flag types of a job template.

Value	Description
GLOBALRSVACCESS	
HIDDEN	
HWJOB	
PRIVATE	
SYNCJOBID	
TEMPLATEISDYNAMIC	True if the template is dynamic (not specified via moab.cfg).
SELECT	True if a job can select this template.

JobTemplateRequirement

Field Name	Type	Description
architecture	**String**	The architecture requirement.
diskRequirement	**Integer**	The amount of disk space required (in MB).
genericResources	**Map<String, Integer>**	Consumable generic attributes associated with individual nodes or the special pseudo-node global, which provides shared cluster (floating) consumable resources.
nodeAccessPolicy	**NodeAccessPolicy**	The node access policy. Specifies how node resources will be shared by a job.
operatingSystem	**String**	The operating system requirement.
requiredDiskPerTask	**Integer**	Disk space (in MB).
requiredFeatures	**Set<String>**	The features required by this template.
requiredMemoryPerTask	**Integer**	Memory (in MB).
requiredProcessorsPerTask	**Integer**	Number of processors.
requiredSwapPerTask	**Integer**	Swap space (in MB).
taskCount	**Integer**	The number of tasks required.

NodeAccessPolicy

This enumeration describes how node resources will be shared by various tasks.

Value	Description
NONE	
SHARED	Tasks from any combination of jobs may utilize available resources.
SHAREDONLY	Only jobs requesting shared node access may utilize available resources.

Value	Description
SINGLEJOB	Tasks from a single job may utilize available resourceses.
SINGLETASK	A single task from a single job may run on the node.
SINGLEUSER	Tasks from any jobs owned by the same user may utilize available resources.
UNIQUEUSER	Any number of tasks from a single job may allocate resources from a node but only if the user has no other jobs running on that node.
SINGLEGROUP	Any number of tasks from the same group may utilize node.
SINGLEACCOUNT	Any number of tasks from the same account may utilize node.

Trigger

Field Name	Type	Description
id	**String**	Trigger id - internal ID used by moab to track triggers
action	**String**	For exec atype triggers, signifies executable and arguments. For jobpreempt atype triggers, signifies PREEMPTPOLICY to apply to jobs that are running on allocated resources. For changeparam atype triggers, specifies the parameter to change and its new value (using the same syntax and behavior as the changeparam command).
actionType	**TriggerActionType**	
blockTime	**Date**	Time (in seconds) Moab will suspend normal operation to wait for trigger execution to finish. Use caution as Moab will completely stop normal operation until BlockTime expires.
description	**String**	
eventType	**TriggerEventType**	
expireTime	**Date**	Time at which trigger should be terminated if it has not already been activated.

Field Name	Type	Description
failOffset	**Date**	Specifies the time (in seconds) that the threshold condition must exist before the trigger fires.
flags	**Set<TriggerFlag>**	
interval	**Boolean**	When used in conjunction with MultiFire and RearmTime trigger will fire at regular intervals. Can be used with TriggerEventType.EPOCH to create a Standing Trigger. Defaults to false
maxRetry	**Integer**	Specifies the number of times Action will be attempted before the trigger is designated a failure.
multiFire	**Boolean**	Specifies whether this trigger can fire multiple times. Defaults to false.
name	**String**	Trigger name - can be auto assigned by moab or requested. Alphanumeric up to 16 characters in length
objectId	**String**	The ID of the object which this is attached to.
objectType	**String**	The type of object which this is attached to. Possible values: • vm - Virtual Machine
offset	**Date**	Relative time offset from event when trigger can fire.
period	**TriggerPeriod**	Can be used in conjunction with Offset to have a trigger fire at the beginning of the specified period. Can be used with EType epoch to create a standing trigger.
rearmTime	**Date**	Time between MultiFire triggers; rearm time is enforced from the trigger event time.
requires	**String**	Variables this trigger requires to be set or not set before it will fire. Preceding the string with an exclamation mark (!) indicates this variable must NOT be set. Used in conjunction with Sets to create trigger dependencies.

Field Name	Type	Description
sets	**String**	Variable values this trigger sets upon success or failure. Preceding the string with an exclamation mark (!) indicates this variable is set upon trigger failure. Preceding the string with a caret (^) indicates this variable is to be exported to the parent object when the current object is destroyed through a completion event. Used in conjunction with Requires to create trigger dependencies.
threshold	**String**	Reservation usage threshold - When reservation usage drops below Threshold, trigger will fire. Threshold usage support is only enabled for reservations and applies to percent processor utilization. gmetric thresholds are supported with job, node, credential, and reservation triggers. See Threshold Triggers in the Moab Workload Manager documentation for more information.
timeout	**Date**	Time allotted to this trigger before it is marked as unsuccessful and its process (if any) killed.
unsets	**String**	Variable this trigger destroys upon success or failure.

TriggerActionType

This enumeration specifies the action type of a trigger.

Value	Description
CANCEL	Only apply to reservation triggers
CHANGE_PARAM	
JOB_PREEMPT	This indicates that the trigger should preempt all jobs currently allocating resources assigned to the trigger's parent object. Only apply to reservation triggers.
MAIL	
THRESHOLD	
INTERNAL	

Value	Description
EXEC	

TriggerEventType

This enumeration specifies the event type of a trigger.

Value	Description
CANCEL	
CHECKPOINT	
CREATE	
END	
EPOCH	
FAIL	
HOLD	
MIGRATE	
MODIFY	
PREEMPT	
STANDING	
START	
THRESHOLD	

TriggerFlag

This enumeration specifies a flag belonging to a trigger.

Value	Description
ATTACH_ERROR	If the trigger outputs anything to stderr, Moab will attach this as a message to the trigger object.
CLEANUP	If the trigger is still running when the parent object completes or is canceled, the trigger will be killed.
CHECKPOINT	Moab should always checkpoint this trigger. See Checkpointing a Trigger in the Moab Workload Manager documentation for more information.
GLOBAL_VARS	The trigger will look in the name space of all nodes with the globalvars flag in addition to its own name space. A specific node to search can be specified using the following format: globalvars+node_id
INTERVAL	Trigger is periodic.
MULTIFIRE	Trigger can fire multiple times.
OBJECT_XML_ STDIN	Trigger passes its parent's object XML information into the trigger's stdin. This only works for exec triggers with reservation type parents.
USER	The trigger will execute under the user ID of the object's owner. If the parent object is sched, the user to run under may be explicitly specified using the format user+<username>, for example flags=user+john:
GLOBAL_ TRIGGER	The trigger will be (or was) inserted into the global trigger list.
ASYNCHRONOUS	An asynchronous trigger.
LEAVE_FILES	Do not remove stderr and stdout files.
PROBE	The trigger's stdout will be monitored.
PROBE_ALL	The trigger's stdout will be monitored.
GENERIC_ SYSTEM_JOB	The trigger belongs to a generic system job (for checkpointing).

Value	Description
REMOVE_STD_FILES	The trigger will delete stdout/stderr files after it has been reset.
RESET_ON_MODIFY	The trigger resets if the object it is attached to is modified, even if multifire is not set.
SOFT_KILL	By default, a `SIGKILL` (kill -9) signal is sent to the kill the script when a trigger times out. This flag will instead send a `SIGTERM` (kill -15) signal to kill the script. The `SIGTERM` signal will allow the script to trap the signal so that the script can clean up any residual information on the system (instead of just dying, as with the `SIGKILL` signal). NOTE: A timed-out trigger will only receive one kill signal. This means that if you specify this flag, a timed-out trigger will only receive the `SIGTERM` signal, and never the `SIGKILL` signal.

TriggerPeriod

This enumeration specifies the period of a trigger.

Value	Description
MINUTE	
HOUR	
DAY	
WEEK	
MONTH	

VMUsagePolicy

This enumeration describes the virtual machine requirements of a job

Value	Description
REQUIREPM	Requires a physical machine.
PREFPM	Prefers a physical machine.

Value	Description
CREATEVM	Creates a virtual machine.
CREATEPERSISTENTVM	Creates a virtual machine that doesn't go away after the job is done.
REQUIREVM	Requires a virtual machine.
PREFVM	Prefers a virtual machine.

API version 2

JobTemplate

This class represents a job template in the Moab Workload Manager. Job templates are used for two primary purposes: (1) to provide a means of generically matching and categorizing jobs, and (2) to provide a means of setting arbitrary default or forced attributes for certain jobs.

Field Name	Type	Description
id	**String**	The unique identifier for this job template.
account	**String**	The account under which this job will run for billing purposes.
args	**String**	Command-line arguments that get passed to commandFile.
commandFile	**String**	The path to the file that is executed when the job runs. This is the script that will actually call all the work of the job. Can be null.
description	**String**	The description of the job.
durationRequested	**Long**	The amount of time (in seconds) requested for the job.
genericSystemJob	**Boolean**	True if this template will instantiate a generic system job.
inheritResources	**Boolean**	True if jobs instantiated from this template inherit resources.
jobDependencies	**Set<JobTemplateDependency>**	The list of dependencies for this job template.
jobFlags	**Set<JobFlag>**	Job flags for this template.
jobTemplateFlags	**Set<JobTemplateFlag>**	Job template flags for this template.
jobTemplateRequirements	**Set<JobTemplateRequirement>**	The requirements for this job template.

Field Name	Type	Description
priority	**Long**	Relative job priority.
qos	**String**	The Quality of Service for the job.
queue	**String**	The class or queue in which the job will run.
select	**Boolean**	True if job template can be directly requested by job at submission.
trigger	**Trigger**	The trigger that is typically assigned to generic system jobs.
vmUsagePolicy	**VMUsagePolicy**	The virtual machine usage policy.

JobTemplateDependency

Field Name	Type	Description
name	**String**	The name of the template on which this template depends.
type	**JobDependencyTypeVersion1**	The type of the dependency.

JobDependencyTypeVersion1

Value	Description
JOBSTART	Job may start at any time after specified jobs have started execution.
JOBSUCCESSFULCOMPLETE	Job may be start at any time after all specified jobs have successfully completed.
JOBFAILEDCOMPLETE	Job may start at any time after any specified jobs have completed unsuccessfully.

Value	Description
JOBCOMPLETE	Job may start at any time after all specified jobs have completed regardless of completion status.
BEFORE	Job may start at any time before specified jobs have started execution. NOTE: Only reported to Moab and then reported back. Moab currently cannot internally handle this type of dependency.
BEFOREANY	Job may start at any time before all specified jobs have completed regardless of completion status. NOTE: Only reported to Moab and then reported back. Moab currently cannot internally handle this type of dependency.
BEFOREOK	Job may start at any time before all specified jobs have successfully completed. NOTE: Only reported to Moab and then reported back. Moab currently cannot internally handle this type of dependency.
BEFORENOTOK	Job may start at any time before any specified jobs have completed unsuccessfully. NOTE: Only reported to Moab and then reported back. Moab currently cannot internally handle this type of dependency.
HIBERNATE	Job was set to Hibernate mode.
SYNCWITH	Job will wait until it can start simultaneously with a master job
SYNCCOUNT	This job will wait until it can start simultaneously with synccount jobs of type syncwith that have all specified this synccount job is their master job.
SET	Job will wait until a variable on a Moab object is set before starting.

JobFlag

This enumeration specifies the flag types of a job.

Value	Description
NONE	
BACKFILL	The job is using backfill to run.
COALLOC	The job can use resources from multiple resource managers and partitions.

Value	Description
ADVRES	The job requires the use of a reservation.
NOQUEUE	The job will attempt to execute immediately or fail.
ARRAYJOB	The job is part of a job array.
ARRAYJOBPARLOCK	This array job will only run in one partition.
ARRAYJOBPARSPAN	This array job will span partitions (default).
ARRAYMASTER	This job is the master of a job array.
BESTEFFORT	The job will succeed if even partial resources are available.
RESTARTABLE	The job is restartable.
SUSPENDABLE	The job is suspendable.
HASPREEMPTED	This job preempted other jobs to start.
PREEMPTEE	The job is a preemptee and therefore can be preempted by other jobs.
PREEMPTOR	The job is a preemptor and therefore can preempt other jobs.
RSVMAP	The job is based on a reservation.
SPVIOLATION	The job was started with a soft policy violation.
IGNNODEPOLICIES	The job will ignore node policies.
IGNPOLICIES	The job will ignore idle, active, class, partition, and system policies.
IGNNODESTATE	The job will ignore node state in order to run.
IGNIDLEJOBRSV	The job can ignore idle job reservations. The job granted access to all idle job reservations.

Value	Description
INTERACTIVE	The job needs to interactive input from the user to run.
FSVIOLATION	The job was started with a fairshare violation.
GLOBALQUEUE	The job is directly submitted without doing any authentication.
NORESOURCES	The job is a system job that does not need any resources.
NORMSTART	The job will not query a resource manager to run.
CLUSTERLOCKED	The job is locked into the current cluster and cannot be migrated elsewhere. This is for grid mode.
FRAGMENT	The job can be run across multiple nodes in individual chunks.
SYSTEMJOB	The job is a system job which simply runs on the same node that Moab is running on. This is usually used for running scripts and other executables in workflows.
ADMINSETIGNPOLICIES	The IGNPOLICIES flag was set by an administrator.
EXTENDSTARTWALLTIME	The job duration (walltime) was extended at job start.
SHAREDMEM	The job will share its memory across nodes.
BLOCKEDBYGRES	The job's generic resource requirement caused the job to start later.
GRESONLY	The job is requesting only generic resources, no compute resources.
TEMPLATESAPPLIED	The job has had all applicable templates applied to it.
META	META job, just a container around resources.
WIDERSVSEARCHALGO	This job prefers the wide search algorithm.
VMTRACKING	The job is a VMTracking job for an externally-created VM (via job template).

Value	Description
DESTROYTEMPLATESUBMITTED	A destroy job has already been created from the template for this job.
PROCSPECIFIED	The job requested processors on the command line.
CANCELONFIRSTFAILURE	Cancel job array on first array job failure.
CANCELONFIRSTSUCCESS	Cancel job array on first array job success.
CANCELONANYFAILURE	Cancel job array on any array job failure.
CANCELONANYSUCCESS	Cancel job array on any array job success.
CANCELONEXITCODE	Cancel job array on a specific exit code.
NOVMMIGRATE	Do not migrate the virtual machine that this job sets up.
PURGEONSUCCESSONLY	Only purge the job if it completed successfully

JobTemplateFlag

This enumeration specifies the flag types of a job template.

Value	Description
GLOBALRSVACCESS	
HIDDEN	
HWJOB	
PRIVATE	
SYNCJOBID	
TEMPLATEISDYNAMIC	True if the template is dynamic (not specified via moab.cfg).
SELECT	True if a job can select this template.

JobTemplateRequirement

Field Name	Type	Description
architecture	**String**	The architecture requirement.
diskRequirement	**Integer**	The amount of disk space required (in MB).
genericResources	**Map<String, Integer>**	Consumable generic attributes associated with individual nodes or the special pseudo-node global, which provides shared cluster (floating) consumable resources.
nodeAccessPolicy	**NodeAccessPolicy**	The node access policy. Specifies how node resources will be shared by a job.
operatingSystem	**String**	The operating system requirement.
requiredDiskPerTask	**Integer**	Disk space (in MB).
requiredFeatures	**Set<String>**	The features required by this template.
requiredMemoryPerTask	**Integer**	Memory (in MB).
requiredProcessorsPerTask	**Integer**	Number of processors.
requiredSwapPerTask	**Integer**	Swap space (in MB).
taskCount	**Integer**	The number of tasks required.

NodeAccessPolicy

This enumeration describes how node resources will be shared by various tasks.

Value	Description
NONE	
SHARED	Tasks from any combination of jobs may utilize available resources.
SHAREDONLY	Only jobs requesting shared node access may utilize available resources.

Value	Description
SINGLEJOB	Tasks from a single job may utilize available resourceses.
SINGLETASK	A single task from a single job may run on the node.
SINGLEUSER	Tasks from any jobs owned by the same user may utilize available resources.
UNIQUEUSER	Any number of tasks from a single job may allocate resources from a node but only if the user has no other jobs running on that node.
SINGLEGROUP	Any number of tasks from the same group may utilize node.
SINGLEACCOUNT	Any number of tasks from the same account may utilize node.

Trigger

Field Name	Type	Description
id	**String**	Trigger id - internal ID used by moab to track triggers
action	**String**	For exec atype triggers, signifies executable and arguments. For jobpreempt atype triggers, signifies PREEMPTPOLICY to apply to jobs that are running on allocated resources. For changeparam atype triggers, specifies the parameter to change and its new value (using the same syntax and behavior as the changeparam command).
actionType	**TriggerActionType**	
blockTime	**Date**	Time (in seconds) Moab will suspend normal operation to wait for trigger execution to finish. Use caution as Moab will completely stop normal operation until BlockTime expires.
description	**String**	
eventType	**TriggerEventType**	
expireTime	**Date**	Time at which trigger should be terminated if it has not already been activated.

Field Name	Type	Description
failOffset	**Date**	Specifies the time (in seconds) that the threshold condition must exist before the trigger fires.
flags	**Set<TriggerFlag>**	
interval	**Boolean**	When used in conjunction with MultiFire and RearmTime trigger will fire at regular intervals. Can be used with TriggerEventType.EPOCH to create a Standing Trigger. Defaults to false
maxRetry	**Integer**	Specifies the number of times Action will be attempted before the trigger is designated a failure.
multiFire	**Boolean**	Specifies whether this trigger can fire multiple times. Defaults to false.
name	**String**	Trigger name - can be auto assigned by moab or requested. Alphanumeric up to 16 characters in length
objectId	**String**	The ID of the object which this is attached to.
objectType	**String**	The type of object which this is attached to. Possible values: • vm - Virtual Machine
offset	**Date**	Relative time offset from event when trigger can fire.
period	**TriggerPeriod**	Can be used in conjunction with Offset to have a trigger fire at the beginning of the specified period. Can be used with EType epoch to create a standing trigger.
rearmTime	**Date**	Time between MultiFire triggers; rearm time is enforced from the trigger event time.
requires	**String**	Variables this trigger requires to be set or not set before it will fire. Preceding the string with an exclamation mark (!) indicates this variable must NOT be set. Used in conjunction with Sets to create trigger dependencies.

Field Name	Type	Description
sets	**String**	Variable values this trigger sets upon success or failure. Preceding the string with an exclamation mark (!) indicates this variable is set upon trigger failure. Preceding the string with a caret (^) indicates this variable is to be exported to the parent object when the current object is destroyed through a completion event. Used in conjunction with Requires to create trigger dependencies.
threshold	**String**	Reservation usage threshold - When reservation usage drops below Threshold, trigger will fire. Threshold usage support is only enabled for reservations and applies to percent processor utilization. gmetric thresholds are supported with job, node, credential, and reservation triggers. See Threshold Triggers in the Moab Workload Manager documentation for more information.
timeout	**Date**	Time allotted to this trigger before it is marked as unsuccessful and its process (if any) killed.
unsets	**String**	Variable this trigger destroys upon success or failure.

TriggerActionType

This enumeration specifies the action type of a trigger.

Value	Description
CANCEL	Only apply to reservation triggers
CHANGE_PARAM	
JOB_PREEMPT	This indicates that the trigger should preempt all jobs currently allocating resources assigned to the trigger's parent object. Only apply to reservation triggers.
MAIL	
THRESHOLD	
INTERNAL	

Value	Description
EXEC	

TriggerEventType

This enumeration specifies the event type of a trigger.

Value	Description
CANCEL	
CHECKPOINT	
CREATE	
END	
EPOCH	
FAIL	
HOLD	
MIGRATE	
MODIFY	
PREEMPT	
STANDING	
START	
THRESHOLD	

TriggerFlag

This enumeration specifies a flag belonging to a trigger.

Value	Description
ATTACH_ERROR	If the trigger outputs anything to stderr, Moab will attach this as a message to the trigger object.
CLEANUP	If the trigger is still running when the parent object completes or is canceled, the trigger will be killed.
CHECKPOINT	Moab should always checkpoint this trigger. See Checkpointing a Trigger in the Moab Workload Manager documentation for more information.
GLOBAL_VARS	The trigger will look in the name space of all nodes with the globalvars flag in addition to its own name space. A specific node to search can be specified using the following format: globalvars+node_id
INTERVAL	Trigger is periodic.
MULTIFIRE	Trigger can fire multiple times.
OBJECT_XML_ STDIN	Trigger passes its parent's object XML information into the trigger's stdin. This only works for exec triggers with reservation type parents.
USER	The trigger will execute under the user ID of the object's owner. If the parent object is sched, the user to run under may be explicitly specified using the format user+<username>, for example flags=user+john:
GLOBAL_ TRIGGER	The trigger will be (or was) inserted into the global trigger list.
ASYNCHRONOUS	An asynchronous trigger.
LEAVE_FILES	Do not remove stderr and stdout files.
PROBE	The trigger's stdout will be monitored.
PROBE_ALL	The trigger's stdout will be monitored.
GENERIC_ SYSTEM_JOB	The trigger belongs to a generic system job (for checkpointing).

Value	Description
REMOVE_STD_FILES	The trigger will delete stdout/stderr files after it has been reset.
RESET_ON_MODIFY	The trigger resets if the object it is attached to is modified, even if multifire is not set.
SOFT_KILL	By default, a `SIGKILL` (kill -9) signal is sent to the kill the script when a trigger times out. This flag will instead send a `SIGTERM` (kill -15) signal to kill the script. The `SIGTERM` signal will allow the script to trap the signal so that the script can clean up any residual information on the system (instead of just dying, as with the `SIGKILL` signal). NOTE: A timed-out trigger will only receive one kill signal. This means that if you specify this flag, a timed-out trigger will only receive the `SIGTERM` signal, and never the `SIGKILL` signal.

TriggerPeriod

This enumeration specifies the period of a trigger.

Value	Description
MINUTE	
HOUR	
DAY	
WEEK	
MONTH	

VMUsagePolicy

This enumeration describes the virtual machine requirements of a job

Value	Description
REQUIREPM	Requires a physical machine.
PREFPM	Prefers a physical machine.

Value	Description
CREATEVM	Creates a virtual machine.
CREATEPERSISTENTVM	Creates a virtual machine that doesn't go away after the job is done.
REQUIREVM	Requires a virtual machine.
PREFVM	Prefers a virtual machine.

Related topics

- **Job templates** on page 1547

Fields: Metric Types

See the associated **Metric types** on page 1549 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`metric-types`	**Permissions** on page 1571
Hooks filename	`metric-types.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

MetricType

Represents a metric visible and known to Moab Workload Manager.

Field Name	Type	Description
id	**String**	The unique ID of this metric type.

API version 2

MetricType

Represents a metric visible and known to Moab Workload Manager.

Field Name	Type	Description
id	**String**	The unique ID of this metric type.

Related topics

- **Metric types** on page 1549

Fields: Nodes

See the associated **Nodes** on page 1551 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`nodes`	**Permissions** on page 1571
Hooks filename	`nodes.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	Yes	**Distinct** on page 1504

API version 3

Node

This class represents a node in the Moab Workload Manager. Moab recognizes a node as a collection of resources with a particular set of associated attributes. This definition is similar to the traditional notion of a node found in a Linux cluster or supercomputer wherein a node is defined as one or more CPUs, associated memory, and possibly other compute resources such as local disk, swap, network adapters, and software licenses. Additionally, this node is described by various attributes such as an architecture type or operating system. Nodes range in size from small uniprocessor PCs to large symmetric multiprocessing (SMP) systems where a single node may consist of hundreds of CPUs and massive amounts of memory.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier of this node. Note: this field is not user-assigned and is generated by the database.
architecture	**String**	No	This node's processor architecture.
attributes	**Map<String, Map>**	No	Attributes is a map of attribute names to tuples (maps) that describe the scheduling attributes of a node. Each tuple should contain the following entries: • **value** - the attribute value • **displayValue** - the attribute display value
classes	**Set<String>**	No	The classes that this node can be scheduled for.
featuresCustom	**Set<String>**	Yes	The features this node advertises which are customizable at run-time. This can be used to define node sets. (See also: featuresReported.)

Field Name	Type	PUT	Description
featuresReported	**Set<String>**	No	The features this node advertises which are reported by resource managers or are present in the Moab Workload Manager configuration. This can be used to define node sets. (See also: featuresCustom.)
index	**Integer**	No	The index for this node as reported by the resource manager.
ipAddress	**String**	No	This node's IPv4 address.
isHypervisor	**Boolean**	No	True if the node is a hypervisor, false otherwise. This is based on the NodeOperatingSystemInformation.hypervisorType field. If hypervisorType is present, the node is a hypervisor. If it is null, then it is not a hypervisor.
jobs	**Set<DomainProxy>**	No	Jobs associated with this node.
lastUpdatedDate	**Date**	No	The timestamp of the last moment when this node was updated. There is no guarantee that all user modifications to a node would be picked up. This will also be changed every RMPOLLINTERVAL even if a resource manager does not report information on this node.
messages	**Set<Message>**	Yes	The list of messages attached to this node. They can be attached by admins, the resource manager layer, or triggers.

Field Name	Type	PUT	Description
metrics	**Map<String, Double>**	Yes	Metrics are the measurable, quantitative, and changing aspects of this node. They are used to define workload placement, attach triggers, etc. There are some built-in metrics: • **speed** - A number from 0.0 to 1.0 describing the relative speed of the system for computational tasks. This is a composite metric, and is defined on a per-site basis. • **cpuLoad** - This is the CPU load on this node. This value is defined at the resource manager layer, but is generally defined on a per-operating system basis. For example, Unix-based OS's use some aspect of the Unix load average, as reported by the resource manager layer, while Windows-based OS's use CPU utilization.
migrationDisabled	**Boolean**	No	True if VM migration is disabled on this node.
name	**String**	No	The name of this node. This name is unique *per instance* of Moab Workload Manager (i.e. not globally).
operatingSystem	**NodeOperatingSystemInformation**	Yes	Describes the current or expected operating system image information for this node. The operatingSystem.image field can be changed using PUT.
partition	**String**	Yes	The partition this node belongs to.
processorSpeed	**Integer**	No	The speed, in MHz, or the processors on this node.

Field Name	Type	PUT	Description
profilingEnabled	**Boolean**	No	Indicates whether historical data gathering and reporting is enabled for this node. This is also controlled by the same setting on the default node (i.e. all nodes). If set to false (default), node statistics are not gathered.
rack	**Integer**	No	The rack where this node is located in the datacenter/cluster.
reservations	**Set<DomainProxy>**	No	Reservations associated with this node.
resourceManagerMessages	**Map<String, Map>**	No	The resource manager messages for this node. Each key is the name of a resource manager, and the value is the message that the resource manager has posted onto the node.
resourceManagers	**Set<NodeResourceManager>**	No	The resource managers that are reporting or have previously reported this node. Each object also contains information on the resource manager reports.
resources	**Map<String, Resource>**	No	Contains references of a string representing a resource name to a resource object detailing the amount of the resource that is available, configured, etc. Each key is the name of the resource, which equates to the generic resource identifier or one of "processors", "memory", "disk", or "swap". This name may be used as an id in the resource types web service.
slot	**Integer**	No	The slot in the rack where this node is located.

Field Name	Type	PUT	Description
states	[NodeStateInformation]()	Yes	This node's state. The states.powerState and states.state fields can be changed using PUT.
triggers	[Set<DomainProxy>]()	No	Triggers associated with this node.
type	[NodeType]()	No	The type of this node is governed by the types of resources it offers.
variables	**Map<String, Map>**	Yes	Variables is a map of key-value pairs, synonymous, but not directly related to, environment variables. They provide the mechanism to store arbitrary metdata which is useful to external systems in memory on this node.
virtualContainers	[Set<DomainProxy>]()	No	The set of virtual containers that directly (not recursively) contain this node.
virtualMachines	[Set<DomainProxy>]()	No	Virtual machines associated with this node.

DomainProxy

A reference to an object contained within an object. For example, a Virtual Machine object contains a reference to the Node on which it is running. That reference is represented by this class.

Field Name	Type	PUT	Description
name	**String**	No	The name of the object.

Message

Field Name	Type	PUT	Description
count	**Integer**	No	The number of times this message has occurred.
createdDate	**Date**	No	The date this message was created.

Field Name	Type	PUT	Description
expireDate	**Date**	No	The date this message expires.
message	**String**	Yes	The message itself.

NodeOperatingSystemInformation

Describes the current or expected operating system image information for a node.

Field Name	Type	PUT	Description
hypervisorType	**String**	No	The hypervisor technology that this node uses. May be null if the node is not a hypervisor.
image	**String**	Yes	The name of the operating system currently running on this node. In cloud mode, this corresponds to the ID or name of an image in the image management API in MWS. (See also: Image.id, Image.name.)
imageExpected	**String**	No	The name of the image that was requested to run on this node (i.e. with `mnodectl -m os=myOs`). In cloud mode, this corresponds to the ID or name of an image in the image management API in MWS. (See also: Image.id, Image.name.)
imageLastUpdatedDate	**Date**	No	The last time the image of this node was modified.
imagesAvailable	**Set<String>**	No	The list of image names that can be applied to this node. In cloud mode, this corresponds to IDs or names of images in the image management API in MWS. (See also: Image.id, Image.name.)
virtualMachineImages	**Set<String>**	No	The list of virtual machine image names the node is capable of supporting. In cloud mode, this corresponds to IDs or names of images in the image management API in MWS. (See also: Image.id, Image.name.)

NodeResourceManager

Field Name	Type	PUT	Description
isMaster	**Boolean**	No	Indicates whether this resource manager is the "master" of this Node. If true, it means that this resource manager has the final say on all properties reported about this Node. Note that the first resource manager to report a node is the master resource manager.
name	**String**	No	The name of the resource manager, according to Moab. This name appears in both the RMCFG parameter, and when diagnosing resource managers (e.g. `mdiag -R`).
stateReported	**NodeState**	No	The state reported by this resource manager. See the State section for more details.

NodeState

This enumeration tracks the state of a node.

Value	Description
NONE	The node is set to none by the resource manager.
DOWN	The node is not available for workload.
IDLE	The node is available for workload but is not running anything.
BUSY	The node is running workload and cannot accept more.
RUNNING	The node is running workload and can accept more.
DRAINED	The node has been sent the drain request and has no workload on it.
DRAINING	The node has been sent the drain request, but still has workload on it.
FLUSH	The node is being reprovisioned.
RESERVED	The node is being reserved. This is an internal Moab state.
UNKNOWN	The state of the node is unknown.

Resource

Represents counts of resources available, configured, etc.

Field Name	Type	PUT	Description
available	**Integer**	No	The amount of this resource that is currently available for allocation to workload.
configured	**Integer**	No	The amount of this resource that is considered possible to schedule. Overcommit specifically applies to this, in other words, `configured = overcommitFactor * real`.
dedicated	**Integer**	No	The amount of this resource that has been allocated for running workload. When used in a job submission, this number is the amount of the resource required by the job.
real	**Integer**	No	The amount of this resource that physically exists on the node. Overcommit specifically doesn't apply to this. Note that overcommit currently only applies to "processors" and "memory", and so, for most cases, real and configured will always be the same.
utilized	**Integer**	No	The amount of this resource that is currently reported as utilized by resource managers.

NodeStateInformation

Field Name	Type	PUT	Description
powerState	**NodePower**	Yes	The state of the node's power system, as reported by the RM layer. Modifying the powerState is possible, and, if Moab is configured properly, a request will be made to modify the power state accordingly.
powerStateExpected	**NodePower**	No	The expected state of the node's power system. If a user has requested that a node be powered off (e.g. by modifying the powerState attribute to NodePower.OFF), the requested state will be shown in this field until the state change is completed. If there is no pending power change request, this will be null.

Field Name	Type	PUT	Description
state	**NodeState**	Yes	The scheduling state of the Node, as reported by the resource management layer.
stateExpected	**NodeState**	No	The scheduling state of the Node, as expected by Moab. For example, Moab may think that a Node is "Busy" because it has allocated all configured resources, but a resource manager may report the state as "Running" based on actual utilization of the resources.
stateLastUpdatedDate	**Date**	No	A timestamp recording when the state of the Node was last modified.
subState	**String**	No	A text description of the state of the Node, with the intention of giving more details. Resource Managers may use this field to further describe the state being reported. Resource Managers should provide documented meaning to the possible sub-states that they can report.
subStateLast	**String**	No	The previous sub-state of the Node as reported by the resource management layer.
subStateLastUpdatedDate	**Date**	No	A timestamp recording when the sub-state was last modified.

NodePower

Represents the various options for a Node's power state.

Value	Description
NONE	
ON	
OFF	

NodeType

Represents the type of node as reported by a resource manager.

Value	Description
Compute	
License	
Network	
Storage	

API version 2

Node

This class represents a node in the Moab Workload Manager. Moab recognizes a node as a collection of resources with a particular set of associated attributes. This definition is similar to the traditional notion of a node found in a Linux cluster or supercomputer wherein a node is defined as one or more CPUs, associated memory, and possibly other compute resources such as local disk, swap, network adapters, and software licenses. Additionally, this node is described by various attributes such as an architecture type or operating system. Nodes range in size from small uniprocessor PCs to large symmetric multiprocessing (SMP) systems where a single node may consist of hundreds of CPUs and massive amounts of memory.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier of this node. Note: this field is not user-assigned and is generated by the database.
architecture	**String**	No	This node's processor architecture.
attributes	**Map<String, Map>**	No	Attributes is a map of attribute names to tuples (maps) that describe the scheduling attributes of a node. Each tuple should contain the following entries: • **value** - the attribute value • **displayValue** - the attribute display value
classes	**Set<String>**	No	The classes that this node can be scheduled for.
featuresCustom	**Set<String>**	Yes	The features this node advertises which are customizable at run-time. This can be used to define node sets. (See also: featuresReported.)

Field Name	Type	PUT	Description
featuresReported	**Set<String>**	No	The features this node advertises which are reported by resource managers or are present in the Moab Workload Manager configuration. This can be used to define node sets. (See also: featuresCustom.)
index	**Integer**	No	The index for this node as reported by the resource manager.
ipAddress	**String**	No	This node's IPv4 address.
isHypervisor	**Boolean**	No	True if the node is a hypervisor, false otherwise. This is based on the NodeOperatingSystemInformation.hypervisorType field. If hypervisorType is present, the node is a hypervisor. If it is null, then it is not a hypervisor.
jobs	**Set<DomainProxy>**	No	Jobs associated with this node.
lastUpdatedDate	**Date**	No	The timestamp of the last moment when this node was updated. There is no guarantee that all user modifications to a node would be picked up. This will also be changed every RMPOLLINTERVAL even if a resource manager does not report information on this node.
messages	**Set<Message>**	Yes	The list of messages attached to this node. They can be attached by admins, the resource manager layer, or triggers.

Field Name	Type	PUT	Description
metrics	**Map<String, Double>**	Yes	Metrics are the measurable, quantitative, and changing aspects of this node. They are used to define workload placement, attach triggers, etc. There are some built-in metrics: • **speed** - A number from 0.0 to 1.0 describing the relative speed of the system for computational tasks. This is a composite metric, and is defined on a per-site basis. • **cpuLoad** - This is the CPU load on this node. This value is defined at the resource manager layer, but is generally defined on a per-operating system basis. For example, Unix-based OS's use some aspect of the Unix load average, as reported by the resource manager layer, while Windows-based OS's use CPU utilization.
migrationDisabled	**Boolean**	No	True if VM migration is disabled on this node.
name	**String**	No	The name of this node. This name is unique *per instance* of Moab Workload Manager (i.e. not globally).
operatingSystem	**NodeOperatingSystemInformation**	Yes	Describes the current or expected operating system image information for this node. The operatingSystem.image field can be changed using PUT.
partition	**String**	Yes	The partition this node belongs to.
processorSpeed	**Integer**	No	The speed, in MHz, or the processors on this node.

Field Name	Type	P UT	Description
profilingEnabled	**Boolean**	No	Indicates whether historical data gathering and reporting is enabled for this node. This is also controlled by the same setting on the default node (i.e. all nodes). If set to false (default), node statistics are not gathered.
rack	**Integer**	No	The rack where this node is located in the datacenter/cluster.
reservations	**Set<DomainProxy>**	No	Reservations associated with this node.
resourceManagerMessages	**Map<String, Map>**	No	The resource manager messages for this node. Each key is the name of a resource manager, and the value is the message that the resource manager has posted onto the node.
resourceManagers	**Set<NodeResourceManager>**	No	The resource managers that are reporting or have previously reported this node. Each object also contains information on the resource manager reports.
resources	**Map<String, Resource>**	No	Contains references of a string representing a resource name to a resource object detailing the amount of the resource that is available, configured, etc. Each key is the name of the resource, which equates to the generic resource identifier or one of "processors", "memory", "disk", or "swap". This name may be used as an id in the resource types web service.
slot	**Integer**	No	The slot in the rack where this node is located.

Field Name	Type	PUT	Description
states	[NodeStateInformation]()	Yes	This node's state. The states.powerState and states.state fields can be changed using PUT.
triggers	[Set<DomainProxy>]()	No	Triggers associated with this node.
type	[NodeType]()	No	The type of this node is governed by the types of resources it offers.
variables	**Map<String, Map>**	Yes	Variables is a map of key-value pairs, synonymous, but not directly related to, environment variables. They provide the mechanism to store arbitrary metdata which is useful to external systems in memory on this node.
virtualContainers	[Set<DomainProxy>]()	No	The set of virtual containers that directly (not recursively) contain this node.
virtualMachines	[Set<DomainProxy>]()	No	Virtual machines associated with this node.

DomainProxy

A reference to an object contained within an object. For example, a Virtual Machine object contains a reference to the Node on which it is running. That reference is represented by this class.

Field Name	Type	PUT	Description
name	**String**	No	The name of the object.

Message

Field Name	Type	PUT	Description
count	**Integer**	No	The number of times this message has occurred.
createdDate	**Date**	No	The date this message was created.

Field Name	Type	PUT	Description
expireDate	**Date**	No	The date this message expires.
message	**String**	Yes	The message itself.

NodeOperatingSystemInformation

Describes the current or expected operating system image information for a node.

Field Name	Type	PUT	Description
hypervisorType	**String**	No	The hypervisor technology that this node uses. May be null if the node is not a hypervisor.
image	**String**	Yes	The name of the operating system currently running on this node. In cloud mode, this corresponds to the ID or name of an image in the image management API in MWS. (See also: Image.id, Image.name.)
imageExpected	**String**	No	The name of the image that was requested to run on this node (i.e. with `mnodectl -m os=myOs`). In cloud mode, this corresponds to the ID or name of an image in the image management API in MWS. (See also: Image.id, Image.name.)
imageLastUpdatedDate	**Date**	No	The last time the image of this node was modified.
imagesAvailable	**Set<String>**	No	The list of image names that can be applied to this node. In cloud mode, this corresponds to IDs or names of images in the image management API in MWS. (See also: Image.id, Image.name.)
virtualMachineImages	**Set<String>**	No	The list of virtual machine image names the node is capable of supporting. In cloud mode, this corresponds to IDs or names of images in the image management API in MWS. (See also: Image.id, Image.name.)

NodeResourceManager

Field Name	Type	PUT	Description
isMaster	**Boolean**	No	Indicates whether this resource manager is the "master" of this Node. If true, it means that this resource manager has the final say on all properties reported about this Node. Note that the first resource manager to report a node is the master resource manager.
name	**String**	No	The name of the resource manager, according to Moab. This name appears in both the RMCFG parameter, and when diagnosing resource managers (e.g. `mdiag -R`).
stateReported	**NodeState**	No	The state reported by this resource manager. See the State section for more details.

NodeState

This enumeration tracks the state of a node.

Value	Description
NONE	The node is set to none by the resource manager.
DOWN	The node is not available for workload.
IDLE	The node is available for workload but is not running anything.
BUSY	The node is running workload and cannot accept more.
RUNNING	The node is running workload and can accept more.
DRAINED	The node has been sent the drain request and has no workload on it.
DRAINING	The node has been sent the drain request, but still has workload on it.
FLUSH	The node is being reprovisioned.
RESERVED	The node is being reserved. This is an internal Moab state.
UNKNOWN	The state of the node is unknown.

Resource

Represents counts of resources available, configured, etc.

Field Name	Type	PUT	Description
available	**Integer**	No	The amount of this resource that is currently available for allocation to workload.
configured	**Integer**	No	The amount of this resource that is considered possible to schedule. Overcommit specifically applies to this, in other words, `configured = overcommitFactor * real`.
dedicated	**Integer**	No	The amount of this resource that has been allocated for running workload. When used in a job submission, this number is the amount of the resource required by the job.
real	**Integer**	No	The amount of this resource that physically exists on the node. Overcommit specifically doesn't apply to this. Note that overcommit currently only applies to "processors" and "memory", and so, for most cases, real and configured will always be the same.
utilized	**Integer**	No	The amount of this resource that is currently reported as utilized by resource managers.

NodeStateInformation

Field Name	Type	PUT	Description
powerState	**NodePower**	Yes	The state of the node's power system, as reported by the RM layer. Modifying the powerState is possible, and, if Moab is configured properly, a request will be made to modify the power state accordingly.
powerStateExpected	**NodePower**	No	The expected state of the node's power system. If a user has requested that a node be powered off (e.g. by modifying the powerState attribute to NodePower.OFF), the requested state will be shown in this field until the state change is completed. If there is no pending power change request, this will be null.

Field Name	Type	PUT	Description
state	**NodeState**	Yes	The scheduling state of the Node, as reported by the resource management layer.
stateExpected	**NodeState**	No	The scheduling state of the Node, as expected by Moab. For example, Moab may think that a Node is "Busy" because it has allocated all configured resources, but a resource manager may report the state as "Running" based on actual utilization of the resources.
stateLastUpdatedDate	**Date**	No	A timestamp recording when the state of the Node was last modified.
subState	**String**	No	A text description of the state of the Node, with the intention of giving more details. Resource Managers may use this field to further describe the state being reported. Resource Managers should provide documented meaning to the possible sub-states that they can report.
subStateLast	**String**	No	The previous sub-state of the Node as reported by the resource management layer.
subStateLastUpdatedDate	**Date**	No	A timestamp recording when the sub-state was last modified.

NodePower

Represents the various options for a Node's power state.

Value	Description
NONE	
ON	
OFF	

NodeType

Represents the type of node as reported by a resource manager.

Value	Description
Compute	
License	
Network	
Storage	

Related topics

- **Nodes** on page 1551

Fields: Notification Conditions

See the associated **Notification conditions** on page 1558 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`notification-conditions`	**Permissions** on page 1571
Hooks filename	`notification-conditions.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	Yes	**Distinct** on page 1504

API version 3

NotificationCondition

A notification condition is related to an Event, but differs in three distinct areas:

- Notification conditions are a persistent condition of the system or a component rather than a single occurrence.
  - They are ongoing rather than reoccurring, which is why they are generated from NotificationConditions.
  - They may be observed many times, but the condition is always the same.
  - A good test for this is if something "is" wrong rather than something "went" wrong.
- Notification conditions can be acted on to result in a resolved state, mean the administrator or user can and must take actions to "fix" the condition or problem.
- Notification conditions contain state information based on administrator or user input, meaning that they contain information about the condition (similar to events), but also contain the "status" of the administrator's view of the notification, whether it is currently open, dismissed, or ignored.

In general, questions may be asked to ascertain whether an Event or a Notification Condition is the right fit for an occurrence. These questions, along with some sample situations, are provided below.

- Is the occurrence the root cause of a potentially ongoing condition?
  - A VM migration failed because the VM's state was unknown. The root cause was that the state was unknown, not that the VM migration failed. Therefore, VM migration failed would be an event, while the unknown state would be a notification condition.
  - A VM service provision fails because there are no hypervisors that satisfy the requirements. This would be an event. Note that there may be a notification related to this failure, such as a service template requires a feature that does not exist on *any* hypervisors in the system, but this would be distinctly detected and managed from the provision failure event.

- A request to MWS failed because the connection between MWM and MongoDB was misconfigured. The failed request may be represented as an event, but a notification condition should exists that the connection between MWM and MongoDB was down.

- Can an administrator or user affect the outcome of the occurrence?
  - The outcome of a VM migration failing is in the past and cannot be changed by the administrator. However, the outcome of a *future* VM migration may be changed when the administrator resolves the root problem (i.e. VM state is unknown).

A notification condition is an observed condition for which Notifications are created. These conditions are created or updated on every PUT request based on the NotificationCondition.escalationLevel, NotificationCondition.origin, NotificationCondition.message, NotificationCondition.objectType, and NotificationCondition.objectId fields. When notifications are requested, these observed conditions are used to create the notifications for the requesting user.

While notification conditions may not be deleted, they "expire" after a specified amount of time and are no longer considered as active conditions for which notifications are created.

Field Name	Type	PUT	Description
id	**String**	No	The identifier of the condition.
createdDate	**Date**	No	The date that the condition first started appearing.
details	**Map<String, Map>**	No	Arbitrary storage of details for this notification. This could include "pluginType", "pluginId", etc.
escalationLevel	**EscalationLevel**	No	The escalation level of the condition. This indicates who should care about the condition or who can respond to it. This may NOT be EscalationLevel.INTERNAL.
expirationDate	**Date**	No	The date at which the condition is considered "expired" and notifications are no longer created for it. This is typically set using the expirationDuration field.

Field Name	Type	PUT	Description
expirationDuration	**Long**	No	The duration in seconds that may pass before a notification will not be created for a user. Effectively this can disable notifications from being created if they are too old. When this field is set, it will set the expirationDate field automatically each time the condition is updated or on creation. This field must be set to 1 or greater or else set to null.
message	**String**	No	A message detailing the notification and why it exists, with possible action items.
objectId	**String**	No	The identifier of the object which this notification affects, such as "node1" or "vm1".
objectType	**String**	No	The object type that this notification affects, such as "Node", "VM", "System", etc.
observedDate	**Date**	No	The latest date that the condition was observed. If this field is not set in an update request, it will automatically be set to the current date.
origin	**String**	No	The origin of the notification.
tenant	**Map<String, Map>**	No	The tenant that this notification came from. (contains tenant id and name)

EscalationLevel

Value	Description
USER	
POWER_USER	
ADMIN	
INTERNAL	

Related topics

- **Notification conditions** on page 1558

Fields: Notifications

See the associated **Notifications** on page 1563 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`notifications`	**Permissions** on page 1571
Hooks filename	`notifications.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	Yes	**Distinct** on page 1504

API version 3

Notification

Notifications, while related to Events, are used for different purposes. See NotificationCondition for more information on when notifications should be used as opposed to events.

Notifications are a per-user representation of all notification conditions present in the system at any one time. When an administrator or user requests this resource, notifications are automatically created from the notification conditions that they have access to (determined by the Notification.tenant or the NotificationCondition.escalationLevel fields).

Notifications are expected to contain messages and details that may be understood by a user or admin depending on the escalation level, and contain fields that control whether the user or admin will be notified of future updates to their corresponding condition.

Notifications cannot be deleted, but they can be marked as ignored (see Notification.ignoredDate or dismissed (see Notification.dismissedDate).

Field Name	Type	PUT	Description
id	**String**	No	The identifier of the notification.
conditionId	**String**	No	The identifier of the NotificationCondition from which this notification was created.
createdDate	**Date**	No	The date that the notification condition first appeared.
details	**Map<String, Map>**	No	Arbitrary storage of details for this notification. This could include "pluginType", "pluginId", etc.
dismissedDate	**Date**	No	The date that the notification was dismissed by a user or admin, meaning that they acknowledged the notification and wanted to know of future updates to this notification. This field is cleared every time the attached notification condition is updated/observed again. (See also: conditionId.)
ignoredDate	**Date**	No	The date that the notification was ignored by a user or admin, meaning that they acknowledged the notification now and in the future and did not wish to know of any updates. This field is never cleared, even if the attached notification condition is updated/observed again.
message	**String**	No	A message detailing the notification and why it exists, with possible action items.

Field Name	Type	PUT	Description
objectId	**String**	No	The identifier of the object which this notification affects, such as "node1" or "vm1".
objectType	**String**	No	The object type that this notification affects, such as "Node", "VM", "System", etc.
observedDate	**Date**	No	The latest date that the notification condition was observed. If this field, ignoredDate, and dismissedDate are not set during an update (i.e. a user/admin is not ignoring or dismissing the notification), this field will automatically be set to the current date.
origin	**String**	No	The origin of the notification.
user	**String**	No	The user that this notification was created for.

Related topics

- **Notifications** on page 1563

Fields: Plugins

See the associated **Plugins** on page 1577 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`plugins`	**Permissions** on page 1571
Hooks filename	`plugins.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

PluginInstance

This class represents a configured plugin created from a plugin type.

Field Name	Type	POST	PUT	Description
id	**String**	Yes	No	Unique identifier for the plugin. Must contain at least one letter and must also start with a letter. Reserved IDs are "all" and "moab". If these are used an error with be returned.
autoStart	**Boolean**	Yes	Yes	Whether the plugin should start automatically when created.
config	**Map<String, Map>**	Yes	Yes	The configuration of the plugin. Plugin types may define constraints on the configuration, therefore it is recommended to view the plugin type's documentation for more information on required and optional fields. Regardless, the plugin configuration supports arbitrary keys and values.
dateCreated	**Date**	No	No	The date that this plugin was created.
lastPollDate	**Date**	No	No	The date of the last polling event that occurred. This may be null if the plugin is in the STOPPED state or has not yet been polled.
lastUpdated	**Date**	No	No	The date that this plugin was last updated.
nextPollDate	**Date**	No	No	The date of the next polling event that is scheduled to occur. This may be null if the plugin is in the STOPPED state.
pluginType	**String**	Yes	No	The plugin name as in Native or Example for the plugin called ExamplePlugin.
pollInterval	**Integer**	Yes	Yes	The polling interval to use for the plugin in seconds. This is ignored if the plugin type does not support polling.

Field Name	Type	POST	PUT	Description
precedence	**Long**	Yes	Yes	The precedence of this plugin, with the lowest value being the highest precedence. Minimum of 1. This is used when doing data consolidation when reporting current state data. Lower numbers results in a higher precedence (i.e. 1 is higher precedence than 10). If not specified during creation, this will be automatically set to 1 for the first plugin created, then 1 greater for each subsequently created plugin (i.e. 1 for plugin1, 2 for plugin2, etc). It is always set to 1 greater than the plugin with the greatest precedence number (i.e. 11 if two plugins exist with precedence 1 and 10).
state	**PluginState**	No	No	The current state of the plugin. Defaults to PluginState.STOPPED.

PluginState

Represents the current state of a plugin.

Value	Description
STOPPED	The plugin is created and ready for use, but is not currently receiving any events
STARTED	The plugin is currently receiving events and is working correctly.
PAUSED	The plugin is currently not receiving any events but is also not stopped. This should be used when polling or other events should stop only temporarily without firing the stop events.
ERRORED	MWS has detected an error with the plugin and has automatically stopped it. Errors could be due to the following reasons: 1. An invalid configuration was detected when running the AbstractPlugin.configure method. 2. An unexpected exception was thrown during an event, such as during polling.

API version 2

PluginInstance

This class represents a configured plugin created from a plugin type.

Field Name	Type	POST	PUT	Description
id	**String**	Yes	No	Unique identifier for the plugin. Must contain at least one letter and must also start with a letter. Reserved IDs are "all" and "moab". If these are used an error with be returned.
autoStart	**Boolean**	Yes	Yes	Whether the plugin should start automatically when created.
config	**Map<String, Map>**	Yes	Yes	The configuration of the plugin. Plugin types may define constraints on the configuration, therefore it is recommended to view the plugin type's documentation for more information on required and optional fields. Regardless, the plugin configuration supports arbitrary keys and values.
dateCreated	**Date**	No	No	The date that this plugin was created.
lastPollDate	**Date**	No	No	The date of the last polling event that occurred. This may be null if the plugin is in the STOPPED state or has not yet been polled.
lastUpdated	**Date**	No	No	The date that this plugin was last updated.
nextPollDate	**Date**	No	No	The date of the next polling event that is scheduled to occur. This may be null if the plugin is in the STOPPED state.
pluginType	**String**	Yes	No	The plugin name as in Native or Example for the plugin called ExamplePlugin.
pollInterval	**Integer**	Yes	Yes	The polling interval to use for the plugin in seconds. This is ignored if the plugin type does not support polling.

Field Name	Type	POST	PUT	Description
precedence	**Long**	Yes	Yes	The precedence of this plugin, with the lowest value being the highest precedence. Minimum of 1. This is used when doing data consolidation when reporting current state data. Lower numbers results in a higher precedence (i.e. 1 is higher precedence than 10). If not specified during creation, this will be automatically set to 1 for the first plugin created, then 1 greater for each subsequently created plugin (i.e. 1 for plugin1, 2 for plugin2, etc). It is always set to 1 greater than the plugin with the greatest precedence number (i.e. 11 if two plugins exist with precedence 1 and 10).
state	**PluginState**	No	No	The current state of the plugin. Defaults to PluginState.STOPPED.

PluginState

Represents the current state of a plugin.

Value	Description
STOPPED	The plugin is created and ready for use, but is not currently receiving any events
STARTED	The plugin is currently receiving events and is working correctly.
PAUSED	The plugin is currently not receiving any events but is also not stopped. This should be used when polling or other events should stop only temporarily without firing the stop events.
ERRORED	MWS has detected an error with the plugin and has automatically stopped it. Errors could be due to the following reasons: 1. An invalid configuration was detected when running the AbstractPlugin.configure method. 2. An unexpected exception was thrown during an event, such as during polling.

Related topics

- **Plugins** on page 1577

Fields: Plugin Types

See the associated **Plugin types** on page 1585 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`plugin-types`	**Permissions** on page 1571
Hooks filename	`plugin-types.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

PluginType

Represents a MWS plugin type. All fields in this class are generated from plugin project and type metadata and cannot be modified directly. Consequentially, all fields are only valid for list/show/GET operations.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier of the plugin type. This is based on the class name of the plugin. Ex: **Plugin Class Name -> ID** NativePlugin -> Native MSMPlugin -> MSM MyExamplePlugin -> MyExample
author	**String**	No	The main author (company or person) of the plugin type.
commonsVersion	**String**	No	A string representing the restriction on which version of the plugin framework (plugins-commons dependency) is required for the plugin type. In the format 'COMMONS_VERSION > *', meaning that any version greater or equal to COMMONS_VERSION is valid.
description	**String**	No	The full description of the plugin type.
documentationLink	**String**	No	A full URL to the complete documentation for the plugin type.
email	**String**	No	The email of the author.
eventComponent	**Integer**	No	The event component ID of the plugin type. This should be unique for each plugin type and should be 1 or greater.
initialPlugins	**Map<String, Map>**	No	Represents the plugins that are initially configured when the plugin type is loaded. Each key represents the plugin ID.
issueManagementLink	**String**	No	A full URL to the issue management system or project for the plugin type.

Field Name	Type	PUT	Description
license	**String**	No	The license of this plugin type, typically APACHE.
mwsVersion	**String**	No	A string representing the restriction on which version of MWS is required for the plugin type. In the format 'MWS_VERSION > *', meaning that any version greater or equal to MWS_VERSION is valid.
pollMethod	**boolean**	No	Indicates whether the plugin type has a defined 'poll' method (event handler) or not.
realizedEventComponent	**Integer**	No	The fully realized event component ID of the plugin type, including the MWS bits. This should take the form of 0x201. If the eventComponent is not set, this will be 0x2FF, meaning the component ID is an unknown plugin type.
scmLink	**String**	No	A full URL to the Source Control Management (SCM) system or project for the plugin type.
title	**String**	No	A short name describing the plugin type.
website	**String**	No	The website of the author.

API version 2

PluginType

Represents a MWS plugin type. All fields in this class are generated from plugin project and type metadata and cannot be modified directly. Consequentially, all fields are only valid for list/show/GET operations.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier of the plugin type. This is based on the class name of the plugin. Ex: **Plugin Class Name -> ID** NativePlugin -> Native MSMPlugin -> MSM MyExamplePlugin -> MyExample
author	**String**	No	The main author (company or person) of the plugin type.
commonsVersion	**String**	No	A string representing the restriction on which version of the plugin framework (plugins-commons dependency) is required for the plugin type. In the format 'COMMONS_VERSION > *', meaning that any version greater or equal to COMMONS_VERSION is valid.
description	**String**	No	The full description of the plugin type.
documentationLink	**String**	No	A full URL to the complete documentation for the plugin type.
email	**String**	No	The email of the author.
eventComponent	**Integer**	No	The event component ID of the plugin type. This should be unique for each plugin type and should be 1 or greater.
initialPlugins	**Map<String, Map>**	No	Represents the plugins that are initially configured when the plugin type is loaded. Each key represents the plugin ID.
issueManagementLink	**String**	No	A full URL to the issue management system or project for the plugin type.

Field Name	Type	PUT	Description
license	**String**	No	The license of this plugin type, typically APACHE.
mwsVersion	**String**	No	A string representing the restriction on which version of MWS is required for the plugin type. In the format 'MWS_VERSION > *', meaning that any version greater or equal to MWS_VERSION is valid.
pollMethod	**boolean**	No	Indicates whether the plugin type has a defined 'poll' method (event handler) or not.
realizedEventComponent	**Integer**	No	The fully realized event component ID of the plugin type, including the MWS bits. This should take the form of 0x201. If the eventComponent is not set, this will be 0x2FF, meaning the component ID is an unknown plugin type.
scmLink	**String**	No	A full URL to the Source Control Management (SCM) system or project for the plugin type.
title	**String**	No	A short name describing the plugin type.
website	**String**	No	The website of the author.

Related topics

- **Plugin types** on page 1585

Fields: Policies

See the associated **Policies** on page 1589 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`policies`	**Permissions** on page 1571
Hooks filename	`policies.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	Yes	**Distinct** on page 1504

API version 3

Policy

A Moab Workload Manager policy which can affect scheduling decisions such as resource allocation. A policy contains state, identifying information, a priority, and metadata about the policy.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier for the policy. Must contain only lowercase letters and dashes, such as 'auto-vm-migration'.
conflicted	**Boolean**	No	Signifies whether any other policies are currently activated that potentially conflict with this policy. If true, it signifies a **potential** conflict.
description	**String**	No	The user friendly description of the policy.
name	**String**	No	The user friendly name of the policy.
potentialConflicts	**Set<String>**	No	A set of policy IDs that may potentially conflict with this policy.
priority	**Integer**	No	Indicates the absolute priority of the policy with respect to others. It is possible that more than one policy has the same priority. The higher the number, the greater the priority. Minimum is 0.
state	**PolicyState**	Yes	Defines the current state of the policy: enabled or disabled. Defaults to PolicyState.DISABLED.
tags	**Set<String>**	No	A set of strings that can be used to aid in filtering or querying policies.
types	**Set<String>**	No	A set of categories or types that the policy is included in. This may be used to filter or query on groups of policies.

PolicyState

Represents the state of a policy. A policy may only be enabled or disabled.

Value	Description
ENABLED	The policy is enabled or active.
DISABLED	The policy is disabled or inactive.

AutoVMMigrationPolicy

The Moab policy used to enabled and configure policy-based VM migration. Using information about data center applications and server load, Moab can aim to keep VMs in the data center optimally distributed across all hypervisors.

This class inherits fields from Policy.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier for the policy. Must contain only lowercase letters and dashes, such as 'auto-vm-migration'.
conflicted	**Boolean**	No	Signifies whether any other policies are currently activated that potentially conflict with this policy. If true, it signifies a **potential** conflict.
description	**String**	No	The user friendly description of the policy.
genericMetricThresholds	**Map<String, Double>**	Yes	A map of generic metric pairings where each value must be greater than or equal to 0 such as: `METRIC1 => 5.6 METRIC2 => 0.0 METRIC3 => 102.4`
memoryUtilizationThreshold	**Double**	Yes	Defines the utilization threshold for memory. This must be greater than 0 and less than or equal to 1. A value of 1 effectively disables the threshold.
migrationAlgorithmType	**AutoVMMigrationPolicyType**	Yes	Configures the VM migration algorithm utilized when the policy is active. Defaults to NONE. When ENABLED, this must **not** be set to NONE.

Field Name	Type	PUT	Description
name	**String**	No	The user friendly name of the policy.
potentialConflicts	**Set<String>**	No	A set of policy IDs that may potentially conflict with this policy.
priority	**Integer**	No	Indicates the absolute priority of the policy with respect to others. It is possible that more than one policy has the same priority. The higher the number, the greater the priority. Minimum is 0.
processorUtilizationThreshold	**Double**	Yes	Defines the load utilization threshold for processors. This must be greater than 0 and less than or equal to 1. A value of 1 effectively disables the threshold.
state	**PolicyState**	Yes	Defines the current state of the policy: enabled or disabled. Defaults to PolicyState.DISABLED.
tags	**Set<String>**	No	A set of strings that can be used to aid in filtering or querying policies.
types	**Set<String>**	No	A set of categories or types that the policy is included in. This may be used to filter or query on groups of policies.

AutoVMMigrationPolicyType

Represents the algorithm used to migrate VMs when the AutoVMMigrationPolicy is used.

Value	Description
NONE	Used when the Auto VM Migration policy is currently disabled in Moab and before any settings are saved the first time. For example, if the policy is disabled on the first read of Moab policies, the AutoVMMigrationPolicy.migrationAlgorithmType will be set to NONE. If the policy is enabled and the type is set to OVERCOMMIT, followed by a disabling of the policy, it will then be represented as having a state of DISABLED with a migrationAlgorithmType of OVERCOMMIT.
OVERCOMMIT	Use the "overcommit" algorithm for migration. The goal of this algorithm is to equalize loads across hypervisors as migrations are queued due to overcommit conditions. This places VMs to be migrated on the least-loaded hypervisor available.
CONSOLIDATION	Use the "consolidation" algorithm for migration. The goal of this algorithm is to load hypervisors as close to thresholds as possible, without exceeding them. This policy places VMs to be migrated on the most loaded hypervisor possible, within these constraints. A second loop of this policy will select lightly-loaded hypervisors to be evacuated completely.

PolicyState

Represents the state of a policy. A policy may only be enabled or disabled.

Value	Description
ENABLED	The policy is enabled or active.
DISABLED	The policy is disabled or inactive.

HVAllocationOvercommitPolicy

The Hypervisor Allocation Overcommit policy controls how many virtual machines can be placed on a hypervisor. By enabling this policy, you are allowing Moab to allocate more resoures to a set of virtual machines than a hypervisor may actually have. This is possible due to virtualization. In other words, this policy allows you to set the high-water mark for virtual machine allocation for hypervisors. At least one of these limits must be greater than 1.0, or the policy will not be able to set to a state of PolicyState.ENABLED.

This class inherits fields from Policy.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier for the policy. Must contain only lowercase letters and dashes, such as 'auto-vm-migration'.
conflicted	**Boolean**	No	Signifies whether any other policies are currently activated that potentially conflict with this policy. If true, it signifies a **potential** conflict.
description	**String**	No	The user friendly description of the policy.
memoryAllocationLimit	**Double**	Yes	Setting this to 1 effectively disables the allocation overcommit based on memory. If this and processorAllocationLimit are both set to 1.0 (the default), the policy state cannot be set to PolicyState.ENABLED.
name	**String**	No	The user friendly name of the policy.
potentialConflicts	**Set<String>**	No	A set of policy IDs that may potentially conflict with this policy.
priority	**Integer**	No	Indicates the absolute priority of the policy with respect to others. It is possible that more than one policy has the same priority. The higher the number, the greater the priority. Minimum is 0.

Field Name	Type	PUT	Description
processorAllocationLimit	**Double**	Yes	The Allocation Limit defines the upper bound or maximum amount of VCPUs that can be created on any given hypervisor (HV). For example, if you have a hypervisor with 12 processors or cores (Moab sees them as 12 processors), and have an Allocation Limit of 2.0 for procs, then Moab will not allow, under any condition, more than 24 VCPU's to be allocated on this hypervisor. Remember: a VM can have one or more VCPU's. So, in this example, the HV could only support 8 VM's if they all had 3 VPCU's each. It could support 4 VM's if they had 6 VPCU's each, and so forth From http://www.adaptivecomputing.com/resources/docs/mwm/7-1-1/Content/topics/vm/allocation_limits_and_utilization_threshold.html Setting this to 1 effectively disables the allocation overcommit based on processors. If this and memoryAllocationLimit are both set to 1.0 (the default), the policy state cannot be set to PolicyState.ENABLED.
state	**PolicyState**	Yes	Defines the current state of the policy: enabled or disabled. Defaults to PolicyState.DISABLED.
tags	**Set<String>**	No	A set of strings that can be used to aid in filtering or querying policies.
types	**Set<String>**	No	A set of categories or types that the policy is included in. This may be used to filter or query on groups of policies.

PolicyState

Represents the state of a policy. A policy may only be enabled or disabled.

Value	Description
ENABLED	The policy is enabled or active.
DISABLED	The policy is disabled or inactive.

NodeAllocationPolicy

Node allocation is the process of selecting the best resources to allocate to a job from a list of available resources. Moab contains a number of allocation algorithms that address this in the NodeAllocationPolicy.

This class inherits fields from Policy.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier for the policy. Must contain only lowercase letters and dashes, such as 'auto-vm-migration'.
conflicted	**Boolean**	No	Signifies whether any other policies are currently activated that potentially conflict with this policy. If true, it signifies a **potential** conflict.
customPriorityFunction	**String**	Yes	Defines the priority function when the CustomPriority algorithm is used.
description	**String**	No	The user friendly description of the policy.
name	**String**	No	The user friendly name of the policy.
nodeAllocationAlgorithm	**NodeAllocationAlgorithm**	Yes	Configures the node allocation algorithm utilized when the policy is active. Defaults to NONE. When ENABLED, this must **not** be set to NONE.
potentialConflicts	**Set<String>**	No	A set of policy IDs that may potentially conflict with this policy.
priority	**Integer**	No	Indicates the absolute priority of the policy with respect to others. It is possible that more than one policy has the same priority. The higher the number, the greater the priority. Minimum is 0.

Field Name	Type	PUT	Description
state	**PolicyState**	Yes	Defines the current state of the policy: enabled or disabled. Defaults to PolicyState.DISABLED.
tags	**Set<String>**	No	A set of strings that can be used to aid in filtering or querying policies.
types	**Set<String>**	No	A set of categories or types that the policy is included in. This may be used to filter or query on groups of policies.

NodeAllocationAlgorithm

Represents the algorithm used to allocate Nodes when the NodeAllocationPolicy is used.

Value	Description
NONE	
InReportedOrder	
InReverseReportedOrder	
CustomPriority	
ProcessorLoad	
MinimumConfiguredResources	
Contiguous	
ProcessorSpeedBalance	
NodeSpeed	

PolicyState

Represents the state of a policy. A policy may only be enabled or disabled.

Value	Description
ENABLED	The policy is enabled or active.
DISABLED	The policy is disabled or inactive.

MigrationExclusionListPolicy

Specify which virtual machines and hypervisors to exclude from automatic migration operations.

This class inherits fields from Policy.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier for the policy. Must contain only lowercase letters and dashes, such as 'auto-vm-migration'.
conflicted	**Boolean**	No	Signifies whether any other policies are currently activated that potentially conflict with this policy. If true, it signifies a **potential** conflict.
description	**String**	No	The user friendly description of the policy.
hvExclusionList	**List<String>**	Yes	The list of hypervisor IDs on the exclusion list.
name	**String**	No	The user friendly name of the policy.
potentialConflicts	**Set<String>**	No	A set of policy IDs that may potentially conflict with this policy.
priority	**Integer**	No	Indicates the absolute priority of the policy with respect to others. It is possible that more than one policy has the same priority. The higher the number, the greater the priority. Minimum is 0.
state	**PolicyState**	Yes	Defines the current state of the policy: enabled or disabled. Defaults to PolicyState.DISABLED.
tags	**Set<String>**	No	A set of strings that can be used to aid in filtering or querying policies.

Field Name	Type	PUT	Description
types	**Set<String>**	No	A set of categories or types that the policy is included in. This may be used to filter or query on groups of policies.
vmExclusionList	**List<String>**	Yes	The list of VM IDs on the exclusion list.

PolicyState

Represents the state of a policy. A policy may only be enabled or disabled.

Value	Description
ENABLED	The policy is enabled or active.
DISABLED	The policy is disabled or inactive.

FairsharePolicy

Fairshare allows historical resource utilization information to be incorporated into job feasibility and priority decisions. This feature allows site administrators to set system utilization targets for users, groups, accounts, classes, and QoS levels. Administrators can also specify the time frame over which resource utilization is evaluated in determining whether the goal is being reached. Parameters allow sites to specify the utilization metric, how historical information is aggregated, and the effect of fairshare state on scheduling behavior. You can specify fairshare targets for any credentials (such as user, group, and class) that administrators want such information to affect. http://docs.adaptivecomputing.com/mwm/archive/6-0-4/6.3fairshare.php

This class inherits fields from Policy.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier for the policy. Must contain only lowercase letters and dashes, such as 'auto-vm-migration'.
conflicted	**Boolean**	No	Signifies whether any other policies are currently activated that potentially conflict with this policy. If true, it signifies a **potential** conflict.

Field Name	Type	PUT	Description
decayFactor	**Double**	Yes	Specifies decay rate applied to past fairshare interval when computing effective fairshare usage. Values may be in the range of 0.01 to 1.0. A smaller value causes more rapid decay causing aged usage to contribute less to the overall effective fairshare usage. A value of 1.0 indicates that no decay will occur and all fairshare intervals will be weighted equally when determining effective fairshare usage.
depth	**Integer**	Yes	Number of fairshare windows factored into current fairshare utilization. Note: The number of available fairshare windows is bounded by the MAX_FSDEPTH value (32 in Moab).
description	**String**	No	The user friendly description of the policy.
intervalSeconds	**Long**	Yes	Specifies the length of each fairshare window in seconds.
name	**String**	No	The user friendly name of the policy.
potentialConflicts	**Set<String>**	No	A set of policy IDs that may potentially conflict with this policy.
priority	**Integer**	No	Indicates the absolute priority of the policy with respect to others. It is possible that more than one policy has the same priority. The higher the number, the greater the priority. Minimum is 0.
state	**PolicyState**	Yes	Defines the current state of the policy: enabled or disabled. Defaults to PolicyState.DISABLED.
tags	**Set<String>**	No	A set of strings that can be used to aid in filtering or querying policies.
types	**Set<String>**	No	A set of categories or types that the policy is included in. This may be used to filter or query on groups of policies.

Field Name	Type	PUT	Description
usageMetric	FairshareUsageMetric	Yes	As Moab runs, it records how available resources are used. Each iteration it updates fairshare resource utilization statistics. Resource utilization is tracked in accordance with the usage metric allowing various aspects of resource consumption information to be measured. The usage metric allows selection of both the types of resources to be tracked as well as the method of tracking. It provides the option of tracking usage by dedicated or consumed resources, where dedicated usage tracks what the scheduler assigns to the job and consumed usage tracks what the job actually uses.

PolicyState

Represents the state of a policy. A policy may only be enabled or disabled.

Value	Description
ENABLED	The policy is enabled or active.
DISABLED	The policy is disabled or inactive.

FairshareUsageMetric

Specifies the unit of tracking FairsharePolicy usage.
http://docs.adaptivecomputing.com/mwm/archive/6-0-4/6.3fairshare.php#fspolicy

Value	Description
NONE	
DEDICATED_ PROCESSOR_ SECONDS_DELIVERED	Usage tracked by processor seconds dedicated to each job relative to other processor seconds dedicated to other jobs on the system. (Useful in dedicated node environments.)
DEDICATED_ PROCESSOR_ SECONDS_AVAILABLE	Usage tracked by processor seconds dedicated to each job relative to all available processor seconds dedicated to other jobs on the system. (Useful in dedicated node environments.)

Value	Description
DEDICATED_ PROCESSOR_ EQUIVALENT_ SECONDS_DELIVERED	Usage tracked by processor-equivalent seconds dedicated to each job relative to other processor-equivalent seconds dedicated to other jobs on the system. (Useful in dedicated and shared nodes environments).
UTILIZED_ PROCESSOR_ SECONDS_DELIVERED	Usage tracked by processor seconds used by each job relative to other processor seconds used by other jobs on the system. (Useful in shared node/SMP environments.)

API version 2

Policy

A Moab Workload Manager policy which can affect scheduling decisions such as resource allocation. A policy contains state, identifying information, a priority, and metadata about the policy.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier for the policy. Must contain only lowercase letters and dashes, such as 'auto-vm-migration'.
conflicted	**Boolean**	No	Signifies whether any other policies are currently activated that potentially conflict with this policy. If true, it signifies a **potential** conflict.
description	**String**	No	The user friendly description of the policy.
name	**String**	No	The user friendly name of the policy.
potentialConflicts	**Set<String>**	No	A set of policy IDs that may potentially conflict with this policy.
priority	**Integer**	No	Indicates the absolute priority of the policy with respect to others. It is possible that more than one policy has the same priority. The higher the number, the greater the priority. Minimum is 0.
state	**PolicyState**	Yes	Defines the current state of the policy: enabled or disabled. Defaults to PolicyState.DISABLED.
tags	**Set<String>**	No	A set of strings that can be used to aid in filtering or querying policies.
types	**Set<String>**	No	A set of categories or types that the policy is included in. This may be used to filter or query on groups of policies.

PolicyState

Represents the state of a policy. A policy may only be enabled or disabled.

Value	Description
ENABLED	The policy is enabled or active.
DISABLED	The policy is disabled or inactive.

AutoVMMigrationPolicy

The Moab policy used to enabled and configure policy-based VM migration. Using information about data center applications and server load, Moab can aim to keep VMs in the data center optimally distributed across all hypervisors.

This class inherits fields from Policy.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier for the policy. Must contain only lowercase letters and dashes, such as 'auto-vm-migration'.
conflicted	**Boolean**	No	Signifies whether any other policies are currently activated that potentially conflict with this policy. If true, it signifies a **potential** conflict.
description	**String**	No	The user friendly description of the policy.
genericMetricThresholds	**Map<String, Double>**	Yes	A map of generic metric pairings where each value must be greater than or equal to 0 such as: `METRIC1 => 5.6 METRIC2 => 0.0 METRIC3 => 102.4`
memoryUtilizationThreshold	**Double**	Yes	Defines the utilization threshold for memory. This must be greater than 0 and less than or equal to 1. A value of 1 effectively disables the threshold.
migrationAlgorithmType	**AutoVMMigrationPolicyType**	Yes	Configures the VM migration algorithm utilized when the policy is active. Defaults to NONE. When ENABLED, this must **not** be set to NONE.

Field Name	Type	PUT	Description
name	**String**	No	The user friendly name of the policy.
potentialConflicts	**Set<String>**	No	A set of policy IDs that may potentially conflict with this policy.
priority	**Integer**	No	Indicates the absolute priority of the policy with respect to others. It is possible that more than one policy has the same priority. The higher the number, the greater the priority. Minimum is 0.
processorUtilizationThreshold	**Double**	Yes	Defines the load utilization threshold for processors. This must be greater than 0 and less than or equal to 1. A value of 1 effectively disables the threshold.
state	**PolicyState**	Yes	Defines the current state of the policy: enabled or disabled. Defaults to PolicyState.DISABLED.
tags	**Set<String>**	No	A set of strings that can be used to aid in filtering or querying policies.
types	**Set<String>**	No	A set of categories or types that the policy is included in. This may be used to filter or query on groups of policies.

AutoVMMigrationPolicyType

Represents the algorithm used to migrate VMs when the AutoVMMigrationPolicy is used.

Value	Description
NONE	Used when the Auto VM Migration policy is currently disabled in Moab and before any settings are saved the first time. For example, if the policy is disabled on the first read of Moab policies, the AutoVMMigrationPolicy.migrationAlgorithmType will be set to NONE. If the policy is enabled and the type is set to OVERCOMMIT, followed by a disabling of the policy, it will then be represented as having a state of DISABLED with a migrationAlgorithmType of OVERCOMMIT.
OVERCOMMIT	Use the "overcommit" algorithm for migration. The goal of this algorithm is to equalize loads across hypervisors as migrations are queued due to overcommit conditions. This places VMs to be migrated on the least-loaded hypervisor available.
CONSOLIDATION	Use the "consolidation" algorithm for migration. The goal of this algorithm is to load hypervisors as close to thresholds as possible, without exceeding them. This policy places VMs to be migrated on the most loaded hypervisor possible, within these constraints. A second loop of this policy will select lightly-loaded hypervisors to be evacuated completely.

PolicyState

Represents the state of a policy. A policy may only be enabled or disabled.

Value	Description
ENABLED	The policy is enabled or active.
DISABLED	The policy is disabled or inactive.

HVAllocationOvercommitPolicy

The Hypervisor Allocation Overcommit policy controls how many virtual machines can be placed on a hypervisor. By enabling this policy, you are allowing Moab to allocate more resoures to a set of virtual machines than a hypervisor may actually have. This is possible due to virtualization. In other words, this policy allows you to set the high-water mark for virtual machine allocation for hypervisors. At least one of these limits must be greater than 1.0, or the policy will not be able to set to a state of PolicyState.ENABLED.

This class inherits fields from Policy.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier for the policy. Must contain only lowercase letters and dashes, such as 'auto-vm-migration'.
conflicted	**Boolean**	No	Signifies whether any other policies are currently activated that potentially conflict with this policy. If true, it signifies a **potential** conflict.
description	**String**	No	The user friendly description of the policy.
memoryAllocationLimit	**Double**	Yes	Setting this to 1 effectively disables the allocation overcommit based on memory. If this and processorAllocationLimit are both set to 1.0 (the default), the policy state cannot be set to PolicyState.ENABLED.
name	**String**	No	The user friendly name of the policy.
potentialConflicts	**Set<String>**	No	A set of policy IDs that may potentially conflict with this policy.
priority	**Integer**	No	Indicates the absolute priority of the policy with respect to others. It is possible that more than one policy has the same priority. The higher the number, the greater the priority. Minimum is 0.

Field Name	Type	PUT	Description
processorAllocationLimit	**Double**	Yes	The Allocation Limit defines the upper bound or maximum amount of VCPUs that can be created on any given hypervisor (HV). For example, if you have a hypervisor with 12 processors or cores (Moab sees them as 12 processors), and have an Allocation Limit of 2.0 for procs, then Moab will not allow, under any condition, more than 24 VCPU's to be allocated on this hypervisor. Remember: a VM can have one or more VCPU's. So, in this example, the HV could only support 8 VM's if they all had 3 VPCU's each. It could support 4 VM's if they had 6 VPCU's each, and so forth From http://www.adaptivecomputing.com/resources/docs/mwm/7-1-1/Content/topics/vm/allocation_limits_and_utilization_threshold.html Setting this to 1 effectively disables the allocation overcommit based on processors. If this and memoryAllocationLimit are both set to 1.0 (the default), the policy state cannot be set to PolicyState.ENABLED.
state	**PolicyState**	Yes	Defines the current state of the policy: enabled or disabled. Defaults to PolicyState.DISABLED.
tags	**Set<String>**	No	A set of strings that can be used to aid in filtering or querying policies.
types	**Set<String>**	No	A set of categories or types that the policy is included in. This may be used to filter or query on groups of policies.

PolicyState

Represents the state of a policy. A policy may only be enabled or disabled.

Value	Description
ENABLED	The policy is enabled or active.
DISABLED	The policy is disabled or inactive.

NodeAllocationPolicy

Node allocation is the process of selecting the best resources to allocate to a job from a list of available resources. Moab contains a number of allocation algorithms that address this in the NodeAllocationPolicy.

This class inherits fields from Policy.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier for the policy. Must contain only lowercase letters and dashes, such as 'auto-vm-migration'.
conflicted	**Boolean**	No	Signifies whether any other policies are currently activated that potentially conflict with this policy. If true, it signifies a **potential** conflict.
customPriorityFunction	**String**	Yes	Defines the priority function when the CustomPriority algorithm is used.
description	**String**	No	The user friendly description of the policy.
name	**String**	No	The user friendly name of the policy.
nodeAllocationAlgorithm	**NodeAllocationAlgorithm**	Yes	Configures the node allocation algorithm utilized when the policy is active. Defaults to NONE. When ENABLED, this must **not** be set to NONE.
potentialConflicts	**Set<String>**	No	A set of policy IDs that may potentially conflict with this policy.
priority	**Integer**	No	Indicates the absolute priority of the policy with respect to others. It is possible that more than one policy has the same priority. The higher the number, the greater the priority. Minimum is 0.

Field Name	Type	PUT	Description
state	**PolicyState**	Yes	Defines the current state of the policy: enabled or disabled. Defaults to PolicyState.DISABLED.
tags	**Set<String>**	No	A set of strings that can be used to aid in filtering or querying policies.
types	**Set<String>**	No	A set of categories or types that the policy is included in. This may be used to filter or query on groups of policies.

NodeAllocationAlgorithm

Represents the algorithm used to allocate Nodes when the NodeAllocationPolicy is used.

Value	Description
NONE	
InReportedOrder	
InReverseReportedOrder	
CustomPriority	
ProcessorLoad	
MinimumConfiguredResources	
Contiguous	
ProcessorSpeedBalance	
NodeSpeed	

PolicyState

Represents the state of a policy. A policy may only be enabled or disabled.

Value	Description
ENABLED	The policy is enabled or active.
DISABLED	The policy is disabled or inactive.

MigrationExclusionListPolicy

Specify which virtual machines and hypervisors to exclude from automatic migration operations.

This class inherits fields from Policy.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier for the policy. Must contain only lowercase letters and dashes, such as 'auto-vm-migration'.
conflicted	**Boolean**	No	Signifies whether any other policies are currently activated that potentially conflict with this policy. If true, it signifies a **potential** conflict.
description	**String**	No	The user friendly description of the policy.
hvExclusionList	**List<String>**	Yes	The list of hypervisor IDs on the exclusion list.
name	**String**	No	The user friendly name of the policy.
potentialConflicts	**Set<String>**	No	A set of policy IDs that may potentially conflict with this policy.
priority	**Integer**	No	Indicates the absolute priority of the policy with respect to others. It is possible that more than one policy has the same priority. The higher the number, the greater the priority. Minimum is 0.
state	**PolicyState**	Yes	Defines the current state of the policy: enabled or disabled. Defaults to PolicyState.DISABLED.
tags	**Set<String>**	No	A set of strings that can be used to aid in filtering or querying policies.

Field Name	Type	PUT	Description
types	**Set<String>**	No	A set of categories or types that the policy is included in. This may be used to filter or query on groups of policies.
vmExclusionList	**List<String>**	Yes	The list of VM IDs on the exclusion list.

PolicyState

Represents the state of a policy. A policy may only be enabled or disabled.

Value	Description
ENABLED	The policy is enabled or active.
DISABLED	The policy is disabled or inactive.

FairsharePolicy

Fairshare allows historical resource utilization information to be incorporated into job feasibility and priority decisions. This feature allows site administrators to set system utilization targets for users, groups, accounts, classes, and QoS levels. Administrators can also specify the time frame over which resource utilization is evaluated in determining whether the goal is being reached. Parameters allow sites to specify the utilization metric, how historical information is aggregated, and the effect of fairshare state on scheduling behavior. You can specify fairshare targets for any credentials (such as user, group, and class) that administrators want such information to affect. http://docs.adaptivecomputing.com/mwm/archive/6-0-4/6.3fairshare.php

This class inherits fields from Policy.

Field Name	Type	PUT	Description
id	**String**	No	The unique identifier for the policy. Must contain only lowercase letters and dashes, such as 'auto-vm-migration'.
conflicted	**Boolean**	No	Signifies whether any other policies are currently activated that potentially conflict with this policy. If true, it signifies a **potential** conflict.

Field Name	Type	PUT	Description
decayFactor	**Double**	Yes	Specifies decay rate applied to past fairshare interval when computing effective fairshare usage. Values may be in the range of 0.01 to 1.0. A smaller value causes more rapid decay causing aged usage to contribute less to the overall effective fairshare usage. A value of 1.0 indicates that no decay will occur and all fairshare intervals will be weighted equally when determining effective fairshare usage.
depth	**Integer**	Yes	Number of fairshare windows factored into current fairshare utilization. Note: The number of available fairshare windows is bounded by the MAX_FSDEPTH value (32 in Moab).
description	**String**	No	The user friendly description of the policy.
intervalSeconds	**Long**	Yes	Specifies the length of each fairshare window in seconds.
name	**String**	No	The user friendly name of the policy.
potentialConflicts	**Set<String>**	No	A set of policy IDs that may potentially conflict with this policy.
priority	**Integer**	No	Indicates the absolute priority of the policy with respect to others. It is possible that more than one policy has the same priority. The higher the number, the greater the priority. Minimum is 0.
state	**PolicyState**	Yes	Defines the current state of the policy: enabled or disabled. Defaults to PolicyState.DISABLED.
tags	**Set<String>**	No	A set of strings that can be used to aid in filtering or querying policies.
types	**Set<String>**	No	A set of categories or types that the policy is included in. This may be used to filter or query on groups of policies.

Field Name	Type	PUT	Description
usageMetric	FairshareUsageMetric	Yes	As Moab runs, it records how available resources are used. Each iteration it updates fairshare resource utilization statistics. Resource utilization is tracked in accordance with the usage metric allowing various aspects of resource consumption information to be measured. The usage metric allows selection of both the types of resources to be tracked as well as the method of tracking. It provides the option of tracking usage by dedicated or consumed resources, where dedicated usage tracks what the scheduler assigns to the job and consumed usage tracks what the job actually uses.

PolicyState

Represents the state of a policy. A policy may only be enabled or disabled.

Value	Description
ENABLED	The policy is enabled or active.
DISABLED	The policy is disabled or inactive.

FairshareUsageMetric

Specifies the unit of tracking FairsharePolicy usage.
http://docs.adaptivecomputing.com/mwm/archive/6-0-4/6.3fairshare.php#fspolicy

Value	Description
NONE	
DEDICATED_ PROCESSOR_ SECONDS_DELIVERED	Usage tracked by processor seconds dedicated to each job relative to other processor seconds dedicated to other jobs on the system. (Useful in dedicated node environments.)
DEDICATED_ PROCESSOR_ SECONDS_AVAILABLE	Usage tracked by processor seconds dedicated to each job relative to all available processor seconds dedicated to other jobs on the system. (Useful in dedicated node environments.)

Value	Description
DEDICATED_ PROCESSOR_ EQUIVALENT_ SECONDS_DELIVERED	Usage tracked by processor-equivalent seconds dedicated to each job relative to other processor-equivalent seconds dedicated to other jobs on the system. (Useful in dedicated and shared nodes environments).
UTILIZED_ PROCESSOR_ SECONDS_DELIVERED	Usage tracked by processor seconds used by each job relative to other processor seconds used by other jobs on the system. (Useful in shared node/SMP environments.)

Related topics

- **Policies** on page 1589

Fields: Principals

See the associated **Principals** on page 1605 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`principals`	**Permissions** on page 1571
Hooks filename	`principals.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	Yes	**Distinct** on page 1504

API version 3

Principal

A principal maps to a set of ldap users, ldap groups, pam users, and/or pam groups. MWS roles are attached to the principals to authorize the group to use the specific MWS roles.

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of this principal.
attachedRoles	**Set<Role>**	Yes	Yes	The MWS roles this principal is authorized to use.
description	**String**	Yes	Yes	The principal description.
groups	**List<Map>**	Yes	Yes	The groups associated with this principal. Each group has a name and a type. The valid types of groups are LDAPOU, LDAPGROUP, PAMGROUP, and SSO. Example group: `{"name": "CN=Engineering,CN=Users,DC=corp,DC=cloud,DC=dev", "type": "LDAPGROUP"}` or `{"name": "engineering", "type": "PAMGROUP"}`
name	**String**	Yes	Yes	The unique human-readable name of this principal. Required during POST.
users	**List<Map>**	Yes	Yes	The users associated with this principal. Each user has a name and type. The valid types of users are LDAP and PAM. Example user: `{"name": "jhammon", "type": "LDAP"}` or `{"name": "jhammon", "type": "PAM"}`

Role

A role defines a set of permissions that are based on the proxy-user. If no proxy user is specified then access to objects in MWS are limited to its application permissions. For example if the application has permission to update all resources in MWS and no proxy-user is specified in the request then the request can access all resources in MWS.

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of this role.

Field Name	Type	POST	PUT	Description
description	**String**	Yes	Yes	The role description.
name	**String**	Yes	Yes	The unique human-readable name of this role. Required during POST.
permissions	**List<Permission>**	Yes	Yes	The set of permissions enforced based on the proxy-user.
scope	**PrivilegeScope**	No	No	

Permission

Represents a permission

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of this role.
action	**String**	No	No	The action that can be performed on the resource.
description	**String**	No	No	A description of this permission.
fieldPath	**String**	No	No	Field level ACL control, if null or '*', all fields are accessible, otherwise requests must match dot delimited path. Currently only checked when doing writable actions. Example - attributes.*: create|update
label	**String**	No	No	A human readable label for this permission.
resource	**String**	No	No	The resource the permission applies to.
resourceFilter	**Map<String, Map>**	No	No	A map used to limit which resource instances this permission applies to. If this is null then the permission will apply to all instances of the resource. For api permissions the filter uses mongo query syntax.

Field Name	Type	POST	PUT	Description
scope	**PrivilegeScope**	No	No	Whether this permission applies to the principal's tenant-associated resources or globally
type	**String**	No	No	The type of the permission. Only 'api' type permissions are enforced.

PrivilegeScope

Some permissions and roles ignore tenants and apply globally. Others apply only to the resources associated with the principal's tenants.

Value	Description
GLOBAL	Describes a role or permission that applies globally, irrespective of the principal's tenants. This scope can be applied to any role or permission.
TENANT	Describes a role or permission that applies only to the resources associated with the principal's tenants. This scope can be applied to any role, but only to those permissions associated with tenanted resources (e.g. nodes, services, etc.).
NONE	Scope doesn't apply to some permissions. As of right now, all non-domain permissions (e.g. those created by Viewpoint) don't need a scope. NONE should therefore be assigned to all non-domain permissions.

PrivilegeScope

Some permissions and roles ignore tenants and apply globally. Others apply only to the resources associated with the principal's tenants.

Value	Description
GLOBAL	Describes a role or permission that applies globally, irrespective of the principal's tenants. This scope can be applied to any role or permission.
TENANT	Describes a role or permission that applies only to the resources associated with the principal's tenants. This scope can be applied to any role, but only to those permissions associated with tenanted resources (e.g. nodes, services, etc.).
NONE	Scope doesn't apply to some permissions. As of right now, all non-domain permissions (e.g. those created by Viewpoint) don't need a scope. NONE should therefore be assigned to all non-domain permissions.

API version 2

Principal

A principal maps to a set of ldap users, ldap groups, pam users, and/or pam groups. MWS roles are attached to the principals to authorize the group to use the specific MWS roles.

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of this principal.
attachedRoles	**Set<Role>**	Yes	Yes	The MWS roles this principal is authorized to use.
description	**String**	Yes	Yes	The principal description.
groups	**List<Map>**	Yes	Yes	The groups associated with this principal. Each group has a name and a type. The valid types of groups are LDAPOU, LDAPGROUP, PAMGROUP, and SSO. Example group: `{"name": "CN=Engineering,CN=Users,DC=corp,DC=cloud,DC=dev", "type": "LDAPGROUP"}` or `{"name": "engineering", "type": "PAMGROUP"}`
name	**String**	Yes	Yes	The unique human-readable name of this principal. Required during POST.
users	**List<Map>**	Yes	Yes	The users associated with this principal. Each user has a name and type. The valid types of users are LDAP and PAM. Example user: `{"name": "jhammon", "type": "LDAP"}` or `{"name": "jhammon", "type": "PAM"}`

Role

A role defines a set of permissions that are based on the proxy-user. If no proxy user is specified then access to objects in MWS are limited to its application permissions. For example if the application has permission to update all resources in MWS and no proxy-user is specified in the request then the request can access all resources in MWS.

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of this role.

Field Name	Type	POST	PUT	Description
description	**String**	Yes	Yes	The role description.
name	**String**	Yes	Yes	The unique human-readable name of this role. Required during POST.
permissions	**List<Permission>**	Yes	Yes	The set of permissions enforced based on the proxy-user.
scope	**PrivilegeScope**	No	No	

Permission

Represents a permission

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of this role.
action	**String**	No	No	The action that can be performed on the resource.
description	**String**	No	No	A description of this permission.
fieldPath	**String**	No	No	Field level ACL control, if null or '*', all fields are accessible, otherwise requests must match dot delimited path. Currently only checked when doing writable actions. Example - attributes.*: create|update
label	**String**	No	No	A human readable label for this permission.
resource	**String**	No	No	The resource the permission applies to.
resourceFilter	**Map<String, Map>**	No	No	A map used to limit which resource instances this permission applies to. If this is null then the permission will apply to all instances of the resource. For api permissions the filter uses mongo query syntax.

Field Name	Type	POST	PUT	Description
scope	**PrivilegeScope**	No	No	Whether this permission applies to the principal's tenant-associated resources or globally
type	**String**	No	No	The type of the permission. Only 'api' type permissions are enforced.

PrivilegeScope

Some permissions and roles ignore tenants and apply globally. Others apply only to the resources associated with the principal's tenants.

Value	Description
GLOBAL	Describes a role or permission that applies globally, irrespective of the principal's tenants. This scope can be applied to any role or permission.
TENANT	Describes a role or permission that applies only to the resources associated with the principal's tenants. This scope can be applied to any role, but only to those permissions associated with tenanted resources (e.g. nodes, services, etc.).
NONE	Scope doesn't apply to some permissions. As of right now, all non-domain permissions (e.g. those created by Viewpoint) don't need a scope. NONE should therefore be assigned to all non-domain permissions.

PrivilegeScope

Some permissions and roles ignore tenants and apply globally. Others apply only to the resources associated with the principal's tenants.

Value	Description
GLOBAL	Describes a role or permission that applies globally, irrespective of the principal's tenants. This scope can be applied to any role or permission.
TENANT	Describes a role or permission that applies only to the resources associated with the principal's tenants. This scope can be applied to any role, but only to those permissions associated with tenanted resources (e.g. nodes, services, etc.).
NONE	Scope doesn't apply to some permissions. As of right now, all non-domain permissions (e.g. those created by Viewpoint) don't need a scope. NONE should therefore be assigned to all non-domain permissions.

Related topics

- **Principals** on page 1605

Fields: Report Datapoints

See the associated **Reports** on page 1614 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`reports/datapoints`	**Permissions** on page 1571
Hooks filename	`reports.datapoints.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	Yes	**Distinct** on page 1504

API version 3

Datapoint

A metric that measures system state over a specified period of time. For example, a datapoint may contain data on CPU utilization by specific users. A datapoint is generated by the consolidation of zero or more Samples. It could be said that a datapoint represents a smoothing of samples.

Field Name	Type	Description
id	**Long**	
data	**Map<String, Map>**	The actual consolidated sample data. This property may be 'null' if the Report.minimumSampleSize was not met when consolidating the datapoint.
endDate	**Date**	The ending date that the datapoint covers.
firstSampleDate	**Date**	The date of the first sample consolidated in this datapoint. (See also: Sample.timestamp.)
lastSampleDate	**Date**	The date of the last sample consolidated in this datapoint. (See also: Sample.timestamp.)
startDate	**Date**	The beginning date that the datapoint covers.

API version 2

Datapoint

A metric that measures system state over a specified period of time. For example, a datapoint may contain data on CPU utilization by specific users. A datapoint is generated by the consolidation of zero or more Samples. It could be said that a datapoint represents a smoothing of samples.

Field Name	Type	Description
id	**Long**	
data	**Map<String, Map>**	The actual consolidated sample data. This property may be 'null' if the Report.minimumSampleSize was not met when consolidating the datapoint.
endDate	**Date**	The ending date that the datapoint covers.
firstSampleDate	**Date**	The date of the first sample consolidated in this datapoint. (See also: Sample.timestamp.)
lastSampleDate	**Date**	The date of the last sample consolidated in this datapoint. (See also: Sample.timestamp.)
startDate	**Date**	The beginning date that the datapoint covers.

Related topics

- **Reports** on page 1614

Fields: Reports

See the associated **Reports** on page 1614 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`reports`	**Permissions** on page 1571
Hooks filename	`reports.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	Yes	**Distinct** on page 1504

API version 3

Report

A set of time-based values that share similar context. For example, a report may contain data on CPU or power utilization for all nodes in a cluster.

A report is composed of metadata and a collection of Datapoints. Samples are also associated with reports, but these are consolidated using the Report.consolidationFunction to create Datapoints.

If the datapoint documents are being truncated in any way or there are warnings about documents being too large, it may be necessary to increase the Report.reportDocumentSize.

Field Name	Type	POST	Description
id	**String**	No	The unique identifier for the report. This is automatically assigned and will be ignored if specified duration creation.
consolidationFunction	**String**	Yes	The consolidation function is the process used to convert a set of samples into a datapoint. Currently the only supported function is "average", which is used if none is specified.
datapointDuration	**Long**	Yes	Required. How long the datapoints are, in seconds.
datapoints	**List<Datapoint>**	Yes	This is the set of datapoints that have been consolidated for the report or are desired to be included in the report during creation time. In the latter case, these represent historical data created outside of the reporting framework. Only present when getting a single report.
description	**String**	Yes	A description of the report.
keepSamples	**Boolean**	Yes	Controls if samples are retained after consolidation. Defaults to false, which means that after consolidation, samples are discarded.
minimumSampleSize	**Integer**	Yes	If number of samples is below this number, the datapoint data field is "null". Defaults to 1.
name	**String**	Yes	Required. A unique name identifying the report. Valid characters are all alphanumeric characters, dashes (-), periods (.), and underscores (_).

Field Name	Type	POST	Description
reportDocumentSize	**Long**	Yes	The maximum size in bytes of each datapoint document stored for this report. This option is provided to maximize the amount of disk space used for a single report. The default value for this option is 100*1024, or 100 KB. The maximum value of this option is 16*1024*1024 (16777216) or 16 MB, which represents the maximum document size in MongoDB. See also http://www.mongodb.org/display/DOCS/Documents. Keep in mind that when creating a new report, MongoDB will initialize all needed space for all possible datapoint documents up front. This can easily fill a disk unless this parameter is modified.
reportSize	**Long**	Yes	Required. The size of the report in datapoints. After this number of datapoints is reached, the old datapoints will be discarded. WARNING: On report creation, a Mongo collection will be initialized that is the maximum size of a single entry (currently 16 MB) multiplied by the report size. Be careful in setting a large report size as this will quickly allocate the entire disk if many reports with large report sizes are created.

Datapoint

A metric that measures system state over a specified period of time. For example, a datapoint may contain data on CPU utilization by specific users. A datapoint is generated by the consolidation of zero or more Samples. It could be said that a datapoint represents a smoothing of samples.

Field Name	Type	POST	Description
id	**Long**	No	
data	**Map<String, Map>**	No	The actual consolidated sample data. This property may be 'null' if the Report.minimumSampleSize was not met when consolidating the datapoint.
endDate	**Date**	No	The ending date that the datapoint covers.

Field Name	Type	POST	Description
firstSampleDate	**Date**	No	The date of the first sample consolidated in this datapoint. (See also: Sample.timestamp.)
lastSampleDate	**Date**	No	The date of the last sample consolidated in this datapoint. (See also: Sample.timestamp.)
startDate	**Date**	No	The beginning date that the datapoint covers.

API version 2

Report

A set of time-based values that share similar context. For example, a report may contain data on CPU or power utilization for all nodes in a cluster.

A report is composed of metadata and a collection of Datapoints. Samples are also associated with reports, but these are consolidated using the Report.consolidationFunction to create Datapoints.

If the datapoint documents are being truncated in any way or there are warnings about documents being too large, it may be necessary to increase the Report.reportDocumentSize.

Field Name	Type	POST	Description
id	**String**	No	The unique identifier for the report. This is automatically assigned and will be ignored if specified duration creation.
consolidationFunction	**String**	Yes	The consolidation function is the process used to convert a set of samples into a datapoint. Currently the only supported function is "average", which is used if none is specified.
datapointDuration	**Long**	Yes	Required. How long the datapoints are, in seconds.
datapoints	**List<Datapoint>**	Yes	This is the set of datapoints that have been consolidated for the report or are desired to be included in the report during creation time. In the latter case, these represent historical data created outside of the reporting framework. Only present when getting a single report.
description	**String**	Yes	A description of the report.
keepSamples	**Boolean**	Yes	Controls if samples are retained after consolidation. Defaults to false, which means that after consolidation, samples are discarded.
minimumSampleSize	**Integer**	Yes	If number of samples is below this number, the datapoint data field is "null". Defaults to 1.
name	**String**	Yes	Required. A unique name identifying the report. Valid characters are all alphanumeric characters, dashes (-), periods (.), and underscores (_).

Field Name	Type	POST	Description
reportDocumentSize	**Long**	Yes	The maximum size in bytes of each datapoint document stored for this report. This option is provided to maximize the amount of disk space used for a single report. The default value for this option is 100*1024, or 100 KB. The maximum value of this option is 16*1024*1024 (16777216) or 16 MB, which represents the maximum document size in MongoDB. See also http://www.mongodb.org/display/DOCS/Documents. Keep in mind that when creating a new report, MongoDB will initialize all needed space for all possible datapoint documents up front. This can easily fill a disk unless this parameter is modified.
reportSize	**Long**	Yes	Required. The size of the report in datapoints. After this number of datapoints is reached, the old datapoints will be discarded. WARNING: On report creation, a Mongo collection will be initialized that is the maximum size of a single entry (currently 16 MB) multiplied by the report size. Be careful in setting a large report size as this will quickly allocate the entire disk if many reports with large report sizes are created.

Datapoint

A metric that measures system state over a specified period of time. For example, a datapoint may contain data on CPU utilization by specific users. A datapoint is generated by the consolidation of zero or more Samples. It could be said that a datapoint represents a smoothing of samples.

Field Name	Type	POST	Description
id	**Long**	No	
data	**Map<String, Map>**	No	The actual consolidated sample data. This property may be 'null' if the Report.minimumSampleSize was not met when consolidating the datapoint.
endDate	**Date**	No	The ending date that the datapoint covers.

Field Name	Type	POST	Description
firstSampleDate	**Date**	No	The date of the first sample consolidated in this datapoint. (See also: Sample.timestamp.)
lastSampleDate	**Date**	No	The date of the last sample consolidated in this datapoint. (See also: Sample.timestamp.)
startDate	**Date**	No	The beginning date that the datapoint covers.

Related topics

- **Reports** on page 1614

Fields: Reservations

See the associated **Reservations** on page 1624 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`reservations`	**Permissions** on page 1571
Hooks filename	`reservations.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

Reservation

A reservation is the mechanism by which Moab guarantees the availability of a set of resources at a particular time. Each reservation consists of three major components: (1) a set of resources, (2) a time frame, and (3) an access control list. It is a scheduler role to ensure that the access control list is not violated during the reservation's lifetime (that is, its time frame) on the resources listed. For example, a reservation may specify that node002 is reserved for user Tom on Friday. The scheduler is thus constrained to make certain that only Tom's jobs can use node002 at any time on Friday.

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of the reservation.
accountingAccount	**String**	Yes	No	Accountable Account.
accountingGroup	**String**	Yes	No	Accountable Group.
accountingQOS	**String**	Yes	No	Accountable QOS.
accountingUser	**String**	Yes	No	Accountable User.
aclRules	**Set<AclRule>**	Yes	No	The set of access control rules associated with this reservation.
allocatedNodeCount	**Integer**	No	No	The number of allocated nodes for this reservation.
allocatedNodes	**Set<DomainProxyVersion1>**	No	No	The nodes allocated to the reservation.
allocatedProcessorCount	**Integer**	No	No	The number of allocated processors.
allocatedTaskCount	**Integer**	No	No	The number of allocated tasks.
comments	**String**	Yes	No	Reservation's comments or description.

Field Name	Type	POST	PUT	Description
creationDate	**Date**	No	No	Creation date. Automatically set by Moab when a user creates the reservation.
duration	**Long**	Yes	No	The duration of the reservation (in seconds).
endDate	**Date**	Yes	No	The end date of the reservation. This is especially useful for one-time reservations, which have an exact time for when a reservation ends.
excludeJobs	**Set<String>**	Yes	No	The list of jobs to exclude. Client must also set the IGNJOBRSV reservation flag. Otherwise, results are undefined. Used only during reservation creation.
expireDate	**Date**	No	No	The date/time when the reservation expires and vacates.
flags	**Set<ReservationFlag>**	Yes	No	The flags associated with the reservation.
globalId	**String**	No	No	Global reservation ID.

Field Name	Type	POST	PUT	Description
hostListExpression	**String**	Yes	No	The list of nodes a user can select to reserve. This may or may not be the nodes that are currently allocated to this reservation. Note: Either hostListExpression or taskCount must be set to create a reservation.
idPrefix	**String**	Yes	No	The user-specified prefix for this reservation. If provided, Moab combines the idPrefix with an integer, and the combination is the unique identifier for this reservation.
isActive	**Boolean**	No	No	State whether or not this reservation is currently active.
isTracked	**Boolean**	No	No	States whether reservation resource usage is tracked.
label	**String**	Yes	No	When a label is assigned to a reservation, the reservation can then be referenced by that label as well as by the reservation name.
maxTasks	**Integer**	No	No	The maximum number of tasks for this reservation.
messages	**Set<MessageVersion1>**	No	No	Messages for the reservation.

Field Name	Type	POST	PUT	Description
owner	[EmbeddedCredential]	Yes	No	The owner of the reservation
partitionId	**String**	Yes	No	The ID of the partition this reservation is for.
profile	**String**	Yes	No	The profile that this reservation is using. A profile is a specification of attributes that all reservations share. Used only during reservation creation.
requirements	[ReservationRequirement]	Yes	No	The reservation's requirements.
reservationGroup	**String**	Yes	No	The reservation group to which the reservation belongs.
resources	**Map<String, Integer>**	Yes	No	The reservation's resources. This field is a map, where the key is PROCS, MEM DISK, SWAP, or one or more user-defined keys.
startDate	**Date**	Yes	No	The start time for the reservation. This is especially useful for one-time reservations, which have an exact time for when a reservation starts.
statistics	[ReservationStatistics]	No	No	The reservation's statistical information.
subType	**String**	Yes	No	The reservation sub-type.

Field Name	Type	POST	PUT	Description
taskCount	**Integer**	No	No	The number of tasks that must be allocated to satisfy the reservation request. Note: Either hostListExpression or taskCount must be set to create a reservation.
trigger	**Trigger**	Yes	No	Trigger for reservation. Used only during reservation creation.
triggerIds	**Set<String>**	No	No	The IDs of the triggers attached to this reservation.
uniqueIndex	**String**	No	No	The globally-unique reservation index.
variables	**Map<String, Map>**	Yes	Yes	The set of variables for this reservation.

AclRule

This class represents a rule that can be in Moab's access control list (ACL) mechanism.

The basic AclRule information is the object's name and type. The type directly maps to an AclType value. The default mechanism Moab uses to check the ACL for a particular item is if the user or object coming in has ANY of the values in the ACL, then the user or object is given access. If no values match the user or object in question, the user or object is rejected access.

Field Name	Type	POST	PUT	Description
affinity	**AclAffinity**	No	Yes	Reservation ACLs allow or deny access to reserved resources but they may also be configured to affect a job's affinity for a particular reservation. By default, jobs gravitate toward reservations through a mechanism known as positive affinity. This mechanism allows jobs to run on the most constrained resources leaving other, unreserved resources free for use by other jobs that may not be able to access the reserved resources. Normally this is a desired behavior. However, sometimes, it is desirable to reserve resources for use only as a last resort-using the reserved resources only when there are no other resources available. This last resort behavior is known as negative affinity. Defaults to AclAffinity.POSITIVE.
comparator	**ComparisonOperator**	No	Yes	The type of comparison to make against the ACL object. Defaults to ComparisonOperator.EQUAL.
type	**AclType**	No	Yes	The type of the object that is being granted (or denied) access.
value	**String**	No	Yes	The name of the object that is being granted (or denied) access.

AclAffinity

This enumeration describes the values available for describing how a rule is used in establishing access to an object in Moab. Currently, these ACL affinities are used only for granting access to reservations.

Value	Description
NEGATIVE	Access to the object is repelled using this rule until access is the last choice.

Value	Description
NEUTRAL	Access to the object is not affected by affinity.
POSITIVE	Access to the object is looked at as the first choice.
PREEMPTIBLE	Access to the object given the rule gives preemptible status to the accessor. Supported only during GET.
REQUIRED	The rule in question must be satisified in order to gain access to the object. Supported only during GET.
UNAVAILABLE	The rule does not have its affinity available. Supported only during GET.

ComparisonOperator

This enumeration is used when Moab needs to compare items. One such use is in Access Control Lists (ACLs).

Value	Description
GREATER_THAN	Valid values: ">", "gt"
GREATER_THAN_OR_EQUAL	Valid values: ">=", "ge"
LESS_THAN	Valid values: "<", "lt"
LESS_THAN_OR_EQUAL	Valid values: "<=", "le"
EQUAL	Valid values: "==", "eq", "="
NOT_EQUAL	Valid values: "!=", "ne", "<>"
LEXIGRAPHIC_SUBSTRING	Valid value: "%<"
LEXIGRAPHIC_NOT_EQUAL	Valid value: "%!"
LEXIGRAPHIC_EQUAL	Valid value: "%="

AclType

This enumeration describes the values available for the type of an ACL Rule.

Value	Description
USER	User
GROUP	Group
ACCOUNT	Account or Project
CLASS	Class or Queue
QOS	Quality of Service
CLUSTER	Cluster
JOB_ID	Job ID
RESERVATION_ID	Reservation ID
JOB_TEMPLATE	Job Template
JOB_ATTRIBUTE	Job Attribute
DURATION	Duration in Seconds
PROCESSOR_SECONDS	Processor Seconds
JPRIORITY	Not supported
MEMORY	Not supported
NODE	Not supported
PAR	Not supported
PROC	Not supported
QTIME	Not supported
QUEUE	Not supported

Value	Description
RACK	Not supported
SCHED	Not supported
SYTEM	Not supported
TASK	Not supported
VC	Not supported
XFACTOR	Not supported

DomainProxyVersion1

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The id of the object.

ReservationFlag

The flag types of a reservation.

Value	Description
ALLOWJOBOVERLAP	Allows jobs to overlap this Reservation, but not start during it (unless they have ACL access).
APPLYPROFRESOURCES	Only apply resource allocation info from profile.
DEADLINE	Reservation should be scheduled against a deadline.
IGNIDLEJOBS	Ignore idle job reservations.
IGNJOBRSV	Ignore job reservations, but not user or other reservations.
CHARGE	Charge the idle cycles in the accounting manager.

Value	Description
NOVMMIGRATIONS	Override the VM Migration Policy and don't migrate VMs that overlap this reservation.
OWNERPREEMPTIGNOREMINTIME	Owner ignores preemptmintime for this reservation.
PROVISION	Reservation should be capable of provisioning.
NOACLOVERLAP	Reservation will not look at ACLs to overlap job (when using exclusive).
ADVRES	If set, the reservation is created in advance of needing it.
ADVRESJOBDESTROY	Cancel any jobs associated with the reservation when it is released.
ALLOWGRID	The reservation is set up for use in a grid environment.
ALLOWPRSV	Personal reservations can be created within the space of this standing reservation (and ONLY this standing reservation). By default, when a standing reservation is given the flag ALLOWPRSV, it is given the ACL rule USER==ALL+ allowing all jobs and all users access.
BYNAME	Reservation only allows access to jobs that meet reservation ACLs and explicitly request the resources of this reservation using the job ADVRES flag.
DEDICATEDNODE	If set, only one active reservation is allowed on a node.
DEDICATEDRESOURCE	The reservation is only placed on resources that are not reserved by any other reservation, including jobs and other reservations.
EXCLUDEJOBS	Makes a reservation job exclusive, where only one job can run in the reservation.
ENDTRIGHASFIRED	A trigger has finished firing.
ENFORCENODESET	Enforce node sets when creating reservation.
EXCLUDEALLBUTSB	Reservation only shares resources with sandboxes.

Value	Description
EXCLUDEMYGROUP	Exclude reservations within the same group.
IGNRSV	Forces the reservation onto nodes regardless of whether there are other reservations currently residing on the nodes.
IGNSTATE	Request ignores existing resource reservations, allowing the reservation to be forced onto available resources even if this conflicts with other reservations.
ISACTIVE	If set, the reservation is currently active.
ISCLOSED	If set, the reservation is closed.
ISGLOBAL	If set the reservation applies to all resources.
OWNERPREEMPT	The owner of the reservation is given preemptor status for resources contained in the reservation.
PARENTLOCK	The reservation can only be destroyed by destroying its parent.
PREEMPTEE	The reservation is preemptible.
PLACEHOLDER	The reservation is a placeholder for resources.
PRSV	The reservation is a non-adminstrator, non-standing reservation, user-created reservation.
REQFULL	The reservation will fail if all resources requested cannot be allocated.
SCHEDULEVCRSV	The reservation was created as part of a schedule VC command. This pertains to reservations creating while scheduling MWS Services, and these are filtered from the MWS output of reservations.
SINGLEUSE	The reservation is automatically removed after completion of the first job to use the reserved resources.
SPACEFLEX	The reservation is allowed to adjust resources allocated over time in an attempt to optimize resource utilization.

Value	Description
STANDINGRSV	If set, the reservation was created by a standing reservation instance.
STATIC	Makes a reservation ineligible to modified or canceled by an administrator.
SYSTEMJOB	The reservation was created by a system job.
TIMEFLEX	The reservation is allowed to adjust the reserved time frame in an attempt to optimize resource utilization.
TRIGHASFIRED	The reservation has one or more triggers that have fired on it.
WASACTIVE	The reservation was previously active.
EVACVMS	Evacuate virtual machines on the node when the reservation starts.

MessageVersion1

Field Name	Type	POST	PUT	Description
author	**String**	No	No	The author of the message.
creationTime	**Date**	No	No	The time the message was created in epoch time.
expireTime	**Date**	No	No	The time the message will be deleted in epoch time.
index	**Integer**	No	No	The index of the message relative to other messages in Moab's memory.
message	**String**	No	Yes	The comment information itself.
messageCount	**Integer**	No	No	The number of times this message has been displayed.
priority	**Double**	No	No	An optional priority that can be attached to the comment.

EmbeddedCredential

Field Name	Type	POST	PUT	Description
name	**String**	No	No	
type	**CredentialType**	No	No	

CredentialType

Value	Description
USER	
GROUP	
ACCOUNT	
CLASS	
QOS	
NOT_SPECIFIED	

ReservationRequirement

Represents all the types of requirements a user can request while creating a reservation.

Field Name	Type	POST	PUT	Description
architecture	**String**	Yes	No	Required architecture.
featureList	**Set<String>**	Yes	No	The list of features required for this reservation.
featureMode	**String**	No	No	Required feature mode.
memory	**Integer**	Yes	No	Required node memory, in MB.
nodeCount	**Integer**	No	No	Required number of nodes.
nodeIds	**Set<String>**	No	No	The list of node IDs required for this reservation.

Field Name	Type	POST	PUT	Description
os	**String**	Yes	No	Required Operating System.
taskCount	**Integer**	Yes	No	Required task count.

ReservationStatistics

Represents some basic statistical information that is kept about the usage of reservations. All metrics that are kept track relate to processor-seconds usage.

Field Name	Type	POST	PUT	Description
caps	**Double**	No	No	The current active processor-seconds in the last reported iteration.
cips	**Double**	No	No	The current idle processor-seconds in the last reported iteration.
taps	**Double**	No	No	The total active processor-seconds over the life of the reservation.
tips	**Double**	No	No	The total idle processor-seconds over the life of the reservation.

Trigger

Field Name	Type	POST	PUT	Description
id	**String**	No	No	Trigger id - internal ID used by moab to track triggers
action	**String**	No	No	For exec atype triggers, signifies executable and arguments. For jobpreempt atype triggers, signifies PREEMPTPOLICY to apply to jobs that are running on allocated resources. For changeparam atype triggers, specifies the parameter to change and its new value (using the same syntax and behavior as the changeparam command).

Field Name	Type	POST	PUT	Description
actionType	[TriggerActionType]	No	No	
blockTime	**Date**	No	No	Time (in seconds) Moab will suspend normal operation to wait for trigger execution to finish. Use caution as Moab will completely stop normal operation until BlockTime expires.
description	**String**	No	No	
eventType	[TriggerEventType]	No	No	
expireTime	**Date**	No	No	Time at which trigger should be terminated if it has not already been activated.
failOffset	**Date**	No	No	Specifies the time (in seconds) that the threshold condition must exist before the trigger fires.
flags	[Set<TriggerFlag>]	No	No	
interval	**Boolean**	No	No	When used in conjunction with MultiFire and RearmTime trigger will fire at regular intervals. Can be used with [TriggerEventType.EPOCH] to create a Standing Trigger. Defaults to false
maxRetry	**Integer**	No	No	Specifies the number of times Action will be attempted before the trigger is designated a failure.
multiFire	**Boolean**	No	No	Specifies whether this trigger can fire multiple times. Defaults to false.
name	**String**	No	No	Trigger name - can be auto assigned by moab or requested. Alphanumeric up to 16 characters in length
objectId	**String**	No	No	The ID of the object which this is attached to.

Field Name	Type	POST	PUT	Description
objectType	**String**	No	No	The type of object which this is attached to. Possible values: • vm - Virtual Machine
offset	**Date**	No	No	Relative time offset from event when trigger can fire.
period	**TriggerPeriod**	No	No	Can be used in conjunction with Offset to have a trigger fire at the beginning of the specified period. Can be used with EType epoch to create a standing trigger.
rearmTime	**Date**	No	No	Time between MultiFire triggers; rearm time is enforced from the trigger event time.
requires	**String**	No	No	Variables this trigger requires to be set or not set before it will fire. Preceding the string with an exclamation mark (!) indicates this variable must NOT be set. Used in conjunction with Sets to create trigger dependencies.
sets	**String**	No	No	Variable values this trigger sets upon success or failure. Preceding the string with an exclamation mark (!) indicates this variable is set upon trigger failure. Preceding the string with a caret (^) indicates this variable is to be exported to the parent object when the current object is destroyed through a completion event. Used in conjunction with Requires to create trigger dependencies.

Field Name	Type	POST	PUT	Description
threshold	**String**	No	No	Reservation usage threshold - When reservation usage drops below Threshold, trigger will fire. Threshold usage support is only enabled for reservations and applies to percent processor utilization. gmetric thresholds are supported with job, node, credential, and reservation triggers. See Threshold Triggers in the Moab Workload Manager documentation for more information.
timeout	**Date**	No	No	Time allotted to this trigger before it is marked as unsuccessful and its process (if any) killed.
unsets	**String**	No	No	Variable this trigger destroys upon success or failure.

TriggerActionType

This enumeration specifies the action type of a trigger.

Value	Description
CANCEL	Only apply to reservation triggers
CHANGE_PARAM	
JOB_PREEMPT	This indicates that the trigger should preempt all jobs currently allocating resources assigned to the trigger's parent object. Only apply to reservation triggers.
MAIL	
THRESHOLD	
INTERNAL	
EXEC	

TriggerEventType

This enumeration specifies the event type of a trigger.

Value	Description
CANCEL	
CHECKPOINT	
CREATE	
END	
EPOCH	
FAIL	
HOLD	
MIGRATE	
MODIFY	
PREEMPT	
STANDING	
START	
THRESHOLD	

TriggerFlag

This enumeration specifies a flag belonging to a trigger.

Value	Description
ATTACH_ERROR	If the trigger outputs anything to stderr, Moab will attach this as a message to the trigger object.
CLEANUP	If the trigger is still running when the parent object completes or is canceled, the trigger will be killed.

Value	Description
CHECKPOINT	Moab should always checkpoint this trigger. See Checkpointing a Trigger in the Moab Workload Manager documentation for more information.
GLOBAL_VARS	The trigger will look in the name space of all nodes with the globalvars flag in addition to its own name space. A specific node to search can be specified using the following format: globalvars+node_id
INTERVAL	Trigger is periodic.
MULTIFIRE	Trigger can fire multiple times.
OBJECT_XML_STDIN	Trigger passes its parent's object XML information into the trigger's stdin. This only works for exec triggers with reservation type parents.
USER	The trigger will execute under the user ID of the object's owner. If the parent object is sched, the user to run under may be explicitly specified using the format user+<username>, for example flags=user+john:
GLOBAL_TRIGGER	The trigger will be (or was) inserted into the global trigger list.
ASYNCHRONOUS	An asynchronous trigger.
LEAVE_FILES	Do not remove stderr and stdout files.
PROBE	The trigger's stdout will be monitored.
PROBE_ALL	The trigger's stdout will be monitored.
GENERIC_SYSTEM_JOB	The trigger belongs to a generic system job (for checkpointing).
REMOVE_STD_FILES	The trigger will delete stdout/stderr files after it has been reset.
RESET_ON_MODIFY	The trigger resets if the object it is attached to is modified, even if multifire is not set.

Value	Description
SOFT_KILL	By default, a `SIGKILL` (kill -9) signal is sent to the kill the script when a trigger times out. This flag will instead send a `SIGTERM` (kill -15) signal to kill the script. The `SIGTERM` signal will allow the script to trap the signal so that the script can clean up any residual information on the system (instead of just dying, as with the `SIGKILL` signal). NOTE: A timed-out trigger will only receive one kill signal. This means that if you specify this flag, a timed-out trigger will only receive the `SIGTERM` signal, and never the `SIGKILL` signal.

TriggerPeriod

This enumeration specifies the period of a trigger.

Value	Description
MINUTE	
HOUR	
DAY	
WEEK	
MONTH	

API version 2

Reservation

A reservation is the mechanism by which Moab guarantees the availability of a set of resources at a particular time. Each reservation consists of three major components: (1) a set of resources, (2) a time frame, and (3) an access control list. It is a scheduler role to ensure that the access control list is not violated during the reservation's lifetime (that is, its time frame) on the resources listed. For example, a reservation may specify that node002 is reserved for user Tom on Friday. The scheduler is thus constrained to make certain that only Tom's jobs can use node002 at any time on Friday.

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of the reservation.
accountingAccount	**String**	Yes	No	Accountable Account.
accountingGroup	**String**	Yes	No	Accountable Group.
accountingQOS	**String**	Yes	No	Accountable QOS.
accountingUser	**String**	Yes	No	Accountable User.
aclRules	**Set<AclRule>**	Yes	No	The set of access control rules associated with this reservation.
allocatedNodeCount	**Integer**	No	No	The number of allocated nodes for this reservation.
allocatedNodes	**Set<DomainProxyVersion1>**	No	No	The nodes allocated to the reservation.
allocatedProcessorCount	**Integer**	No	No	The number of allocated processors.
allocatedTaskCount	**Integer**	No	No	The number of allocated tasks.
comments	**String**	Yes	No	Reservation's comments or description.

Field Name	Type	POST	PUT	Description
creationDate	**Date**	No	No	Creation date. Automatically set by Moab when a user creates the reservation.
duration	**Long**	Yes	No	The duration of the reservation (in seconds).
endDate	**Date**	Yes	No	The end date of the reservation. This is especially useful for one-time reservations, which have an exact time for when a reservation ends.
excludeJobs	**Set<String>**	Yes	No	The list of jobs to exclude. Client must also set the IGNJOBRSV reservation flag. Otherwise, results are undefined. Used only during reservation creation.
expireDate	**Date**	No	No	The date/time when the reservation expires and vacates.
flags	**Set<ReservationFlag>**	Yes	No	The flags associated with the reservation.
globalId	**String**	No	No	Global reservation ID.

Field Name	Type	POST	PUT	Description
hostListExpression	**String**	Yes	No	The list of nodes a user can select to reserve. This may or may not be the nodes that are currently allocated to this reservation. Note: Either hostListExpression or taskCount must be set to create a reservation.
idPrefix	**String**	Yes	No	The user-specified prefix for this reservation. If provided, Moab combines the idPrefix with an integer, and the combination is the unique identifier for this reservation.
isActive	**Boolean**	No	No	State whether or not this reservation is currently active.
isTracked	**Boolean**	No	No	States whether reservation resource usage is tracked.
label	**String**	Yes	No	When a label is assigned to a reservation, the reservation can then be referenced by that label as well as by the reservation name.
maxTasks	**Integer**	No	No	The maximum number of tasks for this reservation.
messages	**Set<MessageVersion1>**	No	No	Messages for the reservation.

Field Name	Type	POST	PUT	Description
owner	**EmbeddedCredential**	Yes	No	The owner of the reservation
partitionId	**String**	Yes	No	The ID of the partition this reservation is for.
profile	**String**	Yes	No	The profile that this reservation is using. A profile is a specification of attributes that all reservations share. Used only during reservation creation.
requirements	**ReservationRequirement**	Yes	No	The reservation's requirements.
reservationGroup	**String**	Yes	No	The reservation group to which the reservation belongs.
resources	**Map<String, Integer>**	Yes	No	The reservation's resources. This field is a map, where the key is PROCS, MEM DISK, SWAP, or one or more user-defined keys.
startDate	**Date**	Yes	No	The start time for the reservation. This is especially useful for one-time reservations, which have an exact time for when a reservation starts.
statistics	**ReservationStatistics**	No	No	The reservation's statistical information.
subType	**String**	Yes	No	The reservation sub-type.

Field Name	Type	POST	PUT	Description
taskCount	**Integer**	No	No	The number of tasks that must be allocated to satisfy the reservation request. Note: Either hostListExpression or taskCount must be set to create a reservation.
trigger	[**Trigger**]()	Yes	No	Trigger for reservation. Used only during reservation creation.
triggerIds	**Set<String>**	No	No	The IDs of the triggers attached to this reservation.
uniqueIndex	**String**	No	No	The globally-unique reservation index.
variables	**Map<String, Map>**	Yes	Yes	The set of variables for this reservation.

AclRule

This class represents a rule that can be in Moab's access control list (ACL) mechanism.

The basic AclRule information is the object's name and type. The type directly maps to an AclType value. The default mechanism Moab uses to check the ACL for a particular item is if the user or object coming in has ANY of the values in the ACL, then the user or object is given access. If no values match the user or object in question, the user or object is rejected access.

Field Name	Type	POST	PUT	Description
affinity	**AclAffinity**	No	Yes	Reservation ACLs allow or deny access to reserved resources but they may also be configured to affect a job's affinity for a particular reservation. By default, jobs gravitate toward reservations through a mechanism known as positive affinity. This mechanism allows jobs to run on the most constrained resources leaving other, unreserved resources free for use by other jobs that may not be able to access the reserved resources. Normally this is a desired behavior. However, sometimes, it is desirable to reserve resources for use only as a last resort-using the reserved resources only when there are no other resources available. This last resort behavior is known as negative affinity. Defaults to AclAffinity.POSITIVE.
comparator	**ComparisonOperator**	No	Yes	The type of comparison to make against the ACL object. Defaults to ComparisonOperator.EQUAL.
type	**AclType**	No	Yes	The type of the object that is being granted (or denied) access.
value	**String**	No	Yes	The name of the object that is being granted (or denied) access.

AclAffinity

This enumeration describes the values available for describing how a rule is used in establishing access to an object in Moab. Currently, these ACL affinities are used only for granting access to reservations.

Value	Description
NEGATIVE	Access to the object is repelled using this rule until access is the last choice.

Value	Description
NEUTRAL	Access to the object is not affected by affinity.
POSITIVE	Access to the object is looked at as the first choice.
PREEMPTIBLE	Access to the object given the rule gives preemptible status to the accessor. Supported only during GET.
REQUIRED	The rule in question must be satisified in order to gain access to the object. Supported only during GET.
UNAVAILABLE	The rule does not have its affinity available. Supported only during GET.

ComparisonOperator

This enumeration is used when Moab needs to compare items. One such use is in Access Control Lists (ACLs).

Value	Description
GREATER_THAN	Valid values: ">", "gt"
GREATER_THAN_OR_EQUAL	Valid values: ">=", "ge"
LESS_THAN	Valid values: "<", "lt"
LESS_THAN_OR_EQUAL	Valid values: "<=", "le"
EQUAL	Valid values: "==", "eq", "="
NOT_EQUAL	Valid values: "!=", "ne", "<>"
LEXIGRAPHIC_SUBSTRING	Valid value: "%<"
LEXIGRAPHIC_NOT_EQUAL	Valid value: "%!"
LEXIGRAPHIC_EQUAL	Valid value: "%="

AclType

This enumeration describes the values available for the type of an ACL Rule.

Value	Description
USER	User
GROUP	Group
ACCOUNT	Account or Project
CLASS	Class or Queue
QOS	Quality of Service
CLUSTER	Cluster
JOB_ID	Job ID
RESERVATION_ID	Reservation ID
JOB_TEMPLATE	Job Template
JOB_ATTRIBUTE	Job Attribute
DURATION	Duration in Seconds
PROCESSOR_SECONDS	Processor Seconds
JPRIORITY	Not supported
MEMORY	Not supported
NODE	Not supported
PAR	Not supported
PROC	Not supported
QTIME	Not supported
QUEUE	Not supported

Value	Description
RACK	Not supported
SCHED	Not supported
SYTEM	Not supported
TASK	Not supported
VC	Not supported
XFACTOR	Not supported

DomainProxyVersion1

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The id of the object.

ReservationFlag

The flag types of a reservation.

Value	Description
ALLOWJOBOVERLAP	Allows jobs to overlap this Reservation, but not start during it (unless they have ACL access).
APPLYPROFRESOURCES	Only apply resource allocation info from profile.
DEADLINE	Reservation should be scheduled against a deadline.
IGNIDLEJOBS	Ignore idle job reservations.
IGNJOBRSV	Ignore job reservations, but not user or other reservations.
CHARGE	Charge the idle cycles in the accounting manager.

Value	Description
NOVMMIGRATIONS	Override the VM Migration Policy and don't migrate VMs that overlap this reservation.
OWNERPREEMPTIGNOREMINTIME	Owner ignores preemptmintime for this reservation.
PROVISION	Reservation should be capable of provisioning.
NOACLOVERLAP	Reservation will not look at ACLs to overlap job (when using exclusive).
ADVRES	If set, the reservation is created in advance of needing it.
ADVRESJOBDESTROY	Cancel any jobs associated with the reservation when it is released.
ALLOWGRID	The reservation is set up for use in a grid environment.
ALLOWPRSV	Personal reservations can be created within the space of this standing reservation (and ONLY this standing reservation). By default, when a standing reservation is given the flag ALLOWPRSV, it is given the ACL rule USER==ALL+ allowing all jobs and all users access.
BYNAME	Reservation only allows access to jobs that meet reservation ACLs and explicitly request the resources of this reservation using the job ADVRES flag.
DEDICATEDNODE	If set, only one active reservation is allowed on a node.
DEDICATEDRESOURCE	The reservation is only placed on resources that are not reserved by any other reservation, including jobs and other reservations.
EXCLUDEJOBS	Makes a reservation job exclusive, where only one job can run in the reservation.
ENDTRIGHASFIRED	A trigger has finished firing.
ENFORCENODESET	Enforce node sets when creating reservation.
EXCLUDEALLBUTSB	Reservation only shares resources with sandboxes.

Value	Description
EXCLUDEMYGROUP	Exclude reservations within the same group.
IGNRSV	Forces the reservation onto nodes regardless of whether there are other reservations currently residing on the nodes.
IGNSTATE	Request ignores existing resource reservations, allowing the reservation to be forced onto available resources even if this conflicts with other reservations.
ISACTIVE	If set, the reservation is currently active.
ISCLOSED	If set, the reservation is closed.
ISGLOBAL	If set the reservation applies to all resources.
OWNERPREEMPT	The owner of the reservation is given preemptor status for resources contained in the reservation.
PARENTLOCK	The reservation can only be destroyed by destroying its parent.
PREEMPTEE	The reservation is preemptible.
PLACEHOLDER	The reservation is a placeholder for resources.
PRSV	The reservation is a non-adminstrator, non-standing reservation, user-created reservation.
REQFULL	The reservation will fail if all resources requested cannot be allocated.
SCHEDULEVCRSV	The reservation was created as part of a schedule VC command. This pertains to reservations creating while scheduling MWS Services, and these are filtered from the MWS output of reservations.
SINGLEUSE	The reservation is automatically removed after completion of the first job to use the reserved resources.
SPACEFLEX	The reservation is allowed to adjust resources allocated over time in an attempt to optimize resource utilization.

Value	Description
STANDINGRSV	If set, the reservation was created by a standing reservation instance.
STATIC	Makes a reservation ineligible to modified or canceled by an administrator.
SYSTEMJOB	The reservation was created by a system job.
TIMEFLEX	The reservation is allowed to adjust the reserved time frame in an attempt to optimize resource utilization.
TRIGHASFIRED	The reservation has one or more triggers that have fired on it.
WASACTIVE	The reservation was previously active.
EVACVMS	Evacuate virtual machines on the node when the reservation starts.

MessageVersion1

Field Name	Type	POST	PUT	Description
author	**String**	No	No	The author of the message.
creationTime	**Date**	No	No	The time the message was created in epoch time.
expireTime	**Date**	No	No	The time the message will be deleted in epoch time.
index	**Integer**	No	No	The index of the message relative to other messages in Moab's memory.
message	**String**	No	Yes	The comment information itself.
messageCount	**Integer**	No	No	The number of times this message has been displayed.
priority	**Double**	No	No	An optional priority that can be attached to the comment.

EmbeddedCredential

Field Name	Type	POST	PUT	Description
name	**String**	No	No	
type	**CredentialType**	No	No	

CredentialType

Value	Description
USER	
GROUP	
ACCOUNT	
CLASS	
QOS	
NOT_SPECIFIED	

ReservationRequirement

Represents all the types of requirements a user can request while creating a reservation.

Field Name	Type	POST	PUT	Description
architecture	**String**	Yes	No	Required architecture.
featureList	**Set<String>**	Yes	No	The list of features required for this reservation.
featureMode	**String**	No	No	Required feature mode.
memory	**Integer**	Yes	No	Required node memory, in MB.
nodeCount	**Integer**	No	No	Required number of nodes.
nodeIds	**Set<String>**	No	No	The list of node IDs required for this reservation.

Field Name	Type	POST	PUT	Description
os	**String**	Yes	No	Required Operating System.
taskCount	**Integer**	Yes	No	Required task count.

ReservationStatistics

Represents some basic statistical information that is kept about the usage of reservations. All metrics that are kept track relate to processor-seconds usage.

Field Name	Type	POST	PUT	Description
caps	**Double**	No	No	The current active processor-seconds in the last reported iteration.
cips	**Double**	No	No	The current idle processor-seconds in the last reported iteration.
taps	**Double**	No	No	The total active processor-seconds over the life of the reservation.
tips	**Double**	No	No	The total idle processor-seconds over the life of the reservation.

Trigger

Field Name	Type	POST	PUT	Description
id	**String**	No	No	Trigger id - internal ID used by moab to track triggers
action	**String**	No	No	For exec atype triggers, signifies executable and arguments. For jobpreempt atype triggers, signifies PREEMPTPOLICY to apply to jobs that are running on allocated resources. For changeparam atype triggers, specifies the parameter to change and its new value (using the same syntax and behavior as the changeparam command).

Field Name	Type	POST	PUT	Description
actionType	TriggerActionType	No	No	
blockTime	**Date**	No	No	Time (in seconds) Moab will suspend normal operation to wait for trigger execution to finish. Use caution as Moab will completely stop normal operation until BlockTime expires.
description	**String**	No	No	
eventType	TriggerEventType	No	No	
expireTime	**Date**	No	No	Time at which trigger should be terminated if it has not already been activated.
failOffset	**Date**	No	No	Specifies the time (in seconds) that the threshold condition must exist before the trigger fires.
flags	Set<TriggerFlag>	No	No	
interval	**Boolean**	No	No	When used in conjunction with MultiFire and RearmTime trigger will fire at regular intervals. Can be used with TriggerEventType.EPOCH to create a Standing Trigger. Defaults to false
maxRetry	**Integer**	No	No	Specifies the number of times Action will be attempted before the trigger is designated a failure.
multiFire	**Boolean**	No	No	Specifies whether this trigger can fire multiple times. Defaults to false.
name	**String**	No	No	Trigger name - can be auto assigned by moab or requested. Alphanumeric up to 16 characters in length
objectId	**String**	No	No	The ID of the object which this is attached to.

Field Name	Type	POST	PUT	Description
objectType	**String**	No	No	The type of object which this is attached to. Possible values: • vm - Virtual Machine
offset	**Date**	No	No	Relative time offset from event when trigger can fire.
period	**TriggerPeriod**	No	No	Can be used in conjunction with Offset to have a trigger fire at the beginning of the specified period. Can be used with EType epoch to create a standing trigger.
rearmTime	**Date**	No	No	Time between MultiFire triggers; rearm time is enforced from the trigger event time.
requires	**String**	No	No	Variables this trigger requires to be set or not set before it will fire. Preceding the string with an exclamation mark (!) indicates this variable must NOT be set. Used in conjunction with Sets to create trigger dependencies.
sets	**String**	No	No	Variable values this trigger sets upon success or failure. Preceding the string with an exclamation mark (!) indicates this variable is set upon trigger failure. Preceding the string with a caret (^) indicates this variable is to be exported to the parent object when the current object is destroyed through a completion event. Used in conjunction with Requires to create trigger dependencies.

Field Name	Type	POST	PUT	Description
threshold	**String**	No	No	Reservation usage threshold - When reservation usage drops below Threshold, trigger will fire. Threshold usage support is only enabled for reservations and applies to percent processor utilization. gmetric thresholds are supported with job, node, credential, and reservation triggers. See Threshold Triggers in the Moab Workload Manager documentation for more information.
timeout	**Date**	No	No	Time allotted to this trigger before it is marked as unsuccessful and its process (if any) killed.
unsets	**String**	No	No	Variable this trigger destroys upon success or failure.

TriggerActionType

This enumeration specifies the action type of a trigger.

Value	Description
CANCEL	Only apply to reservation triggers
CHANGE_PARAM	
JOB_PREEMPT	This indicates that the trigger should preempt all jobs currently allocating resources assigned to the trigger's parent object. Only apply to reservation triggers.
MAIL	
THRESHOLD	
INTERNAL	
EXEC	

TriggerEventType

This enumeration specifies the event type of a trigger.

Value	Description
CANCEL	
CHECKPOINT	
CREATE	
END	
EPOCH	
FAIL	
HOLD	
MIGRATE	
MODIFY	
PREEMPT	
STANDING	
START	
THRESHOLD	

TriggerFlag

This enumeration specifies a flag belonging to a trigger.

Value	Description
ATTACH_ERROR	If the trigger outputs anything to stderr, Moab will attach this as a message to the trigger object.
CLEANUP	If the trigger is still running when the parent object completes or is canceled, the trigger will be killed.

Value	Description
CHECKPOINT	Moab should always checkpoint this trigger. See Checkpointing a Trigger in the Moab Workload Manager documentation for more information.
GLOBAL_VARS	The trigger will look in the name space of all nodes with the globalvars flag in addition to its own name space. A specific node to search can be specified using the following format: globalvars+node_id
INTERVAL	Trigger is periodic.
MULTIFIRE	Trigger can fire multiple times.
OBJECT_XML_ STDIN	Trigger passes its parent's object XML information into the trigger's stdin. This only works for exec triggers with reservation type parents.
USER	The trigger will execute under the user ID of the object's owner. If the parent object is sched, the user to run under may be explicitly specified using the format user+<username>, for example flags=user+john:
GLOBAL_ TRIGGER	The trigger will be (or was) inserted into the global trigger list.
ASYNCHRONOUS	An asynchronous trigger.
LEAVE_FILES	Do not remove stderr and stdout files.
PROBE	The trigger's stdout will be monitored.
PROBE_ALL	The trigger's stdout will be monitored.
GENERIC_ SYSTEM_JOB	The trigger belongs to a generic system job (for checkpointing).
REMOVE_STD_ FILES	The trigger will delete stdout/stderr files after it has been reset.
RESET_ON_ MODIFY	The trigger resets if the object it is attached to is modified, even if multifire is not set.

Value	Description
SOFT_KILL	By default, a `SIGKILL` (kill -9) signal is sent to the kill the script when a trigger times out. This flag will instead send a `SIGTERM` (kill -15) signal to kill the script. The `SIGTERM` signal will allow the script to trap the signal so that the script can clean up any residual information on the system (instead of just dying, as with the `SIGKILL` signal). NOTE: A timed-out trigger will only receive one kill signal. This means that if you specify this flag, a timed-out trigger will only receive the `SIGTERM` signal, and never the `SIGKILL` signal.

TriggerPeriod

This enumeration specifies the period of a trigger.

Value	Description
MINUTE	
HOUR	
DAY	
WEEK	
MONTH	

Related topics

- **Reservations** on page 1624

Fields: Resource Types

See the associated **Resource types** on page 1632 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`resource-types`	**Permissions** on page 1571
Hooks filename	`resource-types.groovy`	**Pre and post-processing hooks** on page 1412

Type	Value	Additional information
Distinct query-supported	No	**Distinct** on page 1504

API version 3

ResourceType

Represents a resource type in Moab Workload Manager.

Field Name	Type	Description
id	**String**	The unique ID of this resource type.

API version 2

ResourceType

Represents a resource type in Moab Workload Manager.

Field Name	Type	Description
id	**String**	The unique ID of this resource type.

Related topics

- **Resource types** on page 1632

Fields: Roles

See the associated **Roles** on page 1633 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`roles`	**Permissions** on page 1571
Hooks filename	`roles.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	Yes	**Distinct** on page 1504

API version 3

Role

A role defines a set of permissions that are based on the proxy-user. If no proxy user is specified then access to objects in MWS are limited to its application permissions. For example if the application has permission to update all resources in MWS and no proxy-user is specified in the request then the request can access all resources in MWS.

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of this role.
description	**String**	Yes	Yes	The role description.
name	**String**	Yes	Yes	The unique human-readable name of this role. Required during POST.
permissions	**List<Permission>**	Yes	Yes	The set of permissions enforced based on the proxy-user.
scope	**PrivilegeScope**	No	No	

Permission

Represents a permission

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of this role.
action	**String**	No	No	The action that can be performed on the resource.
description	**String**	No	No	A description of this permission.
fieldPath	**String**	No	No	Field level ACL control, if null or '*', all fields are accessible, otherwise requests must match dot delimited path. Currently only checked when doing writable actions. Example - attributes.*: create|update
label	**String**	No	No	A human readable label for this permission.

Field Name	Type	POST	PUT	Description
resource	**String**	No	No	The resource the permission applies to.
resourceFilter	**Map<String, Map>**	No	No	A map used to limit which resource instances this permission applies to. If this is null then the permission will apply to all instances of the resource. For api permissions the filter uses mongo query syntax.
scope	**PrivilegeScope**	No	No	Whether this permission applies to the principal's tenant-associated resources or globally
type	**String**	No	No	The type of the permission. Only 'api' type permissions are enforced.

PrivilegeScope

Some permissions and roles ignore tenants and apply globally. Others apply only to the resources associated with the principal's tenants.

Value	Description
GLOBAL	Describes a role or permission that applies globally, irrespective of the principal's tenants. This scope can be applied to any role or permission.
TENANT	Describes a role or permission that applies only to the resources associated with the principal's tenants. This scope can be applied to any role, but only to those permissions associated with tenanted resources (e.g. nodes, services, etc.).
NONE	Scope doesn't apply to some permissions. As of right now, all non-domain permissions (e.g. those created by Viewpoint) don't need a scope. NONE should therefore be assigned to all non-domain permissions.

PrivilegeScope

Some permissions and roles ignore tenants and apply globally. Others apply only to the resources associated with the principal's tenants.

Value	Description
GLOBAL	Describes a role or permission that applies globally, irrespective of the principal's tenants. This scope can be applied to any role or permission.

Value	Description
TENANT	Describes a role or permission that applies only to the resources associated with the principal's tenants. This scope can be applied to any role, but only to those permissions associated with tenanted resources (e.g. nodes, services, etc.).
NONE	Scope doesn't apply to some permissions. As of right now, all non-domain permissions (e.g. those created by Viewpoint) don't need a scope. NONE should therefore be assigned to all non-domain permissions.

API version 2

Role

A role defines a set of permissions that are based on the proxy-user. If no proxy user is specified then access to objects in MWS are limited to its application permissions. For example if the application has permission to update all resources in MWS and no proxy-user is specified in the request then the request can access all resources in MWS.

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of this role.
description	**String**	Yes	Yes	The role description.
name	**String**	Yes	Yes	The unique human-readable name of this role. Required during POST.
permissions	**List<Permission>**	Yes	Yes	The set of permissions enforced based on the proxy-user.
scope	**PrivilegeScope**	No	No	

Permission

Represents a permission

Field Name	Type	POST	PUT	Description
id	**String**	No	No	The unique ID of this role.
action	**String**	No	No	The action that can be performed on the resource.
description	**String**	No	No	A description of this permission.
fieldPath	**String**	No	No	Field level ACL control, if null or '*', all fields are accessible, otherwise requests must match dot delimited path. Currently only checked when doing writable actions. Example - attributes.*: create|update
label	**String**	No	No	A human readable label for this permission.

Field Name	Type	POST	PUT	Description
resource	**String**	No	No	The resource the permission applies to.
resourceFilter	**Map<String, Map>**	No	No	A map used to limit which resource instances this permission applies to. If this is null then the permission will apply to all instances of the resource. For api permissions the filter uses mongo query syntax.
scope	**PrivilegeScope**	No	No	Whether this permission applies to the principal's tenant-associated resources or globally
type	**String**	No	No	The type of the permission. Only 'api' type permissions are enforced.

PrivilegeScope

Some permissions and roles ignore tenants and apply globally. Others apply only to the resources associated with the principal's tenants.

Value	Description
GLOBAL	Describes a role or permission that applies globally, irrespective of the principal's tenants. This scope can be applied to any role or permission.
TENANT	Describes a role or permission that applies only to the resources associated with the principal's tenants. This scope can be applied to any role, but only to those permissions associated with tenanted resources (e.g. nodes, services, etc.).
NONE	Scope doesn't apply to some permissions. As of right now, all non-domain permissions (e.g. those created by Viewpoint) don't need a scope. NONE should therefore be assigned to all non-domain permissions.

PrivilegeScope

Some permissions and roles ignore tenants and apply globally. Others apply only to the resources associated with the principal's tenants.

Value	Description
GLOBAL	Describes a role or permission that applies globally, irrespective of the principal's tenants. This scope can be applied to any role or permission.

Value	Description
TENANT	Describes a role or permission that applies only to the resources associated with the principal's tenants. This scope can be applied to any role, but only to those permissions associated with tenanted resources (e.g. nodes, services, etc.).
NONE	Scope doesn't apply to some permissions. As of right now, all non-domain permissions (e.g. those created by Viewpoint) don't need a scope. NONE should therefore be assigned to all non-domain permissions.

Related topics

- **Roles** on page 1633

Fields: Report Samples

See the associated **Reports** on page 1614 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`reports/samples`	**Permissions** on page 1571
Hooks filename	`reports.samples.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	Yes	**Distinct** on page 1504

API version 3

Sample

A single snapshot of system state. It can contain all the same information as Datapoint.data in the sample's data field.

Field Name	Type	POST	Description
id	**Long**	No	
agent	**String**	No	A unique identifier for the agent that recorded this sample.
data	**Map<String, Map>**	No	Arbitrary data that was recorded for this sample. Defaults to an empty object if none is supplied.
timestamp	**Date**	No	The date and time at which this sample was recorded. Defaults to the current date if none is supplied.

API version 2

Sample

A single snapshot of system state. It can contain all the same information as Datapoint.data in the sample's data field.

Field Name	Type	POST	Description
id	**Long**	No	
agent	**String**	No	A unique identifier for the agent that recorded this sample.
data	**Map<String, Map>**	No	Arbitrary data that was recorded for this sample. Defaults to an empty object if none is supplied.
timestamp	**Date**	No	The date and time at which this sample was recorded. Defaults to the current date if none is supplied.

Related topics

- **Reports** on page 1614

Fields: Standing Reservations

See the associated **Standing reservations** on page 1639 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`standing-reservations`	**Permissions** on page 1571
Hooks filename	`standing-reservations.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	No	**Distinct** on page 1504

API version 3

StandingReservation

This class represents a standing reservation.

A standing reservation is any reservation that is not a one-time reservation. This includes reservations that recur every day or every week, or infinite reservations.

Field Name	Type	Description
id	**String**	The unique ID of the standing reservation.
access	**ReservationAccess**	If set to ReservationAccess.SHARED, allows a standing reservation to use resources already allocated to other non-job reservations. Otherwise, these other reservations block resource access.
accounts	**Set<String>**	Specifies that jobs with the associated accounts may use the resources contained within this reservation.
aclRules	**Set<AclRule>**	The set of access control rules associated with this standing reservation.
chargeAccount	**String**	Specifies the account to which Moab will charge all idle cycles within the reservation (via the allocation manager).
chargeUser	**String**	Specifies the user to which Moab will charge all idle cycles within the reservation (via the allocation manager). Must be used in conjunction with chargeAccount
classes	**Set<String>**	Specifies that jobs with the associated classes/queues may use the resources contained within this reservation.
clusters	**Set<String>**	Specifies that jobs originating within the listed clusters may use the resources contained within this reservation.
comment	**String**	Specifies a descriptive message associated with the standing reservation and all child reservations

Field Name	Type	Description
days	**Set<String>**	Specifies which days of the week the standing reservation is active. Valid values are Mon, Tue, Wed, Thu, Fri, Sat, Sun, or [ALL].
depth	**Integer**	Specifies the depth of standing reservations to be created, starting at depth 0 (one per period).
disabled	**Boolean**	Specifies if the standing reservation should no longer spawn child reservations.
endOffset	**Long**	The ending offset, in seconds, from the beginning of the current period (DAY or WEEK), for this standing reservation. See examples at startOffset.
flags	**Set<ReservationFlag>**	Specifies special reservation attributes.
groups	**Set<String>**	Specifies the groups allowed access to this standing reservation.
hosts	**Set<String>**	Specifies the set of hosts that the scheduler can search for resources to satisfy the reservation. If specified using the class:X format, Moab only selects hosts that support the specified class. If TASKCOUNT is also specified, only TASKCOUNT tasks are reserved. Otherwise, all matching hosts are reserved.
jobAttributes	**Set<JobFlag>**	Specifies job attributes that grant a job access to the reservation. Values can be specified with a != assignment to only allow jobs NOT requesting a certain feature inside the reservation.
maxJob	**Integer**	Specifies the maximum number of jobs that can run in the reservation.
maxTime	**Integer**	Specifies the maximum time for jobs allowable. Can be used with affinity to attract jobs with same maxTime.
messages	**Set<String>**	Messages associated with the reservation.

Field Name	Type	Description
nodeFeatures	**Set<String>**	Specifies the required node features for nodes that are part of the standing reservation.
os	**String**	Specifies the operating system that should be in place during the reservation. Moab provisions this OS at reservation start and restores the original OS at reservation completion.
owner	**EmbeddedCredential**	Specifies the owner of the reservation. Setting ownership for a reservation grants the user management privileges, including the power to release it. Setting a user as the owner of a reservation gives that user privileges to query and release the reservation. For sandbox reservations, sandboxes are applied to a specific peer only if owner is set to CLUSTER:<PEERNAME>
partition	**String**	Specifies the partition in which to create the standing reservation. Defaults to ALL.
period	**TimeWindow**	Period of the Standing reservation. Defaults to TimeWindow.DAY.
procLimit	**IntLimit**	Specifies the processor limit for jobs requesting access to this standing reservation.
psLimit	**IntLimit**	Specifies the processor-second limit for jobs requesting access to this standing reservation.
qoses	**Set<String>**	Specifies that jobs with the listed QoS names can access the reserved resources.
reservationAccessList	**Set<Reservation>**	A list of reservations to which the specified reservation has access.
reservationGroup	**String**	The group of the reservation.

Field Name	Type	Description
resources	**Map<String, Integer>**	Specifies what resources constitute a single standing reservation task. (Each task must be able to obtain all of its resources as an atomic unit on a single node.) Supported resources currently include the following: • PROCS (number of processors) • MEM (real memory in MB) • DISK (local disk in MB) • SWAP (virtual memory in MB)
rollbackOffset	**Integer**	Specifies the minimum time in the future at which the reservation may start. This offset is rolling meaning the start time of the reservation will continuously roll back into the future to maintain this offset. Rollback offsets are a good way of providing guaranteed resource access to users under the conditions that they must commit their resources in the future or lose dedicated access. See QoS Credential in the Moab Workload Manager documentation for more information on quality of service and service level agreements.
startOffset	**Long**	The starting offset, in seconds, from the beginning of the current period (DAY or WEEK), for this standing reservation. If period is DAY, the offset is from midnight (00:00) of the current day. If period is WEEK, the offset is from midnight Sunday of the current week. Example 1: For a standing reservation that begins at 9:00 and ends at 17:00 every day, period is DAY, startOffset is 32400 (9*60*60), and endOffset is 61200 (17*60*60). Example 2: For a standing reservation that begins at 9:00 Monday and ends at 17:00 Friday every week, period is WEEK, startOffset is 118800 ((24+9)*60*60), and endOffset is 493200 (((5*24)+17)*60*60).

Field Name	Type	Description
taskCount	**Integer**	Specifies how many tasks should be reserved for the reservation Default is 0 (unlimited tasks).
tasksPerNode	**Integer**	Specifies the minimum number of tasks per node that must be available on eligible nodes. Default is 0 (no TPN constraint)
timeLimit	**Integer**	Specifies the maximum allowed overlap between the standing reservation and a job requesting resource access. Default is null (-1 in moab)
triggers	**Set<Trigger>**	Triggers associated with the reservation.
type	**String**	The type of the reservation.
users	**Set<String>**	Specifies which users have access to the resources reserved by this reservation.

ReservationAccess

The access type of a standing reservation. If set to SHARED, allows a standing reservation to use resources already allocated to other non-job reservations. Otherwise, these other reservations block resource access.

Value	Description
DEDICATED	
SHARED	

AclRule

This class represents a rule that can be in Moab's access control list (ACL) mechanism.

The basic AclRule information is the object's name and type. The type directly maps to an AclType value. The default mechanism Moab uses to check the ACL for a particular item is if the user or object coming in has ANY of the values in the ACL, then the user or object is given access. If no values match the user or object in question, the user or object is rejected access.

Field Name	Type	Description
affinity	**AclAffinity**	Reservation ACLs allow or deny access to reserved resources but they may also be configured to affect a job's affinity for a particular reservation. By default, jobs gravitate toward reservations through a mechanism known as positive affinity. This mechanism allows jobs to run on the most constrained resources leaving other, unreserved resources free for use by other jobs that may not be able to access the reserved resources. Normally this is a desired behavior. However, sometimes, it is desirable to reserve resources for use only as a last resort-using the reserved resources only when there are no other resources available. This last resort behavior is known as negative affinity. Defaults to AclAffinity.POSITIVE.
comparator	**ComparisonOperator**	The type of comparison to make against the ACL object. Defaults to ComparisonOperator.EQUAL.
type	**AclType**	The type of the object that is being granted (or denied) access.
value	**String**	The name of the object that is being granted (or denied) access.

AclAffinity

This enumeration describes the values available for describing how a rule is used in establishing access to an object in Moab. Currently, these ACL affinities are used only for granting access to reservations.

Value	Description
NEGATIVE	Access to the object is repelled using this rule until access is the last choice.
NEUTRAL	Access to the object is not affected by affinity.
POSITIVE	Access to the object is looked at as the first choice.
PREEMPTIBLE	Access to the object given the rule gives preemptible status to the accessor. Supported only during GET.

Value	Description
REQUIRED	The rule in question must be satisified in order to gain access to the object. Supported only during GET.
UNAVAILABLE	The rule does not have its affinity available. Supported only during GET.

ComparisonOperator

This enumeration is used when Moab needs to compare items. One such use is in Access Control Lists (ACLs).

Value	Description
GREATER_THAN	Valid values: ">", "gt"
GREATER_THAN_OR_EQUAL	Valid values: ">=", "ge"
LESS_THAN	Valid values: "<", "lt"
LESS_THAN_OR_EQUAL	Valid values: "<=", "le"
EQUAL	Valid values: "==", "eq", "="
NOT_EQUAL	Valid values: "!=", "ne", "<>"
LEXIGRAPHIC_SUBSTRING	Valid value: "%<"
LEXIGRAPHIC_NOT_EQUAL	Valid value: "%!"
LEXIGRAPHIC_EQUAL	Valid value: "%="

AclType

This enumeration describes the values available for the type of an ACL Rule.

Value	Description
USER	User

Value	Description
GROUP	Group
ACCOUNT	Account or Project
CLASS	Class or Queue
QOS	Quality of Service
CLUSTER	Cluster
JOB_ID	Job ID
RESERVATION_ID	Reservation ID
JOB_TEMPLATE	Job Template
JOB_ATTRIBUTE	Job Attribute
DURATION	Duration in Seconds
PROCESSOR_SECONDS	Processor Seconds
JPRIORITY	Not supported
MEMORY	Not supported
NODE	Not supported
PAR	Not supported
PROC	Not supported
QTIME	Not supported
QUEUE	Not supported
RACK	Not supported

Value	Description
SCHED	Not supported
SYTEM	Not supported
TASK	Not supported
VC	Not supported
XFACTOR	Not supported

ReservationFlag

The flag types of a reservation.

Value	Description
ALLOWJOBOVERLAP	Allows jobs to overlap this Reservation, but not start during it (unless they have ACL access).
APPLYPROFRESOURCES	Only apply resource allocation info from profile.
DEADLINE	Reservation should be scheduled against a deadline.
IGNIDLEJOBS	Ignore idle job reservations.
IGNJOBRSV	Ignore job reservations, but not user or other reservations.
CHARGE	Charge the idle cycles in the accounting manager.
NOVMMIGRATIONS	Override the VM Migration Policy and don't migrate VMs that overlap this reservation.
OWNERPREEMPTIGNOREMINTIME	Owner ignores preemptmintime for this reservation.
PROVISION	Reservation should be capable of provisioning.
NOACLOVERLAP	Reservation will not look at ACLs to overlap job (when using exclusive).

Value	Description
ADVRES	If set, the reservation is created in advance of needing it.
ADVRESJOBDESTROY	Cancel any jobs associated with the reservation when it is released.
ALLOWGRID	The reservation is set up for use in a grid environment.
ALLOWPRSV	Personal reservations can be created within the space of this standing reservation (and ONLY this standing reservation). By default, when a standing reservation is given the flag ALLOWPRSV, it is given the ACL rule USER==ALL+ allowing all jobs and all users access.
BYNAME	Reservation only allows access to jobs that meet reservation ACLs and explicitly request the resources of this reservation using the job ADVRES flag.
DEDICATEDNODE	If set, only one active reservation is allowed on a node.
DEDICATEDRESOURCE	The reservation is only placed on resources that are not reserved by any other reservation, including jobs and other reservations.
EXCLUDEJOBS	Makes a reservation job exclusive, where only one job can run in the reservation.
ENDTRIGHASFIRED	A trigger has finished firing.
ENFORCENODESET	Enforce node sets when creating reservation.
EXCLUDEALLBUTSB	Reservation only shares resources with sandboxes.
EXCLUDEMYGROUP	Exclude reservations within the same group.
IGNRSV	Forces the reservation onto nodes regardless of whether there are other reservations currently residing on the nodes.
IGNSTATE	Request ignores existing resource reservations, allowing the reservation to be forced onto available resources even if this conflicts with other reservations.

Value	Description
ISACTIVE	If set, the reservation is currently active.
ISCLOSED	If set, the reservation is closed.
ISGLOBAL	If set the reservation applies to all resources.
OWNERPREEMPT	The owner of the reservation is given preemptor status for resources contained in the reservation.
PARENTLOCK	The reservation can only be destroyed by destroying its parent.
PREEMPTEE	The reservation is preemptible.
PLACEHOLDER	The reservation is a placeholder for resources.
PRSV	The reservation is a non-adminstrator, non-standing reservation, user-created reservation.
REQFULL	The reservation will fail if all resources requested cannot be allocated.
SCHEDULEVCRSV	The reservation was created as part of a schedule VC command. This pertains to reservations creating while scheduling MWS Services, and these are filtered from the MWS output of reservations.
SINGLEUSE	The reservation is automatically removed after completion of the first job to use the reserved resources.
SPACEFLEX	The reservation is allowed to adjust resources allocated over time in an attempt to optimize resource utilization.
STANDINGRSV	If set, the reservation was created by a standing reservation instance.
STATIC	Makes a reservation ineligible to modified or canceled by an administrator.
SYSTEMJOB	The reservation was created by a system job.
TIMEFLEX	The reservation is allowed to adjust the reserved time frame in an attempt to optimize resource utilization.

Value	Description
TRIGHASFIRED	The reservation has one or more triggers that have fired on it.
WASACTIVE	The reservation was previously active.
EVACVMS	Evacuate virtual machines on the node when the reservation starts.

JobFlag

This enumeration specifies the flag types of a job.

Value	Description
NONE	
BACKFILL	The job is using backfill to run.
COALLOC	The job can use resources from multiple resource managers and partitions.
ADVRES	The job requires the use of a reservation.
NOQUEUE	The job will attempt to execute immediately or fail.
ARRAYJOB	The job is part of a job array.
ARRAYJOBPARLOCK	This array job will only run in one partition.
ARRAYJOBPARSPAN	This array job will span partitions (default).
ARRAYMASTER	This job is the master of a job array.
BESTEFFORT	The job will succeed if even partial resources are available.
RESTARTABLE	The job is restartable.
SUSPENDABLE	The job is suspendable.
HASPREEMPTED	This job preempted other jobs to start.

Value	Description
PREEMPTEE	The job is a preemptee and therefore can be preempted by other jobs.
PREEMPTOR	The job is a preemptor and therefore can preempt other jobs.
RSVMAP	The job is based on a reservation.
SPVIOLATION	The job was started with a soft policy violation.
IGNNODEPOLICIES	The job will ignore node policies.
IGNPOLICIES	The job will ignore idle, active, class, partition, and system policies.
IGNNODESTATE	The job will ignore node state in order to run.
IGNIDLEJOBRSV	The job can ignore idle job reservations. The job granted access to all idle job reservations.
INTERACTIVE	The job needs to interactive input from the user to run.
FSVIOLATION	The job was started with a fairshare violation.
GLOBALQUEUE	The job is directly submitted without doing any authentication.
NORESOURCES	The job is a system job that does not need any resources.
NORMSTART	The job will not query a resource manager to run.
CLUSTERLOCKED	The job is locked into the current cluster and cannot be migrated elsewhere. This is for grid mode.
FRAGMENT	The job can be run across multiple nodes in individual chunks.
SYSTEMJOB	The job is a system job which simply runs on the same node that Moab is running on. This is usually used for running scripts and other executables in workflows.
ADMINSETIGNPOLICIES	The IGNPOLICIES flag was set by an administrator.

Value	Description
EXTENDSTARTWALLTIME	The job duration (walltime) was extended at job start.
SHAREDMEM	The job will share its memory across nodes.
BLOCKEDBYGRES	The job's generic resource requirement caused the job to start later.
GRESONLY	The job is requesting only generic resources, no compute resources.
TEMPLATESAPPLIED	The job has had all applicable templates applied to it.
META	META job, just a container around resources.
WIDERSVSEARCHALGO	This job prefers the wide search algorithm.
VMTRACKING	The job is a VMTracking job for an externally-created VM (via job template).
DESTROYTEMPLATESUBMITTED	A destroy job has already been created from the template for this job.
PROCSPECIFIED	The job requested processors on the command line.
CANCELONFIRSTFAILURE	Cancel job array on first array job failure.
CANCELONFIRSTSUCCESS	Cancel job array on first array job success.
CANCELONANYFAILURE	Cancel job array on any array job failure.
CANCELONANYSUCCESS	Cancel job array on any array job success.
CANCELONEXITCODE	Cancel job array on a specific exit code.
NOVMMIGRATE	Do not migrate the virtual machine that this job sets up.
PURGEONSUCCESSONLY	Only purge the job if it completed successfully

EmbeddedCredential

Field Name	Type	Description
name	String	
type	CredentialType	

CredentialType

Value	Description
USER	
GROUP	
ACCOUNT	
CLASS	
QOS	
NOT_SPECIFIED	

TimeWindow

This enumeration represents some common time windows. It can be used when for many purposes, but was created specifically for statistics.

Value	Description
MINUTE	
HOUR	
DAY	
WEEK	
MONTH	

Value	Description
YEAR	
INFINITY	

IntLimit

Field Name	Type	Description
qualifier	**String**	One of: • < • <= • == • >= • >
value	**Integer**	

Reservation

A reservation is the mechanism by which Moab guarantees the availability of a set of resources at a particular time. Each reservation consists of three major components: (1) a set of resources, (2) a time frame, and (3) an access control list. It is a scheduler role to ensure that the access control list is not violated during the reservation's lifetime (that is, its time frame) on the resources listed. For example, a reservation may specify that node002 is reserved for user Tom on Friday. The scheduler is thus constrained to make certain that only Tom's jobs can use node002 at any time on Friday.

Field Name	Type	Description
id	**String**	The unique ID of the reservation.
accountingAccount	**String**	Accountable Account.
accountingGroup	**String**	Accountable Group.

Field Name	Type	Description
accountingQOS	**String**	Accountable QOS.
accountingUser	**String**	Accountable User.
aclRules	**Set<AclRule>**	The set of access control rules associated with this reservation.
allocatedNodeCount	**Integer**	The number of allocated nodes for this reservation.
allocatedNodes	**Set<DomainProxyVersion1>**	The nodes allocated to the reservation.
allocatedProcessorCount	**Integer**	The number of allocated processors.
allocatedTaskCount	**Integer**	The number of allocated tasks.
comments	**String**	Reservation's comments or description.
creationDate	**Date**	Creation date. Automatically set by Moab when a user creates the reservation.
duration	**Long**	The duration of the reservation (in seconds).
endDate	**Date**	The end date of the reservation. This is especially useful for one-time reservations, which have an exact time for when a reservation ends.
excludeJobs	**Set<String>**	The list of jobs to exclude. Client must also set the IGNJOBRSV reservation flag. Otherwise, results are undefined. Used only during reservation creation.
expireDate	**Date**	The date/time when the reservation expires and vacates.
flags	**Set<ReservationFlag>**	The flags associated with the reservation.
globalId	**String**	Global reservation ID.

Field Name	Type	Description
hostListExpression	**String**	The list of nodes a user can select to reserve. This may or may not be the nodes that are currently allocated to this reservation. Note: Either hostListExpression or taskCount must be set to create a reservation.
idPrefix	**String**	The user-specified prefix for this reservation. If provided, Moab combines the idPrefix with an integer, and the combination is the unique identifier for this reservation.
isActive	**Boolean**	State whether or not this reservation is currently active.
isTracked	**Boolean**	States whether reservation resource usage is tracked.
label	**String**	When a label is assigned to a reservation, the reservation can then be referenced by that label as well as by the reservation name.
maxTasks	**Integer**	The maximum number of tasks for this reservation.
messages	**Set<MessageVersion1>**	Messages for the reservation.
owner	**EmbeddedCredential**	The owner of the reservation
partitionId	**String**	The ID of the partition this reservation is for.
profile	**String**	The profile that this reservation is using. A profile is a specification of attributes that all reservations share. Used only during reservation creation.
requirements	**ReservationRequirement**	The reservation's requirements.
reservationGroup	**String**	The reservation group to which the reservation belongs.

Field Name	Type	Description
resources	**Map<String, Integer>**	The reservation's resources. This field is a map, where the key is PROCS, MEM DISK, SWAP, or one or more user-defined keys.
startDate	**Date**	The start time for the reservation. This is especially useful for one-time reservations, which have an exact time for when a reservation starts.
statistics	**ReservationStatistics**	The reservation's statistical information.
subType	**String**	The reservation sub-type.
taskCount	**Integer**	The number of tasks that must be allocated to satisfy the reservation request. Note: Either hostListExpression or taskCount must be set to create a reservation.
trigger	**Trigger**	Trigger for reservation. Used only during reservation creation.
triggerIds	**Set<String>**	The IDs of the triggers attached to this reservation.
uniqueIndex	**String**	The globally-unique reservation index.
variables	**Map<String, Map>**	The set of variables for this reservation.

DomainProxyVersion1

Field Name	Type	Description
id	**String**	The id of the object.

MessageVersion1

Field Name	Type	Description
author	**String**	The author of the message.
creationTime	**Date**	The time the message was created in epoch time.
expireTime	**Date**	The time the message will be deleted in epoch time.
index	**Integer**	The index of the message relative to other messages in Moab's memory.
message	**String**	The comment information itself.
messageCount	**Integer**	The number of times this message has been displayed.
priority	**Double**	An optional priority that can be attached to the comment.

ReservationRequirement

Represents all the types of requirements a user can request while creating a reservation.

Field Name	Type	Description
architecture	**String**	Required architecture.
featureList	**Set<String>**	The list of features required for this reservation.
featureMode	**String**	Required feature mode.
memory	**Integer**	Required node memory, in MB.
nodeCount	**Integer**	Required number of nodes.
nodeIds	**Set<String>**	The list of node IDs required for this reservation.
os	**String**	Required Operating System.
taskCount	**Integer**	Required task count.

ReservationStatistics

Represents some basic statistical information that is kept about the usage of reservations. All metrics that are kept track relate to processor-seconds usage.

Field Name	Type	Description
caps	**Double**	The current active processor-seconds in the last reported iteration.
cips	**Double**	The current idle processor-seconds in the last reported iteration.
taps	**Double**	The total active processor-seconds over the life of the reservation.
tips	**Double**	The total idle processor-seconds over the life of the reservation.

Trigger

Field Name	Type	Description
id	**String**	Trigger id - internal ID used by moab to track triggers
action	**String**	For exec atype triggers, signifies executable and arguments. For jobpreempt atype triggers, signifies PREEMPTPOLICY to apply to jobs that are running on allocated resources. For changeparam atype triggers, specifies the parameter to change and its new value (using the same syntax and behavior as the changeparam command).
actionType	**TriggerActionType**	
blockTime	**Date**	Time (in seconds) Moab will suspend normal operation to wait for trigger execution to finish. Use caution as Moab will completely stop normal operation until BlockTime expires.
description	**String**	
eventType	**TriggerEventType**	
expireTime	**Date**	Time at which trigger should be terminated if it has not already been activated.

Field Name	Type	Description
failOffset	**Date**	Specifies the time (in seconds) that the threshold condition must exist before the trigger fires.
flags	**Set<TriggerFlag>**	
interval	**Boolean**	When used in conjunction with MultiFire and RearmTime trigger will fire at regular intervals. Can be used with TriggerEventType.EPOCH to create a Standing Trigger. Defaults to false
maxRetry	**Integer**	Specifies the number of times Action will be attempted before the trigger is designated a failure.
multiFire	**Boolean**	Specifies whether this trigger can fire multiple times. Defaults to false.
name	**String**	Trigger name - can be auto assigned by moab or requested. Alphanumeric up to 16 characters in length
objectId	**String**	The ID of the object which this is attached to.
objectType	**String**	The type of object which this is attached to. Possible values: • vm - Virtual Machine
offset	**Date**	Relative time offset from event when trigger can fire.
period	**TriggerPeriod**	Can be used in conjunction with Offset to have a trigger fire at the beginning of the specified period. Can be used with EType epoch to create a standing trigger.
rearmTime	**Date**	Time between MultiFire triggers; rearm time is enforced from the trigger event time.
requires	**String**	Variables this trigger requires to be set or not set before it will fire. Preceding the string with an exclamation mark (!) indicates this variable must NOT be set. Used in conjunction with Sets to create trigger dependencies.

Field Name	Type	Description
sets	**String**	Variable values this trigger sets upon success or failure. Preceding the string with an exclamation mark (!) indicates this variable is set upon trigger failure. Preceding the string with a caret (^) indicates this variable is to be exported to the parent object when the current object is destroyed through a completion event. Used in conjunction with Requires to create trigger dependencies.
threshold	**String**	Reservation usage threshold - When reservation usage drops below Threshold, trigger will fire. Threshold usage support is only enabled for reservations and applies to percent processor utilization. gmetric thresholds are supported with job, node, credential, and reservation triggers. See Threshold Triggers in the Moab Workload Manager documentation for more information.
timeout	**Date**	Time allotted to this trigger before it is marked as unsuccessful and its process (if any) killed.
unsets	**String**	Variable this trigger destroys upon success or failure.

TriggerActionType

This enumeration specifies the action type of a trigger.

Value	Description
CANCEL	Only apply to reservation triggers
CHANGE_ PARAM	
JOB_ PREEMPT	This indicates that the trigger should preempt all jobs currently allocating resources assigned to the trigger's parent object. Only apply to reservation triggers.
MAIL	
THRESHOLD	
INTERNAL	

Value	Description
EXEC	

TriggerEventType

This enumeration specifies the event type of a trigger.

Value	Description
CANCEL	
CHECKPOINT	
CREATE	
END	
EPOCH	
FAIL	
HOLD	
MIGRATE	
MODIFY	
PREEMPT	
STANDING	
START	
THRESHOLD	

TriggerFlag

This enumeration specifies a flag belonging to a trigger.

Value	Description
ATTACH_ERROR	If the trigger outputs anything to stderr, Moab will attach this as a message to the trigger object.
CLEANUP	If the trigger is still running when the parent object completes or is canceled, the trigger will be killed.
CHECKPOINT	Moab should always checkpoint this trigger. See Checkpointing a Trigger in the Moab Workload Manager documentation for more information.
GLOBAL_VARS	The trigger will look in the name space of all nodes with the globalvars flag in addition to its own name space. A specific node to search can be specified using the following format: globalvars+node_id
INTERVAL	Trigger is periodic.
MULTIFIRE	Trigger can fire multiple times.
OBJECT_XML_ STDIN	Trigger passes its parent's object XML information into the trigger's stdin. This only works for exec triggers with reservation type parents.
USER	The trigger will execute under the user ID of the object's owner. If the parent object is sched, the user to run under may be explicitly specified using the format user+<username>, for example flags=user+john:
GLOBAL_ TRIGGER	The trigger will be (or was) inserted into the global trigger list.
ASYNCHRONOUS	An asynchronous trigger.
LEAVE_FILES	Do not remove stderr and stdout files.
PROBE	The trigger's stdout will be monitored.
PROBE_ALL	The trigger's stdout will be monitored.
GENERIC_ SYSTEM_JOB	The trigger belongs to a generic system job (for checkpointing).

Value	Description
REMOVE_STD_FILES	The trigger will delete stdout/stderr files after it has been reset.
RESET_ON_MODIFY	The trigger resets if the object it is attached to is modified, even if multifire is not set.
SOFT_KILL	By default, a `SIGKILL` (kill -9) signal is sent to the kill the script when a trigger times out. This flag will instead send a `SIGTERM` (kill -15) signal to kill the script. The `SIGTERM` signal will allow the script to trap the signal so that the script can clean up any residual information on the system (instead of just dying, as with the `SIGKILL` signal). NOTE: A timed-out trigger will only receive one kill signal. This means that if you specify this flag, a timed-out trigger will only receive the `SIGTERM` signal, and never the `SIGKILL` signal.

TriggerPeriod

This enumeration specifies the period of a trigger.

Value	Description
MINUTE	
HOUR	
DAY	
WEEK	
MONTH	

Trigger

Field Name	Type	Description
id	**String**	Trigger id - internal ID used by moab to track triggers

Field Name	Type	Description
action	**String**	For exec atype triggers, signifies executable and arguments. For jobpreempt atype triggers, signifies PREEMPTPOLICY to apply to jobs that are running on allocated resources. For changeparam atype triggers, specifies the parameter to change and its new value (using the same syntax and behavior as the changeparam command).
actionType	[**TriggerActionType**]()	
blockTime	**Date**	Time (in seconds) Moab will suspend normal operation to wait for trigger execution to finish. Use caution as Moab will completely stop normal operation until BlockTime expires.
description	**String**	
eventType	[**TriggerEventType**]()	
expireTime	**Date**	Time at which trigger should be terminated if it has not already been activated.
failOffset	**Date**	Specifies the time (in seconds) that the threshold condition must exist before the trigger fires.
flags	[**Set<TriggerFlag>**]()	
interval	**Boolean**	When used in conjunction with MultiFire and RearmTime trigger will fire at regular intervals. Can be used with [TriggerEventType.EPOCH]() to create a Standing Trigger. Defaults to false
maxRetry	**Integer**	Specifies the number of times Action will be attempted before the trigger is designated a failure.
multiFire	**Boolean**	Specifies whether this trigger can fire multiple times. Defaults to false.
name	**String**	Trigger name - can be auto assigned by moab or requested. Alphanumeric up to 16 characters in length
objectId	**String**	The ID of the object which this is attached to.

Field Name	Type	Description
objectType	**String**	The type of object which this is attached to. Possible values: • vm - Virtual Machine
offset	**Date**	Relative time offset from event when trigger can fire.
period	**TriggerPeriod**	Can be used in conjunction with Offset to have a trigger fire at the beginning of the specified period. Can be used with EType epoch to create a standing trigger.
rearmTime	**Date**	Time between MultiFire triggers; rearm time is enforced from the trigger event time.
requires	**String**	Variables this trigger requires to be set or not set before it will fire. Preceding the string with an exclamation mark (!) indicates this variable must NOT be set. Used in conjunction with Sets to create trigger dependencies.
sets	**String**	Variable values this trigger sets upon success or failure. Preceding the string with an exclamation mark (!) indicates this variable is set upon trigger failure. Preceding the string with a caret (^) indicates this variable is to be exported to the parent object when the current object is destroyed through a completion event. Used in conjunction with Requires to create trigger dependencies.
threshold	**String**	Reservation usage threshold - When reservation usage drops below Threshold, trigger will fire. Threshold usage support is only enabled for reservations and applies to percent processor utilization. gmetric thresholds are supported with job, node, credential, and reservation triggers. See Threshold Triggers in the Moab Workload Manager documentation for more information.
timeout	**Date**	Time allotted to this trigger before it is marked as unsuccessful and its process (if any) killed.
unsets	**String**	Variable this trigger destroys upon success or failure.

API version 2

StandingReservation

This class represents a standing reservation.

A standing reservation is any reservation that is not a one-time reservation. This includes reservations that recur every day or every week, or infinite reservations.

Field Name	Type	Description
id	**String**	The unique ID of the standing reservation.
access	**ReservationAccess**	If set to ReservationAccess.SHARED, allows a standing reservation to use resources already allocated to other non-job reservations. Otherwise, these other reservations block resource access.
accounts	**Set<String>**	Specifies that jobs with the associated accounts may use the resources contained within this reservation.
aclRules	**Set<AclRule>**	The set of access control rules associated with this standing reservation.
chargeAccount	**String**	Specifies the account to which Moab will charge all idle cycles within the reservation (via the allocation manager).
chargeUser	**String**	Specifies the user to which Moab will charge all idle cycles within the reservation (via the allocation manager). Must be used in conjunction with chargeAccount
classes	**Set<String>**	Specifies that jobs with the associated classes/queues may use the resources contained within this reservation.
clusters	**Set<String>**	Specifies that jobs originating within the listed clusters may use the resources contained within this reservation.
comment	**String**	Specifies a descriptive message associated with the standing reservation and all child reservations

Field Name	Type	Description
days	**Set<String>**	Specifies which days of the week the standing reservation is active. Valid values are Mon, Tue, Wed, Thu, Fri, Sat, Sun, or [ALL].
depth	**Integer**	Specifies the depth of standing reservations to be created, starting at depth 0 (one per period).
disabled	**Boolean**	Specifies if the standing reservation should no longer spawn child reservations.
endOffset	**Long**	The ending offset, in seconds, from the beginning of the current period (DAY or WEEK), for this standing reservation. See examples at startOffset.
flags	**Set<ReservationFlag>**	Specifies special reservation attributes.
groups	**Set<String>**	Specifies the groups allowed access to this standing reservation.
hosts	**Set<String>**	Specifies the set of hosts that the scheduler can search for resources to satisfy the reservation. If specified using the class:X format, Moab only selects hosts that support the specified class. If TASKCOUNT is also specified, only TASKCOUNT tasks are reserved. Otherwise, all matching hosts are reserved.
jobAttributes	**Set<JobFlag>**	Specifies job attributes that grant a job access to the reservation. Values can be specified with a != assignment to only allow jobs NOT requesting a certain feature inside the reservation.
maxJob	**Integer**	Specifies the maximum number of jobs that can run in the reservation.
maxTime	**Integer**	Specifies the maximum time for jobs allowable. Can be used with affinity to attract jobs with same maxTime.
messages	**Set<String>**	Messages associated with the reservation.

Field Name	Type	Description
nodeFeatures	**Set<String>**	Specifies the required node features for nodes that are part of the standing reservation.
os	**String**	Specifies the operating system that should be in place during the reservation. Moab provisions this OS at reservation start and restores the original OS at reservation completion.
owner	**EmbeddedCredential**	Specifies the owner of the reservation. Setting ownership for a reservation grants the user management privileges, including the power to release it. Setting a user as the owner of a reservation gives that user privileges to query and release the reservation. For sandbox reservations, sandboxes are applied to a specific peer only if owner is set to CLUSTER:<PEERNAME>
partition	**String**	Specifies the partition in which to create the standing reservation. Defaults to ALL.
period	**TimeWindow**	Period of the Standing reservation. Defaults to TimeWindow.DAY.
procLimit	**IntLimit**	Specifies the processor limit for jobs requesting access to this standing reservation.
psLimit	**IntLimit**	Specifies the processor-second limit for jobs requesting access to this standing reservation.
qoses	**Set<String>**	Specifies that jobs with the listed QoS names can access the reserved resources.
reservationAccessList	**Set<Reservation>**	A list of reservations to which the specified reservation has access.
reservationGroup	**String**	The group of the reservation.

Field Name	Type	Description
resources	**Map<String, Integer>**	Specifies what resources constitute a single standing reservation task. (Each task must be able to obtain all of its resources as an atomic unit on a single node.) Supported resources currently include the following: • PROCS (number of processors) • MEM (real memory in MB) • DISK (local disk in MB) • SWAP (virtual memory in MB)
rollbackOffset	**Integer**	Specifies the minimum time in the future at which the reservation may start. This offset is rolling meaning the start time of the reservation will continuously roll back into the future to maintain this offset. Rollback offsets are a good way of providing guaranteed resource access to users under the conditions that they must commit their resources in the future or lose dedicated access. See QoS Credential in the Moab Workload Manager documentation for more information on quality of service and service level agreements.
startOffset	**Long**	The starting offset, in seconds, from the beginning of the current period (DAY or WEEK), for this standing reservation. If period is DAY, the offset is from midnight (00:00) of the current day. If period is WEEK, the offset is from midnight Sunday of the current week. Example 1: For a standing reservation that begins at 9:00 and ends at 17:00 every day, period is DAY, startOffset is 32400 (9*60*60), and endOffset is 61200 (17*60*60). Example 2: For a standing reservation that begins at 9:00 Monday and ends at 17:00 Friday every week, period is WEEK, startOffset is 118800 ((24+9)*60*60), and endOffset is 493200 (((5*24)+17)*60*60).

Field Name	Type	Description
taskCount	**Integer**	Specifies how many tasks should be reserved for the reservation Default is 0 (unlimited tasks).
tasksPerNode	**Integer**	Specifies the minimum number of tasks per node that must be available on eligible nodes. Default is 0 (no TPN constraint)
timeLimit	**Integer**	Specifies the maximum allowed overlap between the standing reservation and a job requesting resource access. Default is null (-1 in moab)
triggers	**Set<Trigger>**	Triggers associated with the reservation.
type	**String**	The type of the reservation.
users	**Set<String>**	Specifies which users have access to the resources reserved by this reservation.

ReservationAccess

The access type of a standing reservation. If set to SHARED, allows a standing reservation to use resources already allocated to other non-job reservations. Otherwise, these other reservations block resource access.

Value	Description
DEDICATED	
SHARED	

AclRule

This class represents a rule that can be in Moab's access control list (ACL) mechanism.

The basic AclRule information is the object's name and type. The type directly maps to an AclType value. The default mechanism Moab uses to check the ACL for a particular item is if the user or object coming in has ANY of the values in the ACL, then the user or object is given access. If no values match the user or object in question, the user or object is rejected access.

Field Name	Type	Description
affinity	**AclAffinity**	Reservation ACLs allow or deny access to reserved resources but they may also be configured to affect a job's affinity for a particular reservation. By default, jobs gravitate toward reservations through a mechanism known as positive affinity. This mechanism allows jobs to run on the most constrained resources leaving other, unreserved resources free for use by other jobs that may not be able to access the reserved resources. Normally this is a desired behavior. However, sometimes, it is desirable to reserve resources for use only as a last resort-using the reserved resources only when there are no other resources available. This last resort behavior is known as negative affinity. Defaults to AclAffinity.POSITIVE.
comparator	**ComparisonOperator**	The type of comparison to make against the ACL object. Defaults to ComparisonOperator.EQUAL.
type	**AclType**	The type of the object that is being granted (or denied) access.
value	**String**	The name of the object that is being granted (or denied) access.

AclAffinity

This enumeration describes the values available for describing how a rule is used in establishing access to an object in Moab. Currently, these ACL affinities are used only for granting access to reservations.

Value	Description
NEGATIVE	Access to the object is repelled using this rule until access is the last choice.
NEUTRAL	Access to the object is not affected by affinity.
POSITIVE	Access to the object is looked at as the first choice.
PREEMPTIBLE	Access to the object given the rule gives preemptible status to the accessor. Supported only during GET.

Value	Description
REQUIRED	The rule in question must be satisified in order to gain access to the object. Supported only during GET.
UNAVAILABLE	The rule does not have its affinity available. Supported only during GET.

ComparisonOperator

This enumeration is used when Moab needs to compare items. One such use is in Access Control Lists (ACLs).

Value	Description
GREATER_THAN	Valid values: ">", "gt"
GREATER_THAN_OR_EQUAL	Valid values: ">=", "ge"
LESS_THAN	Valid values: "<", "lt"
LESS_THAN_OR_EQUAL	Valid values: "<=", "le"
EQUAL	Valid values: "==", "eq", "="
NOT_EQUAL	Valid values: "!=", "ne", "<>"
LEXIGRAPHIC_SUBSTRING	Valid value: "%<"
LEXIGRAPHIC_NOT_EQUAL	Valid value: "%!"
LEXIGRAPHIC_EQUAL	Valid value: "%="

AclType

This enumeration describes the values available for the type of an ACL Rule.

Value	Description
USER	User

Value	Description
GROUP	Group
ACCOUNT	Account or Project
CLASS	Class or Queue
QOS	Quality of Service
CLUSTER	Cluster
JOB_ID	Job ID
RESERVATION_ID	Reservation ID
JOB_TEMPLATE	Job Template
JOB_ATTRIBUTE	Job Attribute
DURATION	Duration in Seconds
PROCESSOR_SECONDS	Processor Seconds
JPRIORITY	Not supported
MEMORY	Not supported
NODE	Not supported
PAR	Not supported
PROC	Not supported
QTIME	Not supported
QUEUE	Not supported
RACK	Not supported

Value	Description
SCHED	Not supported
SYTEM	Not supported
TASK	Not supported
VC	Not supported
XFACTOR	Not supported

ReservationFlag

The flag types of a reservation.

Value	Description
ALLOWJOBOVERLAP	Allows jobs to overlap this Reservation, but not start during it (unless they have ACL access).
APPLYPROFRESOURCES	Only apply resource allocation info from profile.
DEADLINE	Reservation should be scheduled against a deadline.
IGNIDLEJOBS	Ignore idle job reservations.
IGNJOBRSV	Ignore job reservations, but not user or other reservations.
CHARGE	Charge the idle cycles in the accounting manager.
NOVMMIGRATIONS	Override the VM Migration Policy and don't migrate VMs that overlap this reservation.
OWNERPREEMPTIGNOREMINTIME	Owner ignores preemptmintime for this reservation.
PROVISION	Reservation should be capable of provisioning.
NOACLOVERLAP	Reservation will not look at ACLs to overlap job (when using exclusive).

Value	Description
ADVRES	If set, the reservation is created in advance of needing it.
ADVRESJOBDESTROY	Cancel any jobs associated with the reservation when it is released.
ALLOWGRID	The reservation is set up for use in a grid environment.
ALLOWPRSV	Personal reservations can be created within the space of this standing reservation (and ONLY this standing reservation). By default, when a standing reservation is given the flag ALLOWPRSV, it is given the ACL rule USER==ALL+ allowing all jobs and all users access.
BYNAME	Reservation only allows access to jobs that meet reservation ACLs and explicitly request the resources of this reservation using the job ADVRES flag.
DEDICATEDNODE	If set, only one active reservation is allowed on a node.
DEDICATEDRESOURCE	The reservation is only placed on resources that are not reserved by any other reservation, including jobs and other reservations.
EXCLUDEJOBS	Makes a reservation job exclusive, where only one job can run in the reservation.
ENDTRIGHASFIRED	A trigger has finished firing.
ENFORCENODESET	Enforce node sets when creating reservation.
EXCLUDEALLBUTSB	Reservation only shares resources with sandboxes.
EXCLUDEMYGROUP	Exclude reservations within the same group.
IGNRSV	Forces the reservation onto nodes regardless of whether there are other reservations currently residing on the nodes.
IGNSTATE	Request ignores existing resource reservations, allowing the reservation to be forced onto available resources even if this conflicts with other reservations.

Value	Description
ISACTIVE	If set, the reservation is currently active.
ISCLOSED	If set, the reservation is closed.
ISGLOBAL	If set the reservation applies to all resources.
OWNERPREEMPT	The owner of the reservation is given preemptor status for resources contained in the reservation.
PARENTLOCK	The reservation can only be destroyed by destroying its parent.
PREEMPTEE	The reservation is preemptible.
PLACEHOLDER	The reservation is a placeholder for resources.
PRSV	The reservation is a non-adminstrator, non-standing reservation, user-created reservation.
REQFULL	The reservation will fail if all resources requested cannot be allocated.
SCHEDULEVCRSV	The reservation was created as part of a schedule VC command. This pertains to reservations creating while scheduling MWS Services, and these are filtered from the MWS output of reservations.
SINGLEUSE	The reservation is automatically removed after completion of the first job to use the reserved resources.
SPACEFLEX	The reservation is allowed to adjust resources allocated over time in an attempt to optimize resource utilization.
STANDINGRSV	If set, the reservation was created by a standing reservation instance.
STATIC	Makes a reservation ineligible to modified or canceled by an administrator.
SYSTEMJOB	The reservation was created by a system job.
TIMEFLEX	The reservation is allowed to adjust the reserved time frame in an attempt to optimize resource utilization.

Value	Description
TRIGHASFIRED	The reservation has one or more triggers that have fired on it.
WASACTIVE	The reservation was previously active.
EVACVMS	Evacuate virtual machines on the node when the reservation starts.

JobFlag

This enumeration specifies the flag types of a job.

Value	Description
NONE	
BACKFILL	The job is using backfill to run.
COALLOC	The job can use resources from multiple resource managers and partitions.
ADVRES	The job requires the use of a reservation.
NOQUEUE	The job will attempt to execute immediately or fail.
ARRAYJOB	The job is part of a job array.
ARRAYJOBPARLOCK	This array job will only run in one partition.
ARRAYJOBPARSPAN	This array job will span partitions (default).
ARRAYMASTER	This job is the master of a job array.
BESTEFFORT	The job will succeed if even partial resources are available.
RESTARTABLE	The job is restartable.
SUSPENDABLE	The job is suspendable.
HASPREEMPTED	This job preempted other jobs to start.

Value	Description
PREEMPTEE	The job is a preemptee and therefore can be preempted by other jobs.
PREEMPTOR	The job is a preemptor and therefore can preempt other jobs.
RSVMAP	The job is based on a reservation.
SPVIOLATION	The job was started with a soft policy violation.
IGNNODEPOLICIES	The job will ignore node policies.
IGNPOLICIES	The job will ignore idle, active, class, partition, and system policies.
IGNNODESTATE	The job will ignore node state in order to run.
IGNIDLEJOBRSV	The job can ignore idle job reservations. The job granted access to all idle job reservations.
INTERACTIVE	The job needs to interactive input from the user to run.
FSVIOLATION	The job was started with a fairshare violation.
GLOBALQUEUE	The job is directly submitted without doing any authentication.
NORESOURCES	The job is a system job that does not need any resources.
NORMSTART	The job will not query a resource manager to run.
CLUSTERLOCKED	The job is locked into the current cluster and cannot be migrated elsewhere. This is for grid mode.
FRAGMENT	The job can be run across multiple nodes in individual chunks.
SYSTEMJOB	The job is a system job which simply runs on the same node that Moab is running on. This is usually used for running scripts and other executables in workflows.
ADMINSETIGNPOLICIES	The IGNPOLICIES flag was set by an administrator.

Value	Description
EXTENDSTARTWALLTIME	The job duration (walltime) was extended at job start.
SHAREDMEM	The job will share its memory across nodes.
BLOCKEDBYGRES	The job's generic resource requirement caused the job to start later.
GRESONLY	The job is requesting only generic resources, no compute resources.
TEMPLATESAPPLIED	The job has had all applicable templates applied to it.
META	META job, just a container around resources.
WIDERSVSEARCHALGO	This job prefers the wide search algorithm.
VMTRACKING	The job is a VMTracking job for an externally-created VM (via job template).
DESTROYTEMPLATESUBMITTED	A destroy job has already been created from the template for this job.
PROCSPECIFIED	The job requested processors on the command line.
CANCELONFIRSTFAILURE	Cancel job array on first array job failure.
CANCELONFIRSTSUCCESS	Cancel job array on first array job success.
CANCELONANYFAILURE	Cancel job array on any array job failure.
CANCELONANYSUCCESS	Cancel job array on any array job success.
CANCELONEXITCODE	Cancel job array on a specific exit code.
NOVMMIGRATE	Do not migrate the virtual machine that this job sets up.
PURGEONSUCCESSONLY	Only purge the job if it completed successfully

EmbeddedCredential

Field Name	Type	Description
name	String	
type	CredentialType	

CredentialType

Value	Description
USER	
GROUP	
ACCOUNT	
CLASS	
QOS	
NOT_SPECIFIED	

TimeWindow

This enumeration represents some common time windows. It can be used when for many purposes, but was created specifically for statistics.

Value	Description
MINUTE	
HOUR	
DAY	
WEEK	
MONTH	

Value	Description
YEAR	
INFINITY	

IntLimit

Field Name	Type	Description
qualifier	**String**	One of: • < • <= • == • >= • >
value	**Integer**	

Reservation

A reservation is the mechanism by which Moab guarantees the availability of a set of resources at a particular time. Each reservation consists of three major components: (1) a set of resources, (2) a time frame, and (3) an access control list. It is a scheduler role to ensure that the access control list is not violated during the reservation's lifetime (that is, its time frame) on the resources listed. For example, a reservation may specify that node002 is reserved for user Tom on Friday. The scheduler is thus constrained to make certain that only Tom's jobs can use node002 at any time on Friday.

Field Name	Type	Description
id	**String**	The unique ID of the reservation.
accountingAccount	**String**	Accountable Account.
accountingGroup	**String**	Accountable Group.

Field Name	Type	Description
accountingQOS	**String**	Accountable QOS.
accountingUser	**String**	Accountable User.
aclRules	**Set<AclRule>**	The set of access control rules associated with this reservation.
allocatedNodeCount	**Integer**	The number of allocated nodes for this reservation.
allocatedNodes	**Set<DomainProxyVersion1>**	The nodes allocated to the reservation.
allocatedProcessorCount	**Integer**	The number of allocated processors.
allocatedTaskCount	**Integer**	The number of allocated tasks.
comments	**String**	Reservation's comments or description.
creationDate	**Date**	Creation date. Automatically set by Moab when a user creates the reservation.
duration	**Long**	The duration of the reservation (in seconds).
endDate	**Date**	The end date of the reservation. This is especially useful for one-time reservations, which have an exact time for when a reservation ends.
excludeJobs	**Set<String>**	The list of jobs to exclude. Client must also set the IGNJOBRSV reservation flag. Otherwise, results are undefined. Used only during reservation creation.
expireDate	**Date**	The date/time when the reservation expires and vacates.
flags	**Set<ReservationFlag>**	The flags associated with the reservation.
globalId	**String**	Global reservation ID.

Field Name	Type	Description
hostListExpression	**String**	The list of nodes a user can select to reserve. This may or may not be the nodes that are currently allocated to this reservation. Note: Either hostListExpression or taskCount must be set to create a reservation.
idPrefix	**String**	The user-specified prefix for this reservation. If provided, Moab combines the idPrefix with an integer, and the combination is the unique identifier for this reservation.
isActive	**Boolean**	State whether or not this reservation is currently active.
isTracked	**Boolean**	States whether reservation resource usage is tracked.
label	**String**	When a label is assigned to a reservation, the reservation can then be referenced by that label as well as by the reservation name.
maxTasks	**Integer**	The maximum number of tasks for this reservation.
messages	**Set<MessageVersion1>**	Messages for the reservation.
owner	**EmbeddedCredential**	The owner of the reservation
partitionId	**String**	The ID of the partition this reservation is for.
profile	**String**	The profile that this reservation is using. A profile is a specification of attributes that all reservations share. Used only during reservation creation.
requirements	**ReservationRequirement**	The reservation's requirements.
reservationGroup	**String**	The reservation group to which the reservation belongs.

Field Name	Type	Description
resources	**Map<String, Integer>**	The reservation's resources. This field is a map, where the key is PROCS, MEM DISK, SWAP, or one or more user-defined keys.
startDate	**Date**	The start time for the reservation. This is especially useful for one-time reservations, which have an exact time for when a reservation starts.
statistics	**ReservationStatistics**	The reservation's statistical information.
subType	**String**	The reservation sub-type.
taskCount	**Integer**	The number of tasks that must be allocated to satisfy the reservation request. Note: Either hostListExpression or taskCount must be set to create a reservation.
trigger	**Trigger**	Trigger for reservation. Used only during reservation creation.
triggerIds	**Set<String>**	The IDs of the triggers attached to this reservation.
uniqueIndex	**String**	The globally-unique reservation index.
variables	**Map<String, Map>**	The set of variables for this reservation.

DomainProxyVersion1

Field Name	Type	Description
id	**String**	The id of the object.

MessageVersion1

Field Name	Type	Description
author	**String**	The author of the message.
creationTime	**Date**	The time the message was created in epoch time.
expireTime	**Date**	The time the message will be deleted in epoch time.
index	**Integer**	The index of the message relative to other messages in Moab's memory.
message	**String**	The comment information itself.
messageCount	**Integer**	The number of times this message has been displayed.
priority	**Double**	An optional priority that can be attached to the comment.

ReservationRequirement

Represents all the types of requirements a user can request while creating a reservation.

Field Name	Type	Description
architecture	**String**	Required architecture.
featureList	**Set<String>**	The list of features required for this reservation.
featureMode	**String**	Required feature mode.
memory	**Integer**	Required node memory, in MB.
nodeCount	**Integer**	Required number of nodes.
nodeIds	**Set<String>**	The list of node IDs required for this reservation.
os	**String**	Required Operating System.
taskCount	**Integer**	Required task count.

ReservationStatistics

Represents some basic statistical information that is kept about the usage of reservations. All metrics that are kept track relate to processor-seconds usage.

Field Name	Type	Description
caps	**Double**	The current active processor-seconds in the last reported iteration.
cips	**Double**	The current idle processor-seconds in the last reported iteration.
taps	**Double**	The total active processor-seconds over the life of the reservation.
tips	**Double**	The total idle processor-seconds over the life of the reservation.

Trigger

Field Name	Type	Description
id	**String**	Trigger id - internal ID used by moab to track triggers
action	**String**	For exec atype triggers, signifies executable and arguments. For jobpreempt atype triggers, signifies PREEMPTPOLICY to apply to jobs that are running on allocated resources. For changeparam atype triggers, specifies the parameter to change and its new value (using the same syntax and behavior as the changeparam command).
actionType	**TriggerActionType**	
blockTime	**Date**	Time (in seconds) Moab will suspend normal operation to wait for trigger execution to finish. Use caution as Moab will completely stop normal operation until BlockTime expires.
description	**String**	
eventType	**TriggerEventType**	
expireTime	**Date**	Time at which trigger should be terminated if it has not already been activated.

Field Name	Type	Description
failOffset	**Date**	Specifies the time (in seconds) that the threshold condition must exist before the trigger fires.
flags	[Set<TriggerFlag>]()	
interval	**Boolean**	When used in conjunction with MultiFire and RearmTime trigger will fire at regular intervals. Can be used with TriggerEventType.EPOCH to create a Standing Trigger. Defaults to false
maxRetry	**Integer**	Specifies the number of times Action will be attempted before the trigger is designated a failure.
multiFire	**Boolean**	Specifies whether this trigger can fire multiple times. Defaults to false.
name	**String**	Trigger name - can be auto assigned by moab or requested. Alphanumeric up to 16 characters in length
objectId	**String**	The ID of the object which this is attached to.
objectType	**String**	The type of object which this is attached to. Possible values: • vm - Virtual Machine
offset	**Date**	Relative time offset from event when trigger can fire.
period	**TriggerPeriod**	Can be used in conjunction with Offset to have a trigger fire at the beginning of the specified period. Can be used with EType epoch to create a standing trigger.
rearmTime	**Date**	Time between MultiFire triggers; rearm time is enforced from the trigger event time.
requires	**String**	Variables this trigger requires to be set or not set before it will fire. Preceding the string with an exclamation mark (!) indicates this variable must NOT be set. Used in conjunction with Sets to create trigger dependencies.

Field Name	Type	Description
sets	**String**	Variable values this trigger sets upon success or failure. Preceding the string with an exclamation mark (!) indicates this variable is set upon trigger failure. Preceding the string with a caret (^) indicates this variable is to be exported to the parent object when the current object is destroyed through a completion event. Used in conjunction with Requires to create trigger dependencies.
threshold	**String**	Reservation usage threshold - When reservation usage drops below Threshold, trigger will fire. Threshold usage support is only enabled for reservations and applies to percent processor utilization. gmetric thresholds are supported with job, node, credential, and reservation triggers. See Threshold Triggers in the Moab Workload Manager documentation for more information.
timeout	**Date**	Time allotted to this trigger before it is marked as unsuccessful and its process (if any) killed.
unsets	**String**	Variable this trigger destroys upon success or failure.

TriggerActionType

This enumeration specifies the action type of a trigger.

Value	Description
CANCEL	Only apply to reservation triggers
CHANGE_PARAM	
JOB_PREEMPT	This indicates that the trigger should preempt all jobs currently allocating resources assigned to the trigger's parent object. Only apply to reservation triggers.
MAIL	
THRESHOLD	
INTERNAL	

Value	Description
EXEC	

TriggerEventType

This enumeration specifies the event type of a trigger.

Value	Description
CANCEL	
CHECKPOINT	
CREATE	
END	
EPOCH	
FAIL	
HOLD	
MIGRATE	
MODIFY	
PREEMPT	
STANDING	
START	
THRESHOLD	

TriggerFlag

This enumeration specifies a flag belonging to a trigger.

Value	Description
ATTACH_ERROR	If the trigger outputs anything to stderr, Moab will attach this as a message to the trigger object.
CLEANUP	If the trigger is still running when the parent object completes or is canceled, the trigger will be killed.
CHECKPOINT	Moab should always checkpoint this trigger. See Checkpointing a Trigger in the Moab Workload Manager documentation for more information.
GLOBAL_VARS	The trigger will look in the name space of all nodes with the globalvars flag in addition to its own name space. A specific node to search can be specified using the following format: globalvars+node_id
INTERVAL	Trigger is periodic.
MULTIFIRE	Trigger can fire multiple times.
OBJECT_XML_STDIN	Trigger passes its parent's object XML information into the trigger's stdin. This only works for exec triggers with reservation type parents.
USER	The trigger will execute under the user ID of the object's owner. If the parent object is sched, the user to run under may be explicitly specified using the format user+<username>, for example flags=user+john:
GLOBAL_TRIGGER	The trigger will be (or was) inserted into the global trigger list.
ASYNCHRONOUS	An asynchronous trigger.
LEAVE_FILES	Do not remove stderr and stdout files.
PROBE	The trigger's stdout will be monitored.
PROBE_ALL	The trigger's stdout will be monitored.
GENERIC_SYSTEM_JOB	The trigger belongs to a generic system job (for checkpointing).

Value	Description
REMOVE_STD_FILES	The trigger will delete stdout/stderr files after it has been reset.
RESET_ON_MODIFY	The trigger resets if the object it is attached to is modified, even if multifire is not set.
SOFT_KILL	By default, a `SIGKILL` (kill -9) signal is sent to the kill the script when a trigger times out. This flag will instead send a `SIGTERM` (kill -15) signal to kill the script. The `SIGTERM` signal will allow the script to trap the signal so that the script can clean up any residual information on the system (instead of just dying, as with the `SIGKILL` signal). NOTE: A timed-out trigger will only receive one kill signal. This means that if you specify this flag, a timed-out trigger will only receive the `SIGTERM` signal, and never the `SIGKILL` signal.

TriggerPeriod

This enumeration specifies the period of a trigger.

Value	Description
MINUTE	
HOUR	
DAY	
WEEK	
MONTH	

Trigger

Field Name	Type	Description
id	**String**	Trigger id - internal ID used by moab to track triggers

Field Name	Type	Description
action	**String**	For exec atype triggers, signifies executable and arguments. For jobpreempt atype triggers, signifies PREEMPTPOLICY to apply to jobs that are running on allocated resources. For changeparam atype triggers, specifies the parameter to change and its new value (using the same syntax and behavior as the changeparam command).
actionType	**TriggerActionType**	
blockTime	**Date**	Time (in seconds) Moab will suspend normal operation to wait for trigger execution to finish. Use caution as Moab will completely stop normal operation until BlockTime expires.
description	**String**	
eventType	**TriggerEventType**	
expireTime	**Date**	Time at which trigger should be terminated if it has not already been activated.
failOffset	**Date**	Specifies the time (in seconds) that the threshold condition must exist before the trigger fires.
flags	**Set<TriggerFlag>**	
interval	**Boolean**	When used in conjunction with MultiFire and RearmTime trigger will fire at regular intervals. Can be used with TriggerEventType.EPOCH to create a Standing Trigger. Defaults to false
maxRetry	**Integer**	Specifies the number of times Action will be attempted before the trigger is designated a failure.
multiFire	**Boolean**	Specifies whether this trigger can fire multiple times. Defaults to false.
name	**String**	Trigger name - can be auto assigned by moab or requested. Alphanumeric up to 16 characters in length
objectId	**String**	The ID of the object which this is attached to.

Field Name	Type	Description
objectType	**String**	The type of object which this is attached to. Possible values: • vm - Virtual Machine
offset	**Date**	Relative time offset from event when trigger can fire.
period	**TriggerPeriod**	Can be used in conjunction with Offset to have a trigger fire at the beginning of the specified period. Can be used with EType epoch to create a standing trigger.
rearmTime	**Date**	Time between MultiFire triggers; rearm time is enforced from the trigger event time.
requires	**String**	Variables this trigger requires to be set or not set before it will fire. Preceding the string with an exclamation mark (!) indicates this variable must NOT be set. Used in conjunction with Sets to create trigger dependencies.
sets	**String**	Variable values this trigger sets upon success or failure. Preceding the string with an exclamation mark (!) indicates this variable is set upon trigger failure. Preceding the string with a caret (^) indicates this variable is to be exported to the parent object when the current object is destroyed through a completion event. Used in conjunction with Requires to create trigger dependencies.
threshold	**String**	Reservation usage threshold - When reservation usage drops below Threshold, trigger will fire. Threshold usage support is only enabled for reservations and applies to percent processor utilization. gmetric thresholds are supported with job, node, credential, and reservation triggers. See Threshold Triggers in the Moab Workload Manager documentation for more information.
timeout	**Date**	Time allotted to this trigger before it is marked as unsuccessful and its process (if any) killed.
unsets	**String**	Variable this trigger destroys upon success or failure.

Related topics

- **Standing reservations** on page 1639

Fields: User's Permissions

See the associated **Permissions** on page 1571 resource section for more information on how to use this resource and supported operations.

Additional references

Type	Value	Additional information
Permissions resource	`permissions/users`	**Permissions** on page 1571
Hooks filename	`permissions.users.groovy`	**Pre and post-processing hooks** on page 1412
Distinct query-supported	Yes	**Distinct** on page 1504

API version 3

UserPermission

Field Name	Type	Description
id	**String**	The unique ID of the cached user permission.
name	**String**	The unique name of the user.
permissions	**List<Permission>**	The list of permissions.

Permission

Represents a permission

Field Name	Type	Description
id	**String**	The unique ID of this role.
action	**String**	The action that can be performed on the resource.
description	**String**	A description of this permission.
fieldPath	**String**	Field level ACL control, if null or '*', all fields are accessible, otherwise requests must match dot delimited path. Currently only checked when doing writable actions. Example - attributes.*: create|update
label	**String**	A human readable label for this permission.
resource	**String**	The resource the permission applies to.
resourceFilter	**Map<String, Map>**	A map used to limit which resource instances this permission applies to. If this is null then the permission will apply to all instances of the resource. For api permissions the filter uses mongo query syntax.
scope	**PrivilegeScope**	Whether this permission applies to the principal's tenant-associated resources or globally
type	**String**	The type of the permission. Only 'api' type permissions are enforced.

PrivilegeScope

Some permissions and roles ignore tenants and apply globally. Others apply only to the resources associated with the principal's tenants.

Value	Description
GLOBAL	Describes a role or permission that applies globally, irrespective of the principal's tenants. This scope can be applied to any role or permission.
TENANT	Describes a role or permission that applies only to the resources associated with the principal's tenants. This scope can be applied to any role, but only to those permissions associated with tenanted resources (e.g. nodes, services, etc.).
NONE	Scope doesn't apply to some permissions. As of right now, all non-domain permissions (e.g. those created by Viewpoint) don't need a scope. NONE should therefore be assigned to all non-domain permissions.

API version 2

UserPermission

Field Name	Type	Description
id	**String**	The unique ID of the cached user permission.
name	**String**	The unique name of the user.
permissions	**List<Permission>**	The list of permissions.

Permission

Represents a permission

Field Name	Type	Description
id	**String**	The unique ID of this role.
action	**String**	The action that can be performed on the resource.
description	**String**	A description of this permission.
fieldPath	**String**	Field level ACL control, if null or '*', all fields are accessible, otherwise requests must match dot delimited path. Currently only checked when doing writable actions. Example - attributes.*: create|update
label	**String**	A human readable label for this permission.
resource	**String**	The resource the permission applies to.
resourceFilter	**Map<String, Map>**	A map used to limit which resource instances this permission applies to. If this is null then the permission will apply to all instances of the resource. For api permissions the filter uses mongo query syntax.
scope	**PrivilegeScope**	Whether this permission applies to the principal's tenant-associated resources or globally
type	**String**	The type of the permission. Only 'api' type permissions are enforced.

PrivilegeScope

Some permissions and roles ignore tenants and apply globally. Others apply only to the resources associated with the principal's tenants.

Value	Description
GLOBAL	Describes a role or permission that applies globally, irrespective of the principal's tenants. This scope can be applied to any role or permission.
TENANT	Describes a role or permission that applies only to the resources associated with the principal's tenants. This scope can be applied to any role, but only to those permissions associated with tenanted resources (e.g. nodes, services, etc.).
NONE	Scope doesn't apply to some permissions. As of right now, all non-domain permissions (e.g. those created by Viewpoint) don't need a scope. NONE should therefore be assigned to all non-domain permissions.

Related topics

- **Permissions** on page 1571

TORQUE Resource Manager

Introduction

This section contains some basic introduction information to help you get started using TORQUE. It contains these topics:

What is a Resource Manager?

While TORQUE has a built-in scheduler, pbs_sched, it is typically used solely as a *resource manager* with a scheduler making requests to it. Resources managers provide the low-level functionality to start, hold, cancel, and monitor jobs. Without these capabilities, a scheduler alone cannot control jobs.

What are Batch Systems?

While TORQUE is flexible enough to handle scheduling a conference room, it is primarily used in batch systems. Batch systems are a collection of computers and other resources (networks, storage systems, license servers, and so forth) that operate under the notion that the whole is greater than the sum of the parts. Some batch systems consist of just a handful of machines running single-processor jobs, minimally managed by the users themselves. Other systems have thousands and thousands of machines executing users' jobs simultaneously while tracking software licenses and access to hardware equipment and storage systems.

Pooling resources in a batch system typically reduces technical administration of resources while offering a uniform view to users. Once configured properly, batch systems abstract away many of the details involved with running and managing jobs, allowing higher resource utilization. For example, users typically only need to specify the minimal constraints of a job and do not need to know the individual machine names of each host on which they are running. With this uniform abstracted view, batch systems can execute thousands and thousands of jobs simultaneously.

Batch systems are comprised of four different components: (1) Master Node, (2) Submit/Interactive Nodes, (3) Compute Nodes, and (4) Resources.

Component	Description
Master Node	A batch system will have a master node where pbs_server runs. Depending on the needs of the systems, a master node may be dedicated to this task, or it may fulfill the roles of other components as well.
Submit/Interactive Nodes	Submit or interactive nodes provide an entry point to the system for users to manage their workload. For these nodes, users are able to submit and track their jobs. Additionally, some sites have one or more nodes reserved for interactive use, such as testing and troubleshooting environment problems. These nodes have client commands (such as qsub and qhold).
Computer Nodes	Compute nodes are the workhorses of the system. Their role is to execute submitted jobs. On each compute node, pbs_mom runs to start, kill, and manage submitted jobs. It communicates with pbs_server on the master node. Depending on the needs of the systems, a compute node may double as the master node (or more).
Resources	Some systems are organized for the express purpose of managing a collection of resources beyond compute nodes. Resources can include high-speed networks, storage systems, license managers, and so forth. Availability of these resources is limited and needs to be managed intelligently to promote fairness and increased utilization.

Basic Job Flow

The life cycle of a job can be divided into four stages: (1) creation, (2) submission, (3) execution, and (4) finalization.

Stage	Description
Creation	Typically, a submit script is written to hold all of the parameters of a job. These parameters could include how long a job should run (**walltime**), what resources are necessary to run, and what to execute. The following is an example submit file: `#PBS -N localBlast` `#PBS -S /bin/sh` `#PBS -l nodes=1:ppn=2,walltime=240:00:00` `#PBS -M user@my.organization.com` `#PBS -m ea` `source ~/.bashrc` `cd $HOME/work/dir` `sh myBlast.sh -i -v` This submit script specifies the name of the job (localBlast), what environment to use (`/bin/sh`), that it needs both processors on a single node (**nodes=1:ppn=2**), that it will run for at most 10 days, and that TORQUE should email "user@my.organization.com" when the job exits or aborts. Additionally, the user specifies where and what to execute.
Submission	A job is submitted with the qsub command. Once submitted, the policies set by the administration and technical staff of the site dictate the priority of the job and therefore, when it will start executing.
Execution	Jobs often spend most of their lifecycle executing. While a job is running, its status can be queried with qstat.
Finalilzation	When a job completes, by default, the `stdout` and `stderr` files are copied to the directory where the job was submitted.

Related topics

- Overview

Overview

This section contains some basic information about TORQUE, including how to install and configure it on your system. For details, see these topics:

- **TORQUE Installation Overview** on page 2193
- **Initializing/Configuring TORQUE on the Server (pbs_server)** on page 2200
- **Advanced configuration** on page 2207
- **Manual Setup of Initial Server Configuration** on page 2221
- **Server Node File Configuration** on page 2222
- **Testing Server Configuration** on page 2224

TORQUE Installation Overview

This section contains information about TORQUE architecture and explains how to install TORQUE. It also describes how to install tpackages on compute nodes and how to enable TORQUE as a service.

For details, see these topics:

Related topics

- **Troubleshooting** on page 2316

TORQUE Architecture

A TORQUE cluster consists of one head node and many compute nodes. The head node runs the pbs_server daemon and the compute nodes run the pbs_mom daemon. Client commands for submitting and managing jobs can be installed on any host (including hosts not running pbs_server or pbs_mom).

The head node also runs a scheduler daemon. The scheduler interacts with pbs_server to make local policy decisions for resource usage and allocate nodes to jobs. A simple FIFO scheduler, and code to construct more advanced schedulers, is provided in the TORQUE source distribution. Most TORQUE users choose to use a packaged, advanced scheduler such as Maui or Moab.

Users submit jobs to pbs_server using the qsub command. When pbs_server receives a new job, it informs the scheduler. When the scheduler finds nodes for the job, it sends instructions to run the job with the node list to pbs_server. Then, pbs_server sends the new job to the first node in the node list and instructs it to launch the job. This node is designated the execution host and is called *Mother Superior*. Other nodes in a job are called *sister MOMs*.

Related topics

- **TORQUE Installation Overview** on page 2193
- **Installing TORQUE** on page 2193

Installing TORQUE

These instructions describe how to install and start TORQUE.

Requirements

Supported Operating Systems

- CentOS 6.5 or later
- Red Hat 6.5 or later
- Scientific Linux 6.5 or later
- SUSE Linux Enterprise Server 11 SP3 or later

> CentOS 5.9, Red Hat 5.9 and Scientific Linux 5.9 are supported, largely to continue support for clusters where the compute nodes operating systems cannot be upgraded. We recommend that the TORQUE head node run on the supported operating systems listed above.

Software Requirements

- libxml2-devel package (package name may vary)
- openssl-devel package (package name may vary)
- Tcl/Tk version 8 or later if you plan to build the GUI portion of TORQUE or use a Tcl based scheduler
- If you use cpusets, libhwloc 1.1 or later is required (for TORQUE 4.0.0 and later)

If you build TORQUE from source (i.e. clone from github), the following additional software is required:

- gcc
- gcc-c++
- A posix compatible version of make
- libtool 1.5.22
- boost-devel 1.36.0

Prerequisites

- TORQUE requires certain ports to be open for essential communication:
  - For client communication to `pbs_server`, all privileged ports must be open (ports under 1024).
  - For `pbs_server` communication to `pbs_mom`, the default port is 15003.
  - For `pbs_mom` to `pbs_server`, the default port is 15001.

  For more information on how to configure the ports that TORQUE uses for communication, see **Configuring Ports** on page 2204.

  > **Important**: If you intend to use TORQUE 5.0.x with Moab, you must run Moab version 8.0.x or later. TORQUE 5.0.x will not work with versions earlier than Moab 8.0.x.

- Make sure your host (with the correct IP address) is in your `/etc/hosts` file.
- The `libxml2-devel`, `openssl-devel`, and `boost-devel` packages must be installed (These packages should already be installed from following the steps in the **Preparing for installation** on page 24).

  RHEL 6.5 and CentOS 6.5, and Scientific Linux 6.5:

  ```
  [root]# yum install openssl-devel libtool-devel libxml2-devel boost-devel gcc
  gcc-c++
  ```

  SLES

  ```
  [root]# zypper install openssl-devel libtool-devel libxml2-devel boost-devel
  gcc gcc-c++
  ```

  RHEL 5 and CentOS 5, and Scientific Linux 5:

  ```
  [root]# yum install openssl-devel libtool-devel libxml2-devel gcc gcc-c++
  ```

  > **Important**: TORQUE requires Boost version 1.36.0 or greater. The boost-devel package provided with RHEL 5, CentOS 5, and Scientific Linux 5 is older than this requirement. A new option, --with-boost-path has been added to configure (see **Customizing the Install** on page 2207 in the *TORQUE Administrator Guide* for more information). This allows you to point TORQUE to a specific version of boost during make. One way to compileTORQUE without installing Boost is to simply download the Boost version you plan to use from: http://www.boost.org/users/history/. Next, untar Boost—you do not need to build it or install it. When you run TORQUE configure, use the --with-boost-path option pointed to the extracted Boost directory.

To install TORQUE

1. Switch the user to root.

```
[user]$ su -
```

2. Download the latest 5.0.1 build from the Adaptive Computing website. It can also be downloaded via command line (github method or the tarball distribution).

   - **Clone the source from github:**

     If you clone the source from github, the libtool package must be installed.

     ```
     # RHEL 6 and Scientific Linux 6:
     [root]# yum install git libtool

     # SLES:
     [root]# zypper install libtool
     ```

     ```
     [root]# git clone https://github.com/adaptivecomputing/torque.git -b 5.0.1 5.0.1
     [root]# cd 5.0.1
     [root]# ./autogen.sh
     ```

     If you are using CentOS 5, use these instructions for installing `libtool`:

     ```
     [root]# cd /tmp
     [root]# wget http://ftpmirror.gnu.org/libtool/libtool-2.4.2.tar.gz
     [root]# tar -xzvf libtool-2.4.2.tar.gz
     [root]# cd libtool-2.4.2
     [root]# ./configure --prefix=/usr
     [root]# make
     [root]# make install
     [root]# cd /tmp
     [root]# git clone https://github.com/adaptivecomputing/torque.git -b
     5.0.1 5.0.1
     [root]# cd 5.0.1
     [root]# ./autogen.sh
     ```

   - **Get the tarball source distribution:**

     ```
     [root]# wget http://www.adaptivecomputing.com/download/torque/torque-
     5.0.1.tar.gz -O torque-5.0.1.tar.gz

     [root]# tar -xzvf torque-5.0.1.tar.gz
     [root]# cd torque-5.0.1/
     ```

3. Run each of the following commands in order.

   ```
   [root]# ./configure
   [root]# make
   [root]# make install
   ```

   For information on what options are available to customize the ./configure command, see **Customizing the Install** on page 2207.

4. Configure the `trqauthd` daemon to start automatically at system boot.

```
* If RHEL distribution, do the following *
[root]# cp contrib/init.d/trqauthd /etc/init.d/
[root]# chkconfig --add trqauthd
[root]# echo /usr/local/lib > /etc/ld.so.conf.d/torque.conf
[root]# ldconfig
[root]# service trqauthd start

* If SLES distribution, do the following *
[root]# cp contrib/init.d/suse.trqauthd /etc/init.d/trqauthd
[root]# chkconfig --add trqauthd
[root]# echo /usr/local/lib > /etc/ld.so.conf.d/torque.conf
[root]# ldconfig
[root]# service trqauthd start
```

5. The make packages command can be used to create self-extracting packages that can be copied and executed on your nodes. For information on creating packages and deploying them, see **Compute Nodes** on page 2198.

   You will also want to scp the `init.d` scripts to the compute nodes and install them there.

6. Verify that the `/var/spool/torque/server_name` file exists and contains the correct name of the server.

```
[root]# echo <pbs_server's_hostname> > /var/spool/torque/server_name
```

7. By default, TORQUE installs all binary files to `/usr/local/bin` and `/usr/local/sbin`. Make sure the path environment variable includes these directories for both the installation user and the root user.

```
[root]# export PATH=/usr/local/bin/:/usr/local/sbin/:$PATH
```

8. Initialize `serverdb` by executing the `torque.setup` script.

```
[root]# ./torque.setup root
```

9. Add nodes to the `/var/spool/torque/server_priv/nodes` file. For information on syntax and options for specifying compute nodes, see **Specifying Compute Nodes** on page 2202.

10. Configure the MOMs if necessary (see **Configuring TORQUE on Compute Nodes** on page 2203 in the TORQUE Administrator Guide).

11. Configure pbs_server and pbs_mom to start automatically at system boot, and then start their daemons.

```
* If RHEL distribution, do the following *
[root]# cp contrib/init.d/pbs_server contrib/init.d/pbs_mom /etc/init.d
[root]# chkconfig --add pbs_server
[root]# chkconfig --add pbs_mom
[root]# service pbs_server restart
[root]# service pbs_mom start

* If SLES distribution, do the following *
[root]# cp contrib/init.d/suse.pbs_server /etc/init.d/pbs_server
[root]# cp contrib/init.d/suse.pbs_mom /etc/init.d/pbs_mom
[root]# chkconfig --add pbs_server
[root]# chkconfig --add pbs_mom
[root]# service pbs_server restart
[root]# service pbs_mom start
```

Related topics

- **Preparing for installation** on page 24
- **Installing Moab Workload Manager** on page 34
- **Component documentation** on page 95

Compute Nodes

Use the Adaptive Computing tpackage system to create self-extracting tarballs which can be distributed and installed on compute nodes. The tpackages are customizable. See the `INSTALL` file for additional options and features.

> If you installed TORQUE using the RPMs, you must install and configure your nodes manually by modifying the `/var/spool/torque/mom_priv/config` file of each one. This file is identical for all compute nodes and can be created on the head node and distributed in parallel to all systems.

```
[root]# vi /var/spool/torque/mom_priv/config

$pbsserver     headnode     # hostname running pbs server
$logevent      225          # bitmap of which events to log

[root]# service pbs_mom restart
```

To create tpackages

1. Configure and make as normal, and then run `make packages`.

```
> make packages
Building ./torque-package-clients-linux-i686.sh ...
Building ./torque-package-mom-linux-i686.sh ...
Building ./torque-package-server-linux-i686.sh ...
Building ./torque-package-gui-linux-i686.sh ...
Building ./torque-package-devel-linux-i686.sh ...
Done.

The package files are self-extracting packages that can be copied and executed on
your production machines. Use --help for options.
```

2. Copy the desired packages to a shared location.

```
> cp torque-package-mom-linux-i686.sh /shared/storage/
> cp torque-package-clients-linux-i686.sh /shared/storage/
```

3. Install the tpackages on the compute nodes.

   Adaptive Computing recommends that you use a remote shell, such as SSH, to install tpackages on remote systems. Set up shared SSH keys if you do not want to supply a password for each host.

   > The only required package for the compute node is mom-linux. Additional packages are recommended so you can use client commands and submit jobs from compute nodes.

   The following is an example of how to copy and install mom-linux in a distributed fashion.

```
> for i in node01 node02 node03 node04 ; do scp torque-package-mom-linux-i686.sh
${i}:/tmp/. ; done
> for i in node01 node02 node03 node04 ; do scp torque-package-clients-linux-
i686.sh ${i}:/tmp/. ; done
> for i in node01 node02 node03 node04 ; do ssh ${i} /tmp/torque-package-mom-linux-
i686.sh --install ; done
> for i in node01 node02 node03 node04 ; do ssh ${i} /tmp/torque-package-clients-
linux-i686.sh --install ; done
```

Alternatively, you can use a tool like xCAT instead of dsh.

To use a tool like xCAT

1. Copy the tpackage to the nodes.

```
> prcp torque-package-linux-i686.sh noderange:/destinationdirectory/
```

2. Install the tpackage.

```
> psh noderange /tmp/torque-package-linux-i686.sh --install
```

Although optional, it is possible to use the TORQUE server as a compute node and install a pbs_mom with the pbs_server daemon.

Related topics

- **Installing TORQUE** on page 2193
- **TORQUE Installation Overview** on page 2193

Enabling TORQUE as a Service

> Enabling TORQUE as a service is optional. In order to run TORQUE as a service, you must enable `trqauthd`. (see **Configuring trqauthd for Client Commands** on page 2205).

The method for enabling TORQUE as a service is dependent on the Linux variant you are using. Startup scripts are provided in the `contrib/init.d/` directory of the source package. To enable TORQUE as a service, run the following on the host for the appropriate TORQUE daemon:

- RedHat (as root)

```
> cp contrib/init.d/pbs_mom /etc/init.d/pbs_mom
> chkconfig --add pbs_mom
> cp contrib/init.d/pbs_server /etc/init.d/pbs_server
> chkconfig --add pbs_server
```

- SuSE (as root)

```
> cp contrib/init.d/suse.pbs_mom /etc/init.d/pbs_mom
> insserv -d pbs_mom
> cp contrib/init.d/suse.pbs_server /etc/init.d/pbs_server
> insserv -d pbs_server
```

- Debian (as root)

```
> cp contrib/init.d/debian.pbs_mom /etc/init.d/pbs_mom
> update-rc.d pbs_mom defaults
> cp contrib/init.d/debian.pbs_server /etc/init.d/pbs_server
> update-rc.d pbs_server defaults
```

You will need to customize these scripts to match your system.

These options can be added to the self-extracting packages. For more details, see the `INSTALL` file.

Related topics

- **TORQUE Installation Overview** on page 2193
- **Installing TORQUE** on page 2193
- **Configuring trqauthd for Client Commands** on page 2205

Initializing/Configuring TORQUE on the Server (pbs_server)

The TORQUE server (pbs_server) contains all the information about a cluster. It knows about all of the MOM nodes in the cluster based on the information in the `$TORQUE_HOME/server_priv/nodes` file (See **Configuring TORQUE on Compute Nodes** on page 2203). It also maintains the status of each MOM node through updates from the MOMs in the cluster (see **pbsnodes** on page 2356). All jobs are submitted via qsub to the server, which maintains a master database of all jobs and their states.

Schedulers such as Moab Workload Manager receive job, queue, and node information from pbs_server and submit all jobs to be run to pbs_server.

The server configuration is maintained in a file named `serverdb`, located in `$TORQUE_HOME/server_priv`. The `serverdb` file contains all parameters pertaining to the operation of TORQUE plus all of the queues which are in the configuration. For pbs_server to run, `serverdb` must be initialized.

You can initialize `serverdb` in two different ways, but the recommended way is to use the `./torque.setup` script:

- As root, execute `./torque.setup` from the build directory (see **./torque.setup** on page 2201).
- Use pbs_server -t create (see **Initializing/Configuring TORQUE on the Server (pbs_server)** on page 2200).

Restart pbs_server after initializing `serverdb`.

```
> qterm
> pbs_server
```

./torque.setup

The `torque.setup` script uses `pbs_server -t` create to initialize `serverdb` and then adds a user as a manager and operator of TORQUE and other commonly used attributes. The syntax is as follows:

`/torque.setup` *username*

```
> ./torque.setup ken
> qmgr -c 'p s'

#
# Create queues and set their attributes.
#
#
# Create and define queue batch
#
create queue batch
set queue batch queue_type = Execution
set queue batch resources_default.nodes = 1
set queue batch resources_default.walltime = 01:00:00
set queue batch enabled = True
set queue batch started = True
#
# Set server attributes.
#
set server scheduling = True
set server acl_hosts = kmn
set server managers = ken@kmn
set server operators = ken@kmn
set server default_queue = batch
set server log_events = 511
set server mail_from = adm
set server node_check_rate = 150
set server tcp_timeout = 6
set server mom_job_sync = True
set server keep_completed = 300
```

pbs_server -t create

The `-t create` option instructs pbs_server to create the `serverdb` file and initialize it with a minimum configuration to run pbs_server.

```
> pbs_server -t create
```

To see the configuration and verify that TORQUE is configured correctly, use **qmgr**:

```
> qmgr -c 'p s'

#
# Set server attributes.
#
set server acl_hosts = kmn
set server log_events = 511
set server mail_from = adm
set server node_check_rate = 150
set server tcp_timeout = 6
```

A single queue named batch and a few needed server attributes are created.

This section contains these topics:

- **Specifying Compute Nodes** on page 2202
- **Configuring TORQUE on Compute Nodes** on page 2203
- **Finalizing Configurations** on page 2207

Related topics

- **Appendix C: Node Manager (MOM) Configuration** on page 2435
- **Advanced configuration** on page 2207

Specifying Compute Nodes

The environment variable *TORQUE_HOME* is where configuration files are stored. If you used the default locations during installation, you do not need to specify the *TORQUE_HOME* environment variable.

The pbs_server must recognize which systems on the network are its compute nodes. Specify each node on a line in the server's nodes file. This file is located at *TORQUE_HOME*/`server_priv/nodes`. In most cases, it is sufficient to specify just the names of the nodes on individual lines; however, various properties can be applied to each node.

> Only a root user can access the `server_priv` directory.

Syntax of nodes file:

```
node-name[:ts] [np=] [gpus=] [properties]
```

- The **node-name** must match the hostname on the node itself, including whether it is fully qualified or shortened.
- The **[:ts]** option marks the node as timeshared. Timeshared nodes are listed by the server in the node status report, but the server does not allocate jobs to them.
- The **[np=]** option specifies the number of virtual processors for a given node. The value can be less than, equal to, or greater than the number of physical processors on any given node.
- The **[gpus=]** option specifies the number of GPUs for a given node. The value can be less than, equal to, or greater than the number of physical GPUs on any given node.

- The node processor count can be automatically detected by the TORQUE server if **auto_node_np** is set to TRUE. This can be set using this command:

  `qmgr -c set server auto_node_np = True`

  Setting **auto_node_np** to TRUE overwrites the value of np set in `TORQUE_HOME/server_priv/nodes`.
- The **[properties]** option allows you to specify arbitrary strings to identify the node. Property strings are alphanumeric characters only and must begin with an alphabetic character.
- Comment lines are allowed in the nodes file if the first non-white space character is the pound sign (#).

The following example shows a possible node file listing.

`TORQUE_HOME/server_priv/nodes`:

```
# Nodes 001 and 003-005 are cluster nodes
#
node001 np=2 cluster01 rackNumber22
#
# node002 will be replaced soon
node002:ts waitingToBeReplaced
# node002 will be replaced soon
#
node003 np=4 cluster01 rackNumber24
node004 cluster01 rackNumber25
node005 np=2 cluster01 rackNumber26 RAM16GB
node006
node007 np=2
node008:ts np=4
...
```

Related topics

- **Initializing/Configuring TORQUE on the Server (pbs_server)** on page 2200

Configuring TORQUE on Compute Nodes

If using TORQUE self-extracting packages with default compute node configuration, no additional steps are required and you can skip this section.

If installing manually, or advanced compute node configuration is needed, edit the *TORQUE_HOME*`/mom_priv/config` file on each node. The recommended settings follow.

`TORQUE_HOME/mom_priv/config`:

```
$pbsserver     headnode       # hostname running pbs server
$logevent      225            # bitmap of which events to log
```

This file is identical for all compute nodes and can be created on the head node and distributed in parallel to all systems.

Related topics

- **Initializing/Configuring TORQUE on the Server (pbs_server)** on page 2200

Configuring Ports

You can optionally configure the various ports that TORQUE uses for communication. Most ports can be configured multiple ways. The ports you can configure are:

- pbs_server listening port
- pbs_mom listening port
- port pbs_server uses to communicate to the pbs_mom
- port pbs_mom uses to communicate to the pbs_server
- port client commands use to communicate to the pbs_server
- port trqauthd uses to communicate to the pbs_server

> If you are running pbspro on the same system, be aware that it uses the same environment variables and /etc/services entries.

Configuring the pbs_server listening port

To configure the port the pbs_server listens on, follow any of these steps:

- Set an environment variable called PBS_BATCH_SERVICE_PORT to the port desired.
- Edit the /etc/services file and set pbs port_num/tcp.
- Start pbs_server with the -p option.

```
$ pbs_server -p port_num
```

- Edit the $PBS_HOME/server_name file and change server_name to server_name:*<port_num>*
- Start pbs_server with the -H option.

```
$ pbs_server -H server_name:port_num
```

Configuring the pbs_mom listening port

To configure the port the pbs_mom listens on, follow any of these steps:

- Set an environment variable called PBS_MOM_SERVICE_PORT to the port desired.
- Edit the /etc/services file and set pbs_mom port_num/tcp.
- Start pbs_mom with the -M option.

```
$ pbs_mom -M port_num
```

- Edit the nodes file entry for that list: add mom_service_port=port_num.

Configuring the port pbs_server uses to communicate with pbs_mom

To configure the port the pbs_server uses to communicate with pbs_mom, follow any of these steps:

- Set an environment variable called `PBS_MOM_SERVICE_PORT` to the port desired.
- Edit the `/etc/services` file and set `pbs_mom port_num/tcp`.
- Start `pbs_mom` with the `-M` option.

```
$ pbs_server -M port_num
```

Configuring the port pbs_mom uses to communicate with pbs_server

To configure the port the `pbs_mom` uses to communicate with `pbs_server`, follow any of these steps:

- Set an environment variable called `PBS_BATCH_SERVICE_PORT` to the port desired.
- Edit the `/etc/services` file and set `pbs port_num/tcp`.
- Start `pbs_mom` with the `-S` option.

```
$ pbs_mom -p port_num
```

- Edit the `nodes` file entry for that list: add `mom_service_port=port_num`.

Configuring the port client commands use to communicate with pbs_server

To configure the port client commands use to communicate with `pbs_server`, follow any of these steps:

- Edit the `/etc/services` file and set `pbs port_num/tcp`.
- Edit the `$PBS_HOME/server_name` file and change `server_name` to `server_name:`*<port_num>*

Configuring the port trqauthd uses to communicate with pbs_server

To configure the port trqauthd uses to communicate with `pbs_server`, follow any of these steps:

- Edit the `$PBS_HOME/server_name` file and change `server_name` to `server_name:`*<port_num>*

Related topics

- **Initializing/Configuring TORQUE on the Server (pbs_server)** on page 2200
- pbs_server
- pbs_mom
- trqauthd
- client commands

Configuring trqauthd for Client Commands

trqauthd is a daemon used by TORQUE client utilities to authorize user connections to pbs_server. Once started, it remains resident. TORQUE client utilities then communicate with trqauthd on port 15005 on the loopback interface. It is multi-threaded and can handle large volumes of simultaneous requests.

Running trqauthd

trqauthd must be run as root. It must also be running on any host where TORQUE client commands will execute.

By default, trqauthd is installed to `/usr/local/bin`.

trqauthd can be invoked directly from the command line or by the use of init.d scripts which are located in the `contrib/init.d` directory of the TORQUE source.

There are three `init.d` scripts for trqauthd in the `contrib/init.d` directory of the TORQUE source tree:

Script	Description
debian.trqauthd	Used for apt-based systems (debian, ubuntu are the most common variations of this)
suse.trqauthd	Used for suse-based systems
trqauthd	An example for other package managers (Redhat, Scientific, CentOS, and Fedora are some common examples)

You should edit these scripts to be sure they will work for your site.

Inside each of the scripts are the variables PBS_DAEMON and PBS_HOME. These two variables should be updated to match your TORQUE installation. PBS_DAEMON needs to point to the location of trqauthd. PBS_HOME needs to match your TORQUE installation.

Choose the script that matches your dist system and copy it to `/etc/init.d`. If needed, rename it to **trqauthd**.

To start the daemon

```
/etc/init.d/trqauthd start
```

To stop the daemon

```
/etc/init.d/trqauthd stop
```

OR

```
service trqauthd start/stop
```

If you receive an error that says "Could not open socket in trq_simple_connect. error 97" and you use a CentOS, RedHat, or Scientific Linux 6+ operating system, check your `/etc/hosts` file for multiple entries of a single host name pointing to the same IP address. Delete the duplicate(s), save the file, and launch trqauthd again.

Related topics

- **Initializing/Configuring TORQUE on the Server (pbs_server)** on page 2200

Finalizing Configurations

After configuring the `serverdb` and the `server_priv/nodes` files, and after ensuring minimal MOM configuration, restart the pbs_server on the server node and the pbs_mom on the compute nodes.

Compute Nodes:

```
> pbs_mom
```

Server Node:

```
> qterm -t quick
> pbs_server
```

After waiting several seconds, the `pbsnodes -a` command should list all nodes in state `free`.

Related topics

- **Initializing/Configuring TORQUE on the Server (pbs_server)** on page 2200

Advanced configuration

This section contains information about how you can customize the installation and configure the server to ensure that the server and nodes are communicating correctly. For details, see these topics:

- **Customizing the Install** on page 2207
- **Server Configuration** on page 2215

Related topics

- **Appendix B: Server Parameters** on page 2417

Customizing the Install

The TORQUE configure command has several options available. Listed below are some suggested options to use when running `./configure`.

- By default, TORQUE does not install the admin manuals. To enable this, use `--enable-docs`.
- By default, only children MOM processes use syslog. To enable syslog for all of TORQUE, use `--enable-syslog`.

Table 4-2: Optional Features

Option	Description
--disable-clients	Directs TORQUE not to build and install the TORQUE client utilities such as qsub, qstat, qdel, etc.
--disable-FEATURE	Do not include FEATURE (same as --enable-FEATURE=no).
--disable-libtool-lock	Avoid locking (might break parallel builds).
--disable-mom	Do not include the MOM daemon.
--disable-mom-check-spool	Don't check free space on spool directory and set an error.
--disable-posixmemlock	Disable the MOM's use of mlockall. Some versions of OSs seem to have buggy POSIX MEMLOCK.
--disable-privports	Disable the use of privileged ports for authentication. Some versions of OSX have a buggy bind () and cannot bind to privileged ports.
--disable-qsub-keep-override	Do not allow the qsub -k flag to override -o -e.
--disable-server	Do not include server and scheduler.
--disable-shell-pipe	Give the job script file as standard input to the shell instead of passing its name via a pipe.
--disable-spool	If disabled, TORQUE will create output and error files directly in $HOME/.pbs_spool if it exists or in $HOME otherwise. By default, TORQUE will spool files in TORQUE_HOME/spool and copy them to the users home directory when the job completes.
--disable-xopen-networking	With HPUX and GCC, don't force usage of XOPEN and libxnet.
--enable-acct-x	Enable adding x attributes to accounting log.

Option	Description
--enable-array	Setting this under IRIX enables the SGI Origin 2000 parallel support. Normally autodetected from the /etc/config/array file.
--enable-blcr	Enable BLCR support.
--enable-cpa	Enable Cray's CPA support.
--enable-cpu-set	Enable Linux 2.6 kernel cpusets. It is recommended that you turn on this feature to prevent a job from expanding across more CPU cores than it is assigned.
--enable-debug	Prints debug information to the console for pbs_server and pbs_mom while they are running. (This is different than **--with-debug** which will compile with debugging symbols.)
--enable-dependency-tracking	Do not reject slow dependency extractors.
--enable-fast-install[=PKGS]	Optimize for fast installation [default=yes].
--enable-FEATURE [=ARG]	Include FEATURE [ARG=yes].
--enable-file-sync	Open files with sync on each write operation. This has a negative impact on TORQUE performance. This is disabled by default.
--enable-force-nodefile	Forces creation of nodefile regardless of job submission parameters. Not on by default.
--enable-gcc-warnings	Enable gcc strictness and warnings. If using gcc, default is to error on any warning.
--enable-geometry-requests	TORQUE is compiled to use procs_bitmap during job submission.
--enable-gui	Include the GUI-clients.

Option	Description
--enable-maintainer-mode	This is for the autoconf utility and tells autoconf to enable so called rebuild rules. See maintainer mode for more information.
--enable-maxdefault	Turn on the RESOURCEMAXDEFAULT flag. Versions of TORQUE earlier than 2.4.5 attempted to apply queue and server defaults to a job that didn't have defaults specified. If a setting still did not have a value after that, TORQUE applied the queue and server maximum values to a job (meaning, the maximum values for an applicable setting were applied to jobs that had no specified or default value). In TORQUE 2.4.5 and later, the queue and server maximum values are no longer used as a value for missing settings. To re-enable this behavior in TORQUE 2.4.5 and later, use `--enable-maxdefault`.
--enable-nochildsignal	Turn on the NO_SIGCHLD flag.
--enable-nodemask	Enable nodemask-based scheduling on the Origin 2000.
--enable-pemask	Enable pemask-based scheduling on the Cray T3e.
--enable-plock-daemons[=ARG]	Enable daemons to lock themselves into memory: logical-or of 1 for pbs_server, 2 for pbs_scheduler, 4 for pbs_mom (no argument means 7 for all three).
--enable-quick-commit	Turn on the QUICKCOMMIT flag.
--enable-shared[=PKGS]	Build shared libraries [default=yes].
--enable-shell-use-argv	Enable this to put the job script name on the command line that invokes the shell. Not on by default. Ignores --enable-shell-pipe setting.
--enable-sp2	Build PBS for an IBM SP2.
--enable-srfs	Enable support for SRFS on Cray.

Option	Description
--enable-static [=PKGS]	Build static libraries [default=yes].
--enable-sys-log	Enable (default) the use of syslog for error reporting.
--enable-tcl-qstat	Setting this builds qstat with Tcl interpreter features. This is enabled if Tcl is enabled.
--enable-unix-sockets	Enable the use of Unix Domain sockets for authentication.

Table 4-3: Optional packages

Option	Description
--with-blcr=DIR	BLCR installation prefix (Available in versions 2.5.6 and 3.0.2 and later).
--with-blcr-include=DIR	Include path for libcr.h (Available in versions 2.5.6 and 3.0.2 and later).
--with-blcr-lib=DIR	Lib path for libcr (Available in versions 2.5.6 and 3.0.2 and later).
--with-blcr-bin=DIR	Bin path for BLCR utilities (Available in versions 2.5.6 and 3.0.2 and later).

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>--with-boost-path=DIR</td><td>Set the path to the Boost header files to be used during make. This option does not require Boost to be built or installed.

The --with-boost-path value must be a directory containing a sub-directory called boost that contains the boost .hpp files.
For example, if downloading the boost 1.55.0 source tarball to the adaptive user's home directory:
<pre>[adaptive]$ cd ~
[adaptive]$ wget
http://sourceforge.net/projects/boost/files/boost/1.55.0/boost_1_55_
0.tar.gz/download
[adaptive]$ tar xzf boost_1_55_0.tar.gz
[adaptive]$ ls boost_1_55_0
boost
boost-build.jam
...</pre>In this case use --with-boost-path=/home/adaptive/boost_1_55_0 during configure.

Another example would be to use an installed version of Boost. If the installed Boost header files exist in /usr/include/boost/*.hpp, use --with-boost-path=/usr/include.</td></tr>
<tr><td>--with-cpa-include=DIR</td><td>Include path for cpalib.h.</td></tr>
<tr><td>--with-cpa-lib=DIR</td><td>Lib path for libcpalib.</td></tr>
<tr><td>--with-debug=no</td><td>Do not compile with debugging symbols.</td></tr>
<tr><td>--with-default-server-r=HOSTNAME</td><td>Set the name of the computer that clients will access when no machine name is specified as part of the queue name. It defaults to the hostname of the machine on which PBS is being compiled.</td></tr>
<tr><td>--with-environ=PATH</td><td>Set the path containing the environment variables for the daemons. For SP2 and AIX systems, suggested setting is to /etc/environment. Defaults to the file "pbs_environment" in the server-home. Relative paths are interpreted within the context of the server-home.</td></tr>
<tr><td>--with-gnu-ld</td><td>Assume the C compiler uses GNU ld [default=no].</td></tr>
<tr><td>--with-mail-domain=MAILDOMAIN</td><td>Override the default domain for outgoing mail messages, i.e. "user@maildomain". The default maildomain is the hostname where the job was submitted from.</td></tr>
</table>

Option	Description
--with-modulefiles[=DIR]	Use module files in specified directory [/etc/modulefiles].
--with-momlogdir	Use this directory for MOM logs.
--with-momlogsuffix	Use this suffix for MOM logs.
--without-PACKAGE	Do not use PACKAGE (same as --with-PACKAGE=no).
--without-readline	Do not include readline support (default: included if found).
--with-PACKAGE[=ARG]	Use PACKAGE [ARG=yes].
--with-pam=DIR	Directory that holds the system PAM modules. Defaults to /lib(64)/security on Linux.
--with-pic	Try to use only PIC/non-PIC objects [default=use both].
--with-qstatrc-file=FILE	Set the name of the file that qstat will use if there is no ".qstatrc" file in the directory where it is being invoked. Relative path names will be evaluated relative to the server home directory (see above). If this option is not specified, the default name for this file will be set to "qstatrc" (no dot) in the server home directory.
--with-rcp	One of "scp", "rcp", "mom_rcp", or the full path of a remote file copy program. scp is the default if found, otherwise mom_rcp is used. Some rcp programs don't always exit with valid error codes in case of failure. mom_rcp is a copy of BSD rcp included with this source that has correct error codes, but it is also old, unmaintained, and doesn't have large file support.
--with-sched=TYPE	Sets the scheduler type. If TYPE is "c", the scheduler will be written in C. If TYPE is "tcl" the server will use a Tcl based scheduler. If TYPE is "basl", TORQUE will use the rule based scheduler. If TYPE is "no", then no scheduling is done. "c" is the default.
--with-sched-code=PATH	Sets the name of the scheduler to use. This only applies to BASL schedulers and those written in the C language. For C schedulers this should be a directory name and for BASL schedulers a filename ending in ".basl". It will be interpreted relative to srctree/src/schedulers.SCHD_TYPE/samples. As an example, an appropriate BASL scheduler relative path would be "nas.basl". The default scheduler code for "C" schedulers is "fifo".

Option	Description
--with-scp	In TORQUE 2.1 and later, SCP is the default remote copy protocol. See **--with-rcp** if a different protocol is desired.
--with-sendmail[=FILE]	Sendmail executable to use.
--with-server-home=DIR	Set the server home/spool directory for PBS use. Defaults to /var/spool/torque.
--with-server-name-file-e=FILE	Set the file that will contain the name of the default server for clients to use. If this is not an absolute pathname, it will be evaluated relative to the server home directory that either defaults to /usr/spool/PBS or is set using the --with-server-home option to configure. If this option is not specified, the default name for this file will be set to "server_name".
--with-tcl	Directory containing tcl configuration (tclConfig.sh).
--with-tclatrsep=CHAR	Set the Tcl attribute separator character this will default to "." if unspecified.
--with-tclinclude	Directory containing the public Tcl header files.
--with-tclx	Directory containing tclx configuration (tclxConfig.sh).
--with-tk	Directory containing tk configuration (tkConfig.sh).
--with-tkinclude	Directory containing the public Tk header files.
--with-tkx	Directory containing tkx configuration (tkxConfig.sh).
--with-tmpdir=DIR	Set the tmp directory that pbs_mom will use. Defaults to "/tmp". This is a Cray-specific feature.
--with-xauth=PATH	Specify path to xauth program.

HAVE_WORDEXP

`Wordxp()` performs a shell-like expansion, including environment variables. By default, `HAVE_WORDEXP` is set to **1** in `src/pbs_config.h`. If set to 1, will limit the characters that can be used in a job name to those allowed for a file in the current environment, such as BASH. If set to 0, any valid character for the file system can be used.

If a user would like to disable this feature by setting `HAVE_WORDEXP` to 0 in `src/include/pbs_config.h`, it is important to note that the error and the output file names will not expand environment variables, including $PBS_JOBID. The other important consideration is that characters that BASH dislikes, such as (), will not be allowed in the output and error file names for jobs by default.

Related topics

- **Advanced configuration** on page 2207
- **Server Configuration** on page 2215

Server Configuration

See these topics for details:

- **Server configuration overview** on page 2215
- **Name service configuration** on page 2215
- **Configuring job submission hosts** on page 2216
- **Configuring TORQUE on a multi-homed server** on page 2217
- **Architecture specific notes** on page 2217
- **Specifying non-root administrators** on page 2217
- **Setting up email** on page 2217
- **Using MUNGE authentication** on page 2218
- **Server Configuration** on page 2215

- **MOM Hierarchy** (optional)

Server configuration overview

There are several steps to ensure that the server and the nodes are completely aware of each other and able to communicate directly. Some of this configuration takes place within TORQUE directly using the qmgr command. Other configuration settings are managed using the pbs_server nodes file, DNS files such as /etc/hosts and the /etc/hosts.equiv file.

Name service configuration

Each node, as well as the server, must be able to resolve the name of every node with which it will interact. This can be accomplished using /etc/hosts, DNS, NIS, or other mechanisms. In the case of /etc/hosts, the file can be shared across systems in most cases.

A simple method of checking proper name service configuration is to verify that the server and the nodes can "ping" each other.

Configuring job submission hosts

Using RCmd authentication

When jobs can be submitted from several different hosts, these hosts should be trusted via the R* commands (such as rsh and rcp). This can be enabled by adding the hosts to the /etc/hosts.equiv file of the machine executing the pbs_server daemon or using other R* command authorization methods. The exact specification can vary from OS to OS (see the man page for **ruserok** to find out how your OS validates remote users). In most cases, configuring this file is as simple as adding a line to your `/etc/hosts.equiv` file, as in the following:

`/etc/hosts.equiv`:

```
#[+ | -] [hostname] [username]
mynode.myorganization.com
.....
```

Either of the hostname or username fields may be replaced with a wildcard symbol (+). The (+) may be used as a stand-alone wildcard but not connected to a username or hostname, e.g., `+node01` or `+user01`. However, a (-) may be used in that manner to specifically exclude a user.

> Following the Linux man page instructions for hosts.equiv may result in a failure. You cannot precede the user or hostname with a (+). To clarify, `node1 +user1` will not work and **user1** will not be able to submit jobs.

For example, the following lines will not work or will not have the desired effect:

```
+node02 user1
node02 +user1
```

These lines will work:

```
node03 +
+ jsmith
node04 -tjones
```

The most restrictive rules must precede more permissive rules. For example, to restrict user tsmith but allow all others, follow this format:

```
node01 -tsmith
node01 +
```

Please note that when a hostname is specified, it must be the fully qualified domain name (FQDN) of the host. Job submission can be further secured using the server or queue **acl_hosts** and **acl_host_enabled** parameters (for details, see **Queue Attributes** on page 2277).

Using the "submit_hosts" service parameter

Trusted submit host access may be directly specified without using RCmd authentication by setting the server submit_hosts parameter via **qmgr** as in the following example:

```
> qmgr -c 'set server submit_hosts = host1'
> qmgr -c 'set server submit_hosts += host2'
> qmgr -c 'set server submit_hosts += host3'
```

> Use of **submit_hosts** is potentially subject to DNS spoofing and should not be used outside of controlled and trusted environments.

Allowing job submission from compute hosts

If preferred, all compute nodes can be enabled as job submit hosts without setting `.rhosts` or `hosts.equiv` by setting the allow_node_submit parameter to **true**.

Configuring TORQUE on a multi-homed server

If the pbs_server daemon is to be run on a multi-homed host (a host possessing multiple network interfaces), the interface to be used can be explicitly set using the SERVERHOST parameter.

Architecture specific notes

With some versions of Mac OS/X, it is required to add the line `$restricted *.<DOMAIN>` to the pbs_mom configuration file. This is required to work around some socket bind bugs in the OS.

Specifying non-root administrators

By default, only root is allowed to start, configure and manage the pbs_server daemon. Additional trusted users can be authorized using the parameters **managers** and **operators**. To configure these parameters use the **qmgr** command, as in the following example:

```
> qmgr
Qmgr: set server managers += josh@*.fsc.com
Qmgr: set server operators += josh@*.fsc.com
```

All manager and operator specifications must include a user name and either a fully qualified domain name or a host expression.

> To enable all users to be trusted as both operators and administrators, place the **+** (plus) character on its own line in the `server_priv/acl_svr/operators` and `server_priv/acl_svr/managers` files.

Setting up email

Moab relies on emails from TORQUE about job events. To set up email, do the following:

To set up email

1. Use the `--with-sendmail` configure option at configure time. TORQUE needs to know where the email application is. If this option is not used, TORQUE tries to find the sendmail executable. If it isn't found, TORQUE cannot send emails.

   ```
   > ./configure --with-sendmail=<path_to_executable>
   ```

2. Set `mail_domain` in your server settings. If your domain is clusterresources.com, execute:

```
> qmgr -c 'set server mail_domain=clusterresources.com'
```

3. (Optional) You can override the default mail_body_fmt and mail_subject_fmt values via **qmgr**:

```
> qmgr -c 'set server mail_body_fmt=Job: %i \n Name: %j \n On host: %h \n \n %m \n
\n %d'
> qmgr -c 'set server mail_subject_fmt=Job %i - %r'
```

By default, users receive e-mails on job aborts. Each user can select which kind of e-mails to receive by using the qsub -m option when submitting the job. If you want to dictate when each user should receive e-mails, use a submit filter (for details, see **Appendix J: Job Submission Filter ("qsub wrapper")** on page 2479).

Using MUNGE authentication

MUNGE is an authentication service that creates and validates user credentials. It was developed by Lawrence Livermore National Laboratory (LLNL) to be highly scalable so it can be used in large environments such as HPC clusters. To learn more about MUNGE and how to install it, see http://code.google.com/p/munge/.

Configuring TORQUE to use MUNGE is a compile time operation. When you are building TORQUE, use –enable-munge-auth as a command line option with ./configure.

```
> ./configure -enable-munge-auth
```

You can use only one authorization method at a time. If –enable-munge-auth is configured, the privileged port ruserok method is disabled.

TORQUE does not link any part of the MUNGE library into its executables. It calls the MUNGE and UNMUNGE utilities which are part of the MUNGE daemon. The MUNGE daemon must be running on the server and all submission hosts. The TORQUE client utilities call MUNGE and then deliver the encrypted credential to pbs_server where the credential is then unmunged and the server verifies the user and host against the authorized users configured in serverdb.

Authorized users are added to serverdb using qmgr and the **authorized_users** parameter. The syntax for **authorized_users** is authorized_users=*<user>*@*<host>*. To add an authorized user to the server you can use the following qmgr command:

```
> qmgr -c 'set server authorized_users=user1@hosta
> qmgr -c 'set server authorized_users+=user2@hosta
```

The previous example adds user1 and user2 from hosta to the list of authorized users on the server. Users can be removed from the list of authorized users by using the **-=** syntax as follows:

```
> qmgr -c 'set server authorized_users-=user1@hosta
```

Users must be added with the <user>@<host> syntax. The user and the host portion can use the '*' wildcard to allow multiple names to be accepted with a single entry. A range of user or host names can be specified using a [a-b] syntax where *a* is the beginning of the range and *b* is the end.

```
> qmgr -c 'set server authorized_users=user[1-10]@hosta
```

This allows user1 through user10 on hosta to run client commands on the server.

Related topics

- **Advanced configuration** on page 2207

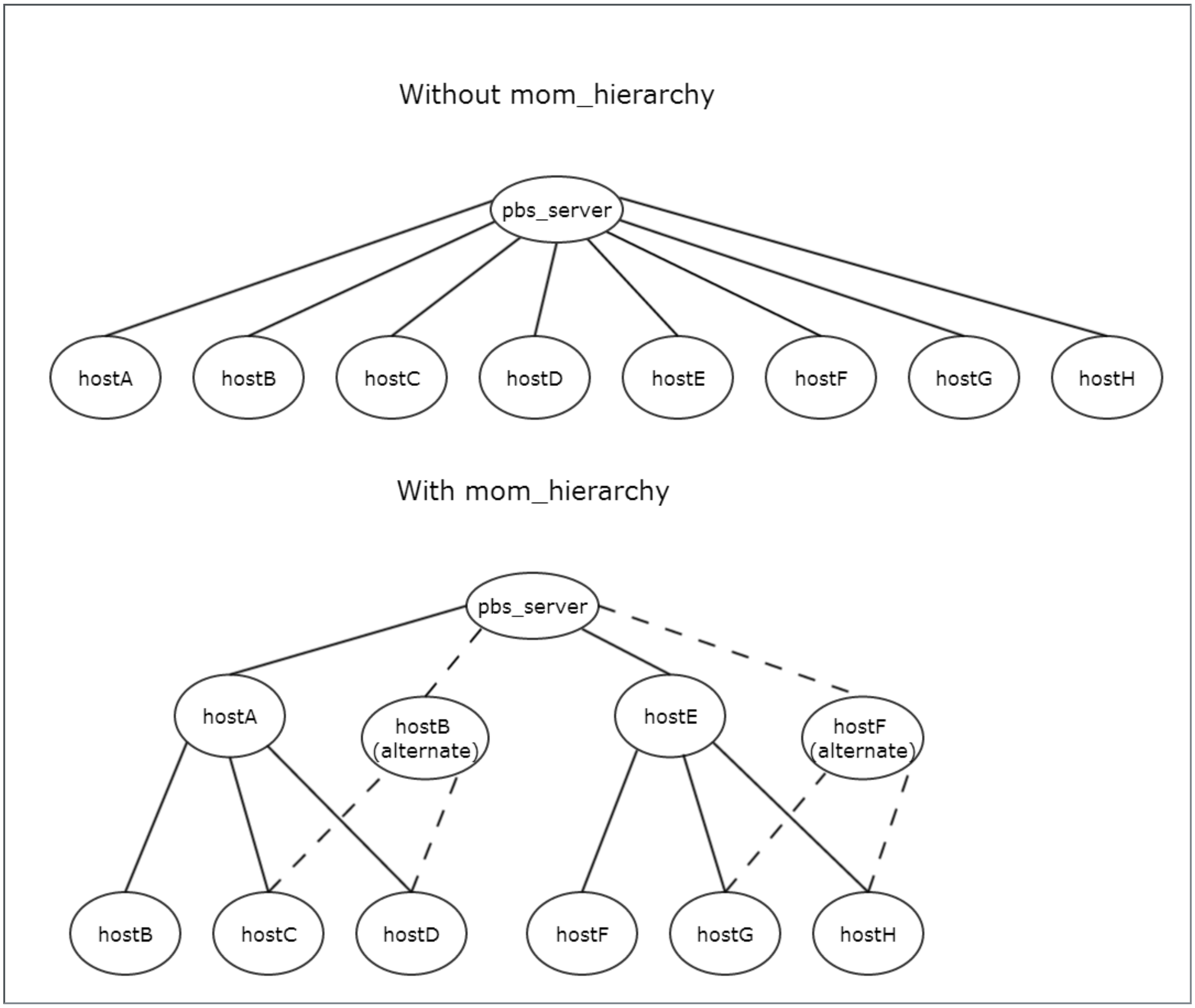
MOM Hierarchy

> Mom hierarchy is designed for large systems to configure how information is passed directly to the pbs_server.

The MOM hierarchy allows you to override the compute nodes' default behavior of reporting status updates directly to the pbs_server. Instead, you configure compute nodes so that each node sends its status update information to another compute node. The compute nodes pass the information up a tree or hierarchy until eventually the information reaches a node that will pass the information directly to pbs_server. This can significantly reduce network traffic and ease the load on the pbs_server in a large system.

MOM hierarchy example

The following example illustrates how information is passed to the pbs_server without and with mom_ hierarchy.



The dotted lines indicates an alternate path if the hierarchy-designated node goes down.

The following is the mom_hierachy_file for the with mom_hierarchy example:

```
<path>
  <level>hostA,hostB</level>
  <level>hostB,hostC,hostD</level>
</path>
<path>
  <level>hostE,hostF</level>
  <level>hostE,hostF,hostG</level>
</path>
```

Setting up the MOM hierarchy

The name of the file that contains the configuration information is named `mom_hierarchy`. By default, it is located in the `/var/spool/torque/server_priv` directory. The file uses syntax similar to XML:

```
<path>
  <level>comma-separated node list</level>
  <level>comma-separated node list</level>
  ...
</path>
...
```

The **<path></path>** tag pair identifies a group of compute nodes. The **<level></level>** tag pair contains a comma-separated list of compute node names listed by their hostnames. Multiple paths can be defined with multiple levels within each path.

Within a **<path></path>** tag pair the levels define the hierarchy. All nodes in the top level communicate directly with the server. All nodes in lower levels communicate to the first available node in the level directly above it. If the first node in the upper level goes down, the nodes in the subordinate level will then communicate to the next node in the upper level. If no nodes are available in an upper level then the node will communicate directly to the server.

If an upper level node has gone down and then becomes available, the lower level nodes will eventually find that the node is available and start sending their updates to that node.

> If you want to specify MOMs on a different port than the default, you must list the node in the form: `hostname:mom_manager_port`.
>
> For example:
>
> ```
> <path>
>   <level>hostname:mom_manager_port,... </level>
>   ...
> </path>
> ...
> ```

Putting the MOM hierarchy on the MOMs

You can put the MOM hierarchy file directly on the MOMs. The default location is `/var/spool/torque/mom_priv/mom_hierarchy`. This way, the pbs_server doesn't have to send the hierarchy to all the MOMs during each pbs_server startup. The hierarchy file still has to exist on the pbs_server and if the file versions conflict, the pbs_server version overwrites the local MOM file. When using a global file system accessible from both the MOMs and the pbs_server, it is recommended that the hierarchy file be symbolically linked to the MOMs.

Once the hierarchy file exists on the MOMs, start pbs_server with the -n option which tells pbs_server to not send the hierarchy file on startup. Instead, pbs_server waits until a MOM requests it.

Manual Setup of Initial Server Configuration

On a new installation of TORQUE, the server database must be initialized using the command `pbs_server -t create`. This command creates a file in *$TORQUEHOME*/`server_priv` named `serverdb` which contains the server configuration information.

The following output from qmgr shows the base configuration created by the command `pbs_server -t create`:

```
qmgr -c 'p s'
#
Set server attributes.
#
set server acl_hosts = kmn
set server log_events = 511
set server mail_from = adm
set server node_check_rate = 150
set server tcp_timeout = 6
```

This is a bare minimum configuration and it is not very useful. By using qmgr, the server configuration can be modified to set up TORQUE to do useful work. The following qmgr commands will create a queue and enable the server to accept and run jobs. These commands must be executed by root.

```
pbs_server -t create
qmgr -c "set server scheduling=true"
qmgr -c "create queue batch queue_type=execution"
qmgr -c "set queue batch started=true"
qmgr -c "set queue batch enabled=true"
qmgr -c "set queue batch resources_default.nodes=1"
qmgr -c "set queue batch resources_default.walltime=3600"
qmgr -c "set server default_queue=batch"
```

When TORQUE reports a new queue to Moab a class of the same name is automatically applied to all nodes.

In this example, the configuration database is initialized and the scheduling interface is activated using ('`scheduling=true`'). This option allows the scheduler to receive job and node events which allow it to be more responsive (See **scheduling** on page 2434 for more information). The next command creates a queue and specifies the queue type. Within PBS, the queue must be declared an '`execution` queue in order for it to run jobs. Additional configuration (i.e., setting the queue to `started` and `enabled`) allows the queue to *accept* job submissions, and *launch* queued jobs.

The next two lines are optional, setting default `node` and `walltime` attributes for a submitted job. These defaults will be picked up by a job if values are not explicitly set by the submitting user. The final line, `default_queue=batch`, is also a convenience line and indicates that a job should be placed in the `batch` queue unless explicitly assigned to another queue.

Additional information on configuration can be found in the admin manual and in the **qmgr** main page.

Related topics

- **TORQUE Installation Overview** on page 2193

Server Node File Configuration

This section contains information about configuring server node files. It explains how to specify node virtual processor counts and GPU counts, as well as how to specify node features or properties. For details, see these topics:

- **Basic Node Specification** on page 2223
- **Specifying Virtual Processor Count for a Node** on page 2223

- **Specifying GPU Count for a Node** on page 2224
- **Specifying Node Features (Node Properties)** on page 2224

Related topics

- **TORQUE Installation Overview** on page 2193
- **Appendix B: Server Parameters** on page 2417
- Moab node feature overview

Basic Node Specification

For the pbs_server to communicate with each of the MOMs, it needs to know which machines to contact. Each node that is to be a part of the batch system must be specified on a line in the server `nodes` file. This file is located at `TORQUE_HOME/server_priv/nodes`. In most cases, it is sufficient to specify just the node name on a line as in the following example:

`server_priv/nodes`:

```
node001
node002
node003
node004
```

Related topics

- **Server Node File Configuration** on page 2222

Specifying Virtual Processor Count for a Node

By default each node has one virtual processor. Increase the number using the **np** attribute in the nodes file. The value of np can be equal to the number of physical cores on the node or it can be set to a value which represents available "execution slots" for the node. The value used is determined by the administrator based on hardware, system, and site criteria.

The following example shows how to set the np value in the nodes file. In this example, we are assuming that node001 and node002 have four physical cores. The administrator wants the value of np for node001 to reflect that it has four cores. However, node002 will be set up to handle multiple virtual processors without regard to the number of physical cores on the system.

`server_priv/nodes`:

```
node001 np=4
node002 np=12
...
```

Related topics

- **Server Node File Configuration** on page 2222

Specifying GPU Count for a Node

Administrators can manually set the number of GPUs on a node or if they are using NVIDIA GPUs and drivers, they can have them detected automatically. For more information about how to set up TORQUE with GPUS, see **Accelerators** on page 794.

To manually set the number of GPUs on a node, use the gpus attribute in the nodes file. The value of GPUs is determined by the administrator based on hardware, system, and site criteria.

The following example shows how to set the GPU value in the nodes file. In the example, we assume node01 and node002 each have two physical GPUs. The administrator wants the value of node001 to reflect the physical GPUs available on that system and adds `gpus=2` to the nodes file entry for node001. However, node002 will be set up to handle multiple virtual GPUs without regard to the number of physical GPUs on the system.

`server_priv/nodes`:

```
node001 gpus=1
node002 gpus=4
...
```

Related topics

- **Server Node File Configuration** on page 2222

Specifying Node Features (Node Properties)

Node features can be specified by placing one or more white space-delimited strings on the line for the associated host as in the following example:

`server_priv/nodes`:

```
node001 np=2 fast ia64
node002 np=4 bigmem fast ia64 smp
...
```

These features can be used by users to request specific nodes when submitting jobs. For example:

```
qsub -l nodes=1:bigmem+1:fast job.sh
```

This job submission will look for a node with the bigmem feature (node002) and a node with the fast feature (either node001 or node002).

Related topics

- **Server Node File Configuration** on page 2222

Testing Server Configuration

If you have initialized TORQUE using the torque.setup script or started TORQUE using pbs_server -t create and pbs_server is still running, terminate the server by calling qterm. Next, start pbs_server again without the `-t create` arguments. Follow the script below to verify your server configuration.

The output for the examples below is based on the nodes file example in Specifying node features and Server configuration.

```
# verify all queues are properly configured
> qstat -q

server:kmn

Queue    Memory   CPU Time   Walltime   Node   Run  Que  Lm  State
-----    ------   --------   --------   ----   ---  ---  --  -----
batch    --        --        --         --     0    0    --  ER
                                               ---  ---
                                               0    0

# view additional server configuration
> qmgr -c 'p s'
#
# Create queues and set their attributes
#
#
# Create and define queue batch
#
create queue batch
set queue batch queue_type = Execution
set queue batch resources_default.nodes = 1
set queue batch resources_default.walltime = 01:00:00
set queue batch enabled = True
set queue batch started = True
#
# Set server attributes.
#
set server scheduling = True
set server acl_hosts = kmn
set server managers = user1@kmn
set server operators = user1@kmn
set server default_queue = batch
set server log_events = 511
set server mail_from = adm
set server node_check_rate = 150
set server tcp_timeout = 300
set server job_stat_rate = 45
set server poll_jobs = True
set server mom_job_sync = True
set server keep_completed = 300
set server next_job_number = 0

# verify all nodes are correctly reporting
> pbsnodes -a
node001
  state=free
  np=2
  properties=bigmem,fast,ia64,smp
  ntype=cluster
  status=rectime=1328810402,varattr=,jobs=,state=free,netload=6814326158,gres=,loadave
=0.21,ncpus=6,physmem=8193724kb,
availmem=13922548kb,totmem=16581304kb,idletime=3,nusers=3,nsessions=18,sessions=1876
1120 1912 1926 1937 1951 2019 2057 28399 2126 2140 2323 5419 17948 19356 27726 22254
29569,uname=Linux kmn 2.6.38-11-generic #48-Ubuntu SMP Fri Jul 29 19:02:55 UTC 2011
x86_64,opsys=linux
  mom_service_port = 15002
  mom_manager_port = 15003
  gpus = 0
# submit a basic job - DO NOT RUN AS ROOT
> su - testuser
> echo "sleep 30" | qsub

# verify jobs display
> qstat

Job id   Name   User      Time Use  S  Queue
------   -----  ----      --------  -- -----
```

```
0.kmn     STDIN   knielson          0   Q   batch
```

At this point, the job should be in the **Q** state and will not run because a scheduler is not running yet. TORQUE can use its native scheduler by running pbs_sched or an advanced scheduler (such as Moab Workload Manager). See **Integrating Schedulers for TORQUE** on page 2304 for details on setting up an advanced scheduler.

Related topics

- **TORQUE Installation Overview** on page 2193

TORQUE on NUMA Systems

Starting in TORQUE version 3.0, TORQUE can be configured to take full advantage of Non-Uniform Memory Architecture (NUMA) systems. The following instructions are a result of development on SGI Altix and UV hardware.

For details, see these topics:

- **TORQUE NUMA Configuration** on page 2227
- **Building TORQUE with NUMA Support** on page 2227

TORQUE NUMA Configuration

There are three steps to configure TORQUE to take advantage of NUMA architectures:

1. Configure TORQUE with `--enable-numa-support`.
2. Create the `mom_priv/mom.layout` file.
3. Configure `server_priv/nodes`.

Related topics

- **TORQUE on NUMA Systems** on page 2227

Building TORQUE with NUMA Support

To turn on NUMA support for TORQUE the `--enable-numa-support` option must be used during the configure portion of the installation. In addition to any other configuration options, add the `--enable-numa-support` option as indicated in the following example:

```
$ ./configure --enable-numa-support
```

> Don't use MOM hierarchy with NUMA.

When TORQUE is enabled to run with NUMA support, there is only a single instance of pbs_mom (MOM) that is run on the system. However, TORQUE will report that there are multiple nodes running in the cluster. While pbs_mom and pbs_server both know there is only one instance of pbs_mom, they manage the cluster as if there were multiple separate MOM nodes.

The mom.layout file is a virtual mapping between the system hardware configuration and how the administrator wants TORQUE to view the system. Each line in mom.layout equates to a node in the cluster and is referred to as a NUMA node.

Automatically Creating mom.layout (Recommended)

A perl script named mom_gencfg is provided in the contrib/ directory that generates the mom.layout file for you. The script can be customized by setting a few variables in it. To automatically create the mom.layout file, follow these instructions (these instructions are also included in the script):

1. Verify hwloc version 1.1 or higher is installed - see contrib/hwloc_install.sh.
2. Install Sys::Hwloc from CPAN.
3. Verify $PBS_HOME is set to the proper value.
4. Update the variables in the 'Config Definitions' section of the script. Especially update firstNodeId and nodesPerBoard if desired. The firstNodeId variable should be set above 0 if you have a root cpuset that you wish to exclude and the nodesPerBoard variable is the number of NUMA nodes per board. Each node is defined in /sys/devices/system/node, in a subdirectory node<node index>
5. Back up your current file in case a variable is set incorrectly or neglected.
6. Run the script.
7.
```
$ ./mom_gencfg
```

Manually Creating mom.layout

To properly set up the mom.layout file, it is important to know how the hardware is configured. Use the topology command line utility and inspect the contents of /sys/devices/system/node. The hwloc library can also be used to create a custom discovery tool.

Typing topology on the command line of a NUMA system produces something similar to the following:

```
Partition number: 0
6 Blades
72 CPUs
378.43 Gb Memory Total

Blade          ID       asic  NASID         Memory
-------------------------------------------------
    0 r001i01b00  UVHub 1.0      0    67089152 kB
    1 r001i01b01  UVHub 1.0      2    67092480 kB
    2 r001i01b02  UVHub 1.0      4    67092480 kB
    3 r001i01b03  UVHub 1.0      6    67092480 kB
    4 r001i01b04  UVHub 1.0      8    67092480 kB
    5 r001i01b05  UVHub 1.0     10    67092480 kB

CPU      Blade PhysID CoreID APIC-ID Family Model Speed L1(KiB) L2(KiB) L3(KiB)
-------------------------------------------------------------------------------
  0 r001i01b00     00     00       0      6    46  2666 32d/32i     256   18432
  1 r001i01b00     00     02       4      6    46  2666 32d/32i     256   18432
  2 r001i01b00     00     03       6      6    46  2666 32d/32i     256   18432
  3 r001i01b00     00     08      16      6    46  2666 32d/32i     256   18432
  4 r001i01b00     00     09      18      6    46  2666 32d/32i     256   18432
  5 r001i01b00     00     11      22      6    46  2666 32d/32i     256   18432
  6 r001i01b00     01     00      32      6    46  2666 32d/32i     256   18432
  7 r001i01b00     01     02      36      6    46  2666 32d/32i     256   18432
  8 r001i01b00     01     03      38      6    46  2666 32d/32i     256   18432
  9 r001i01b00     01     08      48      6    46  2666 32d/32i     256   18432
 10 r001i01b00     01     09      50      6    46  2666 32d/32i     256   18432
 11 r001i01b00     01     11      54      6    46  2666 32d/32i     256   18432
 12 r001i01b01     02     00      64      6    46  2666 32d/32i     256   18432
 13 r001i01b01     02     02      68      6    46  2666 32d/32i     256   18432
 14 r001i01b01     02     03      70      6    46  2666 32d/32i     256   18432
```

From this partial output, note that this system has 72 CPUs on 6 blades. Each blade has 12 CPUs grouped into clusters of 6 CPUs. If the entire content of this command were printed you would see each Blade ID and the CPU ID assigned to each blade.

The topology command shows how the CPUs are distributed, but you likely also need to know where memory is located relative to CPUs, so go to `/sys/devices/system/node`. If you list the node directory you will see something similar to the following:

```
# ls -al
total 0
drwxr-xr-x 14 root root    0 Dec 3 12:14 .
drwxr-xr-x 14 root root    0 Dec 3 12:13 ..
-r--r--r--  1 root root 4096 Dec 3 14:58 has_cpu
-r--r--r--  1 root root 4096 Dec 3 14:58 has_normal_memory
drwxr-xr-x  2 root root 0 Dec 3 12:14 node0
drwxr-xr-x  2 root root 0 Dec 3 12:14 node1
drwxr-xr-x  2 root root 0 Dec 3 12:14 node10
drwxr-xr-x  2 root root 0 Dec 3 12:14 node11
drwxr-xr-x  2 root root 0 Dec 3 12:14 node2
drwxr-xr-x  2 root root 0 Dec 3 12:14 node3
drwxr-xr-x  2 root root 0 Dec 3 12:14 node4
drwxr-xr-x  2 root root 0 Dec 3 12:14 node5
drwxr-xr-x  2 root root 0 Dec 3 12:14 node6
drwxr-xr-x  2 root root 0 Dec 3 12:14 node7
drwxr-xr-x  2 root root 0 Dec 3 12:14 node8
drwxr-xr-x  2 root root 0 Dec 3 12:14 node9
-r--r--r--  1 root root 4096 Dec 3 14:58 online
-r--r--r--  1 root root 4096 Dec 3 14:58 possible
```

The directory entries node0, node1,...node11 represent groups of memory and CPUs local to each other. These groups are a node board, a grouping of resources that are close together. In most cases, a node board is made up of memory and processor cores. Each bank of memory is called a memory node by the operating system, and there are certain CPUs that can access that memory very rapidly. Note under the directory for node board node0 that there is an entry called `cpulist`. This contains the CPU IDs of all CPUs local to the memory in node board 0.

Now create the `mom.layout` file. The content of cpulist 0-5 are local to the memory of node board 0, and the memory and cpus for that node are specified in the layout file by saying nodes=0. The cpulist for node board 1 shows 6-11 and memory node index 1. To specify this, simply write nodes=1. Repeat this for all twelve node boards and create the following `mom.layout` file for the 72 CPU system.

```
nodes=0
nodes=1
nodes=2
nodes=3
nodes=4
nodes=5
nodes=6
nodes=7
nodes=8
nodes=9
nodes=10
nodes=11
```

Each line in the mom.layout file is reported as a node to pbs_server by the pbs_mom daemon.

The `mom.layout` file does not need to match the hardware layout exactly. It is possible to combine node boards and create larger NUMA nodes. The following example shows how to do this:

```
nodes=0-1
```

The memory nodes can be combined the same as CPUs. The memory nodes combined must be contiguous. You cannot combine mem 0 and 2.

Configuring server_priv/nodes

The pbs_server requires awareness of how the MOM is reporting nodes since there is only one MOM daemon and multiple MOM nodes. So, configure the `server_priv/nodes` file with the **num_node_boards** and **numa_board_str** attributes. The attribute num_node_boards tells pbs_server how many numa nodes are reported by the MOM. Following is an example of how to configure the nodes file with num_node_boards:

```
numa-10 np=72 num_node_boards=12
```

This line in the nodes file tells pbs_server there is a host named numa-10 and that it has 72 processors and 12 nodes. The pbs_server divides the value of np (72) by the value for num_node_boards (12) and determines there are 6 CPUs per NUMA node.

In this example, the NUMA system is uniform in its configuration of CPUs per node board, but a system does not need to be configured with the same number of CPUs per node board. For systems with non-uniform CPU distributions, use the attribute **numa_board_str** to let pbs_server know where CPUs are located in the cluster.

The following is an example of how to configure the `server_priv/nodes` file for non-uniformly distributed CPUs:

```
Numa-11 numa_board_str=6,8,12
```

In this configuration, pbs_server knows it has three MOM nodes and the nodes have 6, 8, and 12 CPUs respectively. Note that the attribute np is not used. The np attribute is ignored because the number of CPUs per node is expressly given.

Enforcement of memory resource limits

TORQUE can better enforce memory limits with the use of the utility **memacctd**. The memacctd utility is provided by SGI on SuSe Linux Enterprise Edition (SLES). It is a daemon that caches memory footprints when it is queried. When configured to use the memory monitor, TORQUE queries memacctd. It is up to the user to make sure memacctd is installed. See the SGI memacctd man page for more information.

To configure TORQUE to use memacctd for memory enforcement

1. Start **memacctd** as instructed by SGI.
2. Reconfigure TORQUE with `--enable-memacct`. This will link in the necessary library when TORQUE is recompiled.
3. Recompile and reinstall TORQUE.
4. Restart all MOM nodes.
5. (Optional) Alter the **qsub** filter to include a default memory limit for all jobs that are not submitted with memory limit.

Related topics

- **TORQUE NUMA Configuration** on page 2227
- **TORQUE on NUMA Systems** on page 2227

TORQUE Multi-MOM

Starting in TORQUE version 3.0 users can run multiple MOMs on a single node. The initial reason to develop a multiple MOM capability was for testing purposes. A small cluster can be made to look larger since each MOM instance is treated as a separate node.

When running multiple MOMs on a node each MOM must have its own service and manager ports assigned. The default ports used by the MOM are 15002 and 15003. With the multi-mom alternate ports can be used without the need to change the default ports for pbs_server even when running a single instance of the MOM.

For details, see these topics:

- **Multi-MOM Configuration** on page 2231
- **Stopping pbs_mom in Multi-MOM Mode** on page 2233

Multi-MOM Configuration

There are three steps to setting up multi-MOM capability:

1. **Configure server_priv/nodes** on page 2232
2. **/etc/hosts file** on page 2232
3. **Starting pbs_mom with multi-MOM options** on page 2232

Configure server_priv/nodes

The attributes **mom_service_port** and **mom_manager_port** were added to the nodes file syntax to accommodate multiple MOMs on a single node. By default pbs_mom opens ports 15002 and 15003 for the service and management ports respectively. For multiple MOMs to run on the same IP address they need to have their own port values so they can be distinguished from each other. pbs_server learns about the port addresses of the different MOMs from entries in the `server_priv/nodes` file. The following is an example of a nodes file configured for multiple MOMs:

```
hosta   np=2
hosta-1 np=2 mom_service_port=30001 mom_manager_port=30002
hosta-2 np=2 mom_service_port=31001 mom_manager_port=31002
hosta-3 np=2 mom_service_port=32001 mom_manager_port=32002
```

Note that all entries have a unique host name and that all port values are also unique. The entry `hosta` does not have a mom_service_port or mom_manager_port given. If unspecified, then the MOM defaults to ports 15002 and 15003.

/etc/hosts file

Host names in the `server_priv/nodes` file must be resolvable. Creating an alias for each host enables the server to find the IP address for each MOM; the server uses the port values from the `server_priv/nodes` file to contact the correct MOM. An example `/etc/hosts` entry for the previous `server_priv/nodes` example might look like the following:

```
192.65.73.10 hosta hosta-1 hosta-2 hosta-3
```

Even though the host name and all the aliases resolve to the same IP address, each MOM instance can still be distinguished from the others because of the unique port value assigned in the `server_priv/nodes` file.

Starting pbs_mom with multi-MOM options

To start multiple instances of pbs_mom on the same node, use the following syntax (see **pbs_mom** on page 2344 for details):

```
pbs_mom -m -M <port value of MOM_service_port> -R <port value of MOM_manager_port> -A
<name of MOM alias>
```

Continuing based on the earlier example, if you want to create four MOMs on `hosta`, type the following at the command line:

```
# pbs_mom -m -M 30001 -R 30002 -A hosta-1
# pbs_mom -m -M 31001 -R 31002 -A hosta-2
# pbs_mom -m -M 32001 -R 32002 -A hosta-3
# pbs_mom
```

Notice that the last call to pbs_mom uses no arguments. By default pbs_mom opens on ports 15002 and 15003. No arguments are necessary because there are no conflicts.

Related topics

- **TORQUE Multi-MOM** on page 2231
- **Stopping pbs_mom in Multi-MOM Mode** on page 2233

Stopping pbs_mom in Multi-MOM Mode

Terminate pbs_mom by using the `momctl -s` command (for details, see **momctl**). For any MOM using the default manager port 15003, the `momctl -s` command stops the MOM. However, to terminate MOMs with a manager port value not equal to 15003, you must use the following syntax:

```
momctl -s -p <port value of MOM_manager_port>
```

The **-p** option sends the terminating signal to the MOM manager port and the MOM is terminated.

Related topics

- **TORQUE Multi-MOM** on page 2231
- **Multi-MOM Configuration** on page 2231

Submitting and Managing Jobs

This section contains information about how you can submit and manage jobs with TORQUE. For details, see the following topics:

Job Submission

Job submission is accomplished using the **qsub** command, which takes a number of command line arguments and integrates such into the specified PBS command file. The PBS command file may be specified as a filename on the qsub command line or may be entered via STDIN.

- The PBS command file does not need to be executable.
- The PBS command file may be *piped* into qsub (i.e., `cat pbs.cmd | qsub`).
- In the case of parallel jobs, the PBS command file is staged to, and executed on, the first allocated compute node only. (Use **pbsdsh** to run actions on multiple nodes.)
- The command script is executed from the user's home directory in all cases. (The script may determine the submission directory by using the $PBS_O_WORKDIR environment variable)
- The command script will be executed using the default set of user environment variables unless the **-V** or **-v** flags are specified to include aspects of the job submission environment.
- PBS directives should be declared first in the job script.

```
#PBS -S /bin/bash
#PBS -m abe
#PBS -M <yourEmail@company.com>
echo sleep 300
```

This is an example of properly declared PBS directives.

```
#PBS -S /bin/bash
SOMEVARIABLE=42
#PBS -m abe
#PBS -M <yourEmail@company.com>
echo sleep 300
```

This is an example of improperly declared PBS directives. PBS directives below "SOMEVARIABLE=42" are ignored.

> By default, job submission is allowed only on the TORQUE server host (host on which **pbs_server** is running). Enablement of job submission from other hosts is documented in **Server Configuration** on page 2215.

> Versions of TORQUE earlier than 2.4.5 attempted to apply queue and server defaults to a job that didn't have defaults specified. If a setting still did not have a value after that, TORQUE applied the queue and server maximum values to a job (meaning, the maximum values for an applicable setting were applied to jobs that had no specified or default value).
>
> In TORQUE 2.4.5 and later, the queue and server maximum values are no longer used as a value for missing settings.

This section contains these topics:

- **Requesting Generic Resources** on page 2244
- **Requesting Floating Resources** on page 2244
- **Requesting Other Resources** on page 2245
- **Exported Batch Environment Variables** on page 2245
- **Enabling Trusted Submit Hosts** on page 2247
- **Example Submit Scripts** on page 2247

Related topics

- Maui Documentation
- http://www.lunarc.lu.se
- http://www.clusters.umaine.edu/wiki/index.php/Example_Submission_Scripts
- **Appendix J: Job Submission Filter ("qsub wrapper")** on page 2479 – Allow local checking and modification of submitted job

Multiple Job Submission

Sometimes users will want to submit large numbers of jobs based on the same job script. Rather than using a script to repeatedly call qsub, a feature known as job arrays now exists to allow the creation of multiple jobs with one qsub command. Additionally, this feature includes a new job naming convention that allows users to reference the entire set of jobs as a unit, or to reference one particular job from the set.

Job arrays are submitted through the **-t** option to qsub, or by using `#PBS -t` in your batch script. This option takes a comma-separated list consisting of either a single job ID number, or a pair of numbers separated by a dash. Each of these jobs created will use the same script and will be running in a nearly identical environment.

```
> qsub -t 0-4 job_script
1098[].hostname

> qstat -t
1098[0].hostname ...
1098[1].hostname ...
1098[2].hostname ...
1098[3].hostname ...
1098[4].hostname ...
```

Versions of TORQUE earlier than 2.3 had different semantics for the **-t** argument. In these versions, -t took a single integer number—a count of the number of jobs to be created.

Each 1098[x] job has an environment variable called PBS_ARRAYID, which is set to the value of the array index of the job, so 1098[0].hostname would have PBS_ARRAYID set to 0. This allows you to create job arrays where each job in the array performs slightly different actions based on the value of this variable, such as performing the same tasks on different input files. One other difference in the environment between jobs in the same array is the value of the PBS_JOBNAME variable.

```
# These two examples are equivalent in TORQUE 2.2
> qsub -t 0-99
> qsub -t 100

# You can also pass comma delimited lists of ids and ranges:
> qsub -t 0,10,20,30,40
> qsub -t 0-50,60,70,80
```

Running qstat displays a job summary, which provides an overview of the array's state. To see each job in the array, run qstat **-t**.

The **qalter**, **qdel**, **qhold**, and **qrls** commands can operate on arrays—either the entire array or a range of that array. Additionally, any job in the array may be accessed normally by using that job's ID, just as you would with any other job. For example, running the following command would run only the specified job:

```
qrun 1098[0].hostname
```

Slot Limit

The slot limit is a way for administrators to limit the number of jobs from a job array that can be eligible for scheduling at the same time. When a slot limit is used, TORQUE puts a hold on all jobs in the array that exceed the slot limit. When an eligible job in the array completes, TORQUE removes the hold flag from the next job in the array. Slot limits can be declared globally with the max_slot_limit parameter, or on a per-job basis with qsub -t.

Related topics

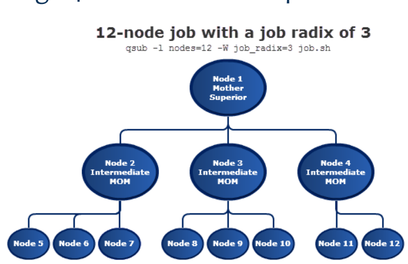
- **Job Submission** on page 2234

Managing Multi-node Jobs

By default, when a multi-node job runs, the Mother Superior manages the job across all the sister nodes by communicating with each of them and updating `pbs_server`. Each of the sister nodes sends its updates and stdout and stderr directly to the Mother Superior. When you run an extremely large job using hundreds or thousands of nodes, you may want to reduce the amount of network traffic sent from the sisters to the Mother Superior by specifying a job radix. Job radix sets a maximum number of nodes with which the Mother Superior and resulting intermediate MOMs communicate and is specified using the **-W** on page 2406 option for qsub.

For example, if you submit a smaller, 12-node job and specify `job_radix=3`, Mother Superior and each resulting intermediate MOM is only allowed to receive communication from 3 subordinate nodes.

Image 4-1: Job radix example



The Mother Superior picks three sister nodes with which to communicate the job information. Each of those nodes (intermediate MOMs) receives a list of all sister nodes that will be subordinate to it. They each contact up to three nodes and pass the job information on to those nodes. This pattern continues until the bottom level is reached. All communication is now passed across this new hierarchy. The stdout and stderr data is aggregated and sent up the tree until it reaches the Mother Superior, where it is saved and copied to the `.o` and `.e` files.

Job radix is meant for extremely large jobs only. It is a tunable parameter and should be adjusted according to local conditions in order to produce the best results.

Requesting Resources

Various resources can be requested at the time of job submission. A job can request a particular node, a particular node attribute, or even a number of nodes with particular attributes. Either native TORQUE resources or external scheduler resource extensions may be specified. The native TORQUE resources are listed in the following table:

Resource	Format	Description
arch	string	Specifies the administrator defined system architecture required. This defaults to whatever the **PBS_MACH** string is set to in "local.mk".
cput	seconds, or [[HH:]MM;]SS	Maximum amount of CPU time used by all processes in the job.

<table>
<tr><th>Resource</th><th>Format</th><th>Description</th></tr>
<tr><td>cpuclock</td><td>string</td><td>
Specify the CPU clock frequency for each node requested for this job. A cpuclock request applies to every processor on every node in the request. Specifying varying CPU frequencies for different nodes or different processors on nodes in a single job request is not supported.
Not all processors support all possible frequencies or ACPI states. If the requested frequency is not supported by the CPU, the nearest frequency is used.
ALPS 1.4 or later is required when using cpuclock on Cray.
The clock frequency can be specified via:

a number that indicates the clock frequency (with or without the SI unit suffix).
<pre>qsub -l cpuclock=1800,nodes=2 script.sh
qsub -l cpuclock=1800mhz,nodes=2 script.sh</pre>
This job requests 2 nodes and specifies their CPU frequencies should be set to 1800 MHz.

a Linux power governor policy name. The governor names are:

performance: This governor instructs Linux to operate each logical processor at its maximum clock frequency.
This setting consumes the most power and workload executes at the fastest possible speed.
powersave: This governor instructs Linux to operate each logical processor at its minimum clock frequency.
This setting executes workload at the slowest possible speed. This setting does not necessarily consume the least amount of power since applications execute slower, and may actually consume more energy because of the additional time needed to complete the workload's execution.
ondemand: This governor dynamically switches the logical processor's clock frequency to the maximum value when system load is high and to the minimum value when the system load is low.
This setting causes workload to execute at the fastest possible speed or the slowest possible speed, depending on OS load. The system switches between consuming the most power and the least power.

</td></tr>
</table>

Resource	Format	Description
		The power saving benefits of *ondemand* might be non-existent due to frequency switching latency if the system load causes clock frequency changes too often. This has been true for older processors since changing the clock frequency required putting the processor into the C3 "sleep" state, changing its clock frequency, and then waking it up, all of which required a significant amount of time. Newer processors, such as the Intel Xeon E5-2600 Sandy Bridge processors, can change clock frequency dynamically and much faster. ◦ *conservative*: This governor operates like the *ondemand* governor but is more conservative in switching between frequencies. It switches more gradually and uses all possible clock frequencies. This governor can switch to an intermediate clock frequency if it seems appropriate to the system load and usage, which the *ondemand* governor does not do. `qsub -l cpuclock=performance,nodes=2 script.sh` *This job requests 2 nodes and specifies their CPU frequencies should be set to the performance power governor policy.* • an ACPI performance state (or P-state) with or without the P prefix. P-states are a special range of values (0-15) that map to specific frequencies. Not all processors support all 16 states, however, they all start at `P0`. `P0` sets the CPU clock frequency to the highest performance state which runs at the maximum frequency. `P15` sets the CPU clock frequency to the lowest performance state which runs at the lowest frequency. `qsub -l cpuclock=3,nodes=2 script.sh` `qsub -l cpuclock=p3,nodes=2 script.sh` *This job requests 2 nodes and specifies their CPU frequencies should be set to a performance state of 3.* When reviewing job or node properties when **cpuclock** was used, be mindful of unit conversion. The OS reports frequency in Hz, not MHz or GHz.

Resource	Format	Description
epilogue	string	Specifies a user owned epilogue script which will be run before the system epilogue and epilogue.user scripts at the completion of a job. The syntax is `epilogue=<file>`. The file can be designated with an absolute or relative path. For more information, see **Appendix G: Prologue and Epilogue Scripts** on page 2469.
feature	string	Specifies a property or feature for the job. Feature corresponds to TORQUE node properties and Moab features. `qsub script.sh -l procs=10,feature=bigmem`
file	**size**	The amount of total disk requested for the job. (Ignored on Unicos.)
host	string	Name of the host on which the job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent.
mem	**size**	Maximum amount of physical memory used by the job. Ignored on Darwin, Digital Unix, Free BSD, HPUX 11, IRIX, NetBSD, and SunOS. Not implemented on AIX and HPUX 10. The `mem` resource will only work for single-node jobs. If your job requires multiple nodes, use `pmem` instead.
ncpus	integer	The number of processors in one task where a task cannot span nodes. You cannot request both ncpus and nodes in the same job.
nice	integer	Number between -20 (highest priority) and 19 (lowest priority). Adjust the process execution priority.

Resource	Format	Description
nodes	{<node_count> | <hostname>} [:ppn=<ppn>] [:gpus=<gpu>] [:<property> [:<property>]...] [+ ...]	Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more node_specs joined with the + (plus) character: node_spec [+node_spec...]. Each node_spec is a number of nodes required of the type declared in the node_spec and a name of one or more properties desired for the nodes. The number, the name, and each property in the node_spec are separated by a : (colon). If no number is specified, one (1) is assumed. The name of a node is its hostname. The properties of nodes are: • **ppn=#** - Specify the number of virtual processors per node requested for this job. The number of virtual processors available on a node by default is 1, but it can be configured in the `$TORQUE_HOME/server_priv/nodes` file using the **np** attribute (see **Server Node File Configuration** on page 2222). The virtual processor can relate to a physical core on the node or it can be interpreted as an "execution slot" such as on sites that set the node np value greater than the number of physical cores (or hyper-thread contexts). The ppn value is a characteristic of the hardware, system, and site, and its value is to be determined by the administrator. • **gpus=#** - Specify the number of GPUs per node requested for this job. The number of GPUs available on a node can be configured in the $TORQUE_HOME/server_priv/nodes file using the gpu attribute (see **Server Node File Configuration** on page 2222). The GPU value is a characteristic of the hardware, system, and site, and its value is to be determined by the administrator. • **property** - A string assigned by the system administrator specifying a node's features. Check with your administrator as to the node names and properties available to you. TORQUE does not have a TPN (tasks per node) property. You can specify TPN in Moab Workload Manager with TORQUE as your resource manager, but TORQUE does not recognize the property when it is submitted directly to it via qsub. See **qsub -l nodes** on page 2243 for examples. By default, the node resource is mapped to a virtual node (that is, directly to a processor, not a full physical compute node). This behavior can be changed within Maui or Moab by setting the JOBNODEMATCHPOLICY parameter. See **Appendix A: Moab Parameters** on page 902 for more information.
opsys	string	Specifies the administrator defined operating system as defined in the MOM configuration file.

Resource	Format	Description
other	string	Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. This does not work for msub using Moab and Maui.
pcput	seconds, or [[HH:]MM:]SS	Maximum amount of CPU time used by any single process in the job.
pmem	**size**	Maximum amount of physical memory used by any single process of the job. (Ignored on Fujitsu. Not implemented on Digital Unix and HPUX.)
procs	procs=<integer>	*(Applicable in version 2.5.0 and later.)* The number of processors to be allocated to a job. The processors can come from one or more qualified node(s). Only one procs declaration may be used per submitted **qsub** command. `> qsub -l nodes=3 -1 procs=2`
procs_bit-map	string	A string made up of 1's and 0's in reverse order of the processor cores requested. A **procs_bitmap=1110** means the job requests a node that has four available cores, but the job runs exclusively on cores two, three, and four. With this bitmap, core one is not used. For more information, see **Scheduling Cores** on page 2275.
prologue	string	Specifies a user owned prologue script which will be run after the system prologue and prologue.user scripts at the beginning of a job. The syntax is `prologue=<file>`. The file can be designated with an absolute or relative path. For more information, see **Appendix G: Prologue and Epilogue Scripts** on page 2469.
pvmem	**size**	Maximum amount of virtual memory used by any single process in the job. (Ignored on Unicos.)
size	integer	For TORQUE, this resource has no meaning. It is passed on to the scheduler for interpretation. In the Moab scheduler, the size resource is intended for use in Cray installations only.
software	string	Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. (See **License Management** on page 663 for more information.)

Resource	Format	Description
vmem	size	Maximum amount of virtual memory used by all concurrent processes in the job. (Ignored on Unicos.)
walltime	seconds, or [[HH:]MM:]SS	Maximum amount of real time during which the job can be in the running state.

size

The size format specifies the maximum amount in terms of bytes or words. It is expressed in the form *integer[suffix]*. The suffix is a multiplier defined in the following table ("b" means bytes [the default] and "w" means words). The size of a word is calculated on the execution server as its word size.

Suffix		Multiplier
b	w	1
kb	kw	1024
mb	mw	1,048,576
gb	gw	1,073,741,824
tb	tw	1,099,511,627,776

Example 4-10: qsub -l nodes

Usage	Description
`> qsub -l nodes=12`	Request 12 nodes of any type
`> qsub -l nodes=2:server+14`	Request 2 "server" nodes and 14 other nodes (a total of 16) - this specifies two node_specs, "2:server" and "14"
`> qsub -l nodes-s=server:hippi+10:noserver+3:bigmem:hippi`	Request (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not serv-ers, and (c) 3 nodes that have a large amount of memory and have hippi
`> qsub -l nodes=b2005+b1803+b1813`	Request 3 specific nodes by hostname

Usage	Description
`> qsub -l nodes=4:ppn=2`	Request 2 processors on each of four nodes
`> qsub -l nodes=1:ppn=4`	Request 4 processors on one node
`> qsub -l nodes=2:blue:ppn=2+red:ppn=3+b1014`	Request 2 processors on each of two blue nodes, three processors on one red node, and the compute node "b1014"

Example 4-11:

This job requests a node with 200MB of available memory:

```
> qsub -l mem=200mb /home/user/script.sh
```

Example 4-12:

This job will wait until node01 is free with 200MB of available memory:

```
> qsub -l nodes=node01,mem=200mb /home/user/script.sh
```

Related topics

- **Job Submission** on page 2234

Requesting Generic Resources

When **generic** resources have been assigned to nodes using the server's nodes file, these resources can be requested at the time of job submission using the *other* field. (See **Managing Consumable Generic Resources** on page 573 for details on configuration within Moab).

Example 4-13: Generic

This job will run on any node that has the generic resource **matlab**.

```
> qsub -l other=matlab /home/user/script.sh
```

> This can also be requested at the time of job submission using the `-W x=GRES:matlab` flag.

Related topics

- **Requesting Resources** on page 2237
- **Job Submission** on page 2234

Requesting Floating Resources

When **floating** resources have been set up inside Moab, they can be requested in the same way as **generic** resources. Moab will automatically understand that these resources are floating and will schedule the job accordingly. (See **Managing Shared Cluster Resources (Floating Resources)** on page 567 for details on configuration within Moab.)

Example 4-14: Floating

This job will run on any node when there are enough floating resources available.

```
> qsub -l other=matlab /home/user/script.sh
```

> This can also be requested at the time of job submission using the `-W x=GRES:matlab` flag.

Related topics

- **Requesting Resources** on page 2237
- **Job Submission** on page 2234

Requesting Other Resources

Many other resources can be requested at the time of job submission using the Moab Workload Manager. See **Resource Manager Extensions** on page 618 for a list of these supported requests and correct syntax.

Related topics

- **Requesting Resources** on page 2237
- **Job Submission** on page 2234

Exported Batch Environment Variables

When a batch job is started, a number of variables are introduced into the job's environment that can be used by the batch script in making decisions, creating output files, and so forth. These variables are listed in the following table:

Variable	Description
PBS_JOBNAME	User specified jobname
PBS_ARRAYID	Zero-based value of job array index for this job (in version 2.2.0 and later)
PBS_GPUFILE	Line-delimited list of GPUs allocated to the job located in `$TORQUE_HOME/aux/jobidgpu`. Each line follows the following format: `<host>-gpu<number>` For example, `myhost-gpu1`.
PBS_O_WORKDIR	Job's submission directory

Variable	Description
PBS_ENVIRONMENT	N/A
PBS_TASKNUM	Number of tasks requested
PBS_O_HOME	Home directory of submitting user
PBS_MOMPORT	Active port for MOM daemon
PBS_O_LOGNAME	Name of submitting user
PBS_O_LANG	Language variable for job
PBS_JOBCOOKIE	Job cookie
PBS_JOBID	Unique pbs job id
PBS_NODENUM	Node offset number
PBS_NUM_NODES	Number of nodes allocated to the job
PBS_NUM_PPN	Number of procs per node allocated to the job
PBS_O_SHELL	Script shell
PBS_O_HOST	Host on which job script is currently running
PBS_QUEUE	Job queue
PBS_NODEFILE	File containing line delimited list of nodes allocated to the job
PBS_NP	Number of execution slots (cores) for the job
PBS_O_PATH	Path variable used to locate executables within job script

Related topics

- **Requesting Resources** on page 2237
- **Job Submission** on page 2234

Enabling Trusted Submit Hosts

By default, only the node running the pbs_server daemon is allowed to submit jobs. Additional nodes can be trusted as submit hosts by taking any of the following steps:

- Set the **allow_node_submit** server parameter (see **Allowing job submission from compute hosts** on page 2217).

  Allows any host trusted as a compute host to also be trusted as a submit host.

- Set the **submit_hosts** server parameter (see **Using the "submit_hosts" service parameter** on page 2216).

  Allows specified hosts to be trusted as a submit host.

- Use **.rhosts** to enable `ruserok()` based authentication (see **Using RCmd authentication** on page 2216).

See **Configuring job submission hosts** on page 2216 for more information.

> When you enable **allow_node_submit** on page 2418, you must also enable the **allow_proxy_user** on page 2418 parameter to allow user proxying when submitting and running jobs.

Related topics

- **Job Submission** on page 2234

Example Submit Scripts

The following is an example job test script:

```
#!/bin/sh
#
#This is an example script example.sh
#
#These commands set up the Grid Environment for your job:
#PBS -N ExampleJob
#PBS -l nodes=1,walltime=00:01:00
#PBS -q np_workq
#PBS -M YOURUNIQNAME@umich.edu
#PBS -m abe

#print the time and date
date

#wait 10 seconds
sleep 10

#print the time and date again
date
```

Related topics

- **Job Submission** on page 2234

Job Files

TORQUE 4.5.0 was updated to accept XML-based job files in addition to the binary job files. The change allows job files to be more human-readable and easier to parse. Below is a sample job file in the new XML format:

```
<?xml version="1.0"?>
<job>
  <version>131842</version>
  <state>1</state>
  <substate>10</substate>
  <server_flags>33</server_flags>
  <start_time>0</start_time>
  <jobid>340</jobid>
  <fileprefix>340</fileprefix>
  <queue>batch</queue>
  <destination_queue></destination_queue>
  <record_type>1</record_type>
  <mom_address>0</mom_address>
  <mom_port>11</mom_port>
  <mom_rmport>0</mom_rmport>
  <attributes>
    <Job_Name flags="1">job2.sh</Job_Name>
    <Job_Owner flags="1">echan@moabServer.cn</Job_Owner>
    <job_state flags="1">Q</job_state>
    <queue flags="3">batch</queue>
    <server flags="1">company.com</server>
    <Checkpoint flags="1">u</Checkpoint>
    <ctime flags="1">1384292754</ctime>
    <Error_Path flags="1">moabServer.cn:/home/echan/work/job2.sh.e340</Error_Path>
    <Hold_Types flags="1">n</Hold_Types>
    <Join_Path flags="1">n</Join_Path>
    <Keep_Files flags="1">n</Keep_Files>
    <Mail_Points flags="1">a</Mail_Points>
    <mtime flags="1">1384292754</mtime>
    <Output_Path flags="1">moabServer.cn:/home/echan/work/job2.sh.o340</Output_Path>
    <Priority flags="1">0</Priority>
    <qtime flags="1">1384292754</qtime>
    <Rerunable flags="1">True</Rerunable>
    <Resource_List>
      <epilogue flags="1">/tmp/epilogue.sh</epilogue>
      <neednodes flags="1">moabServer:ppn=1</neednodes>
      <nodect flags="1">1</nodect>
      <nodes flags="1">moabServer:ppn=1</nodes>
    </Resource_List>
    <substate flags="1">10</substate>
    <Variable_List flags="1">PBS_O_QUEUE=batch
PBS_O_HOME=/home/echan
PBS_O_LOGNAME=echan
PBS_O_
PATH=/home/echan/eclipse:/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/
sbin:/usr/bin:/sbin:/bin:/usr/games:/opt/moab/bin:/opt/moab/sbin
PBS_O_SHELL=/bin/bash
PBS_O_LANG=en_US
PBS_O_WORKDIR=/home/echan/work
PBS_O_HOST=moabServer.cn
PBS_O_SERVER=moabServer
</Variable_List>
    <euser flags="1">echan</euser>
    <egroup flags="5">company</egroup>
    <hop_count flags="1">1</hop_count>
    <queue_rank flags="1">2</queue_rank>
    <queue_type flags="1">E</queue_type>
    <etime flags="1">1384292754</etime>
    <submit_args flags="1">-l nodes=lei:ppn=1 -l epilogue=/tmp/epilogue.sh
./job2.sh</submit_args>
    <fault_tolerant flags="1">False</fault_tolerant>
    <job_radix flags="1">0</job_radix>
```

```
    <submit_host flags="1">lei.ac</submit_host>
  </attributes>
</job>
```

The above job was submitted with this submit command:

```
qsub -l nodes=moabServer:ppn=1 -l epilogue=/tmp/epilogue.sh ./job2.sh
```

Related topics

- **Job Submission** on page 2234

Monitoring Jobs

TORQUE allows users and administrators to monitor submitted jobs with the **qstat** command. If the command is run by a non-administrative user, it will output just that user's jobs. For example:

```
> qstat
Job id           Name             User             Time Use S Queue
---------------- ---------------- ---------------- -------- - -----
4807             scatter          user01           12:56:34 R batch
...
```

Related topics

- **Submitting and Managing Jobs** on page 2233

Canceling Jobs

TORQUE allows users and administrators to cancel submitted jobs with the **qdel** command. The job will be sent TERM and KILL signals killing the running processes. When the top-level job script exits, the job will exit. The only parameter is the ID of the job to be canceled.

If a job is canceled by an operator or manager, an email notification will be sent to the user. Operators and managers may add a comment to this email with the `-m` option.

```
$ qstat
Job id           Name             User             Time Use S Queue
---------------- ---------------- ---------------- -------- - -----
4807             scatter          user01           12:56:34 R batch
...
$ qdel -m "hey! Stop abusing the NFS servers" 4807
$
```

Related topics

- **Submitting and Managing Jobs** on page 2233

Job Preemption

TORQUE supports job preemption by allowing authorized users to suspend and resume jobs. This is supported using one of two methods. If the node supports OS-level preemption, TORQUE will recognize that during the configure process and enable it. Otherwise, the MOM may be configured to launch a custom *checkpoint script* in order to support preempting a job. Using a custom checkpoint script requires that the job understand how to resume itself from a checkpoint after the preemption occurs.

Configuring a checkpoint script on a MOM

To configure the MOM to support a checkpoint script, the `$checkpoint_script` parameter must be set in the MOM's configuration file found in `TORQUE_HOME/mom_priv/config`. The checkpoint script should have execute permissions set. A typical configuration file might look as follows:

`mom_priv/config`:

```
$pbsserver          node06
$logevent           255
$restricted         *.mycluster.org
$checkpoint_script  /opt/moab/tools/mom-checkpoint.sh
```

The second thing that must be done to enable the checkpoint script is to change the value of `MOM_CHECKPOINT` to **1** in `/src/include/pbs_config.h`. (In some instances, MOM_CHECKPOINT may already be defined as 1.) The new line should be as follows:

`/src/include/pbs_config.h`:

```
#define MOM_CHECKPOINT 1
```

Related topics

- **Submitting and Managing Jobs** on page 2233

Keeping Completed Jobs

TORQUE provides the ability to report on the status of completed jobs for a configurable duration after the job has completed. This can be enabled by setting the **keep_completed** on page 2282 attribute on the job execution queue or the **keep_completed** on page 2424 parameter on the server. This should be set to the number of seconds that jobs should be held in the queue. If you set keep_completed on the job execution queue, completed jobs will be reported in the **C** state and the exit status is seen in the `exit_status` job attribute.

> If the Mother Superior and TORQUE server are on the same server, expect the following behavior:
>
> - When keep_completed is set, the job spool files will be deleted when the specified time arrives and TORQUE purges the job from memory.
> - When keep_completed is not set, TORQUE deletes the job spool files upon job completion.
> - If you manually purge a job (qdel -p) before the job completes or time runs out, TORQUE will never delete the spool files.

By maintaining status information about completed (or canceled, failed, etc.) jobs, administrators can better track failures and improve system performance. This allows TORQUE to better communicate with Moab Workload Manager and track the status of jobs. This gives Moab the ability to track specific failures and to schedule the workload around possible hazards. (See NODEFAILURERESERVETIME in **Appendix A: Moab Parameters** on page 902 for more information.)

Related topics

- **Submitting and Managing Jobs** on page 2233

Job Checkpoint and Restart

While TORQUE has had a job checkpoint and restart capability for many years, this was tied to machine specific features. Now TORQUE supports BLCR—an architecture independent package that provides for process checkpoint and restart.

> The support for BLCR is only for serial jobs, not for any MPI type jobs.

This section contains these topics:

Related topics

- **Submitting and Managing Jobs** on page 2233

Introduction to BLCR

BLCR is a kernel level package. It must be downloaded and installed from BLCR.

After building and making the package, it must be installed into the kernel with commands as follows. These can be installed into the file `/etc/modules` but all of the testing was done with explicit invocations of **modprobe**.

Installing BLCR into the kernel:

```
# /sbin/insmod /usr/local/lib/blcr/2.6.12-1.234/blcr_imports.ko
# /sbin/insmod /usr/local/lib/blcr/2.6.12-1.234/blcr_vmadump.ko
# /sbin/insmod /usr/local/lib/blcr/2.6.12-1.234/blcr.ko
```

The BLCR system provides four command line utilities:

- cr_checkpoint
- cr_info
- cr_restart
- cr_run

For more information about BLCR, see the *BLCR Administrator's Guide*.

Related topics

- **Job Checkpoint and Restart** on page 2252

Configuration Files and Scripts

Configuring and Building TORQUE for BLCR:

```
> ./configure --enable-unixsockets=no --enable-blcr
> make
> sudo make install
```

Depending on where BLCR is installed you may also need to use the following configure options to specify BLCR paths:

Option	Description
--with-blcr-include=*DIR*	include path for libcr.h
--with-blcr-lib=*DIR*	lib path for libcr
--with-blcr-bin=*DIR*	bin path for BLCR utilities

The pbs_mom configuration file located in `/var/spool/torque/mom_priv` must be modified to identify the script names associated with invoking the BLCR commands. The following variables should be used in the configuration file when using BLCR checkpointing.

Variable	Description
$checkpoint_interval	How often periodic job checkpoints will be taken (minutes)
$checkpoint_script	The name of the script file to execute to perform a job checkpoint
$restart_script	The name of the script file to execute to perform a job restart
$checkpoint_run_exe	The name of an executable program to be run when starting a checkpointable job (for BLCR, cr_run)

The following example shows the contents of the configuration file used for testing the BLCR feature in TORQUE.

> The script files below must be executable by the user. Be sure to use `chmod` to set the permissions to 754.

Example 4-15: Script file permissions

```
# chmod 754 blcr*
# ls -l
total 20
-rwxr-xr-- 1 root root 2112 2008-03-11 13:14 blcr_checkpoint_script
-rwxr-xr-- 1 root root 1987 2008-03-11 13:14 blcr_restart_script
-rw-r--r-- 1 root root 215 2008-03-11 13:13 config
drwxr-x--x 2 root root 4096 2008-03-11 13:21 jobs
-rw-r--r-- 1 root root 7 2008-03-11 13:15 mom.lock
```

Example 4-16: `mom_priv/config`

```
$checkpoint_script /var/spool/torque/mom_priv/blcr_checkpoint_script
$restart_script /var/spool/torque/mom_priv/blcr_restart_script
$checkpoint_run_exe /usr/local/bin/cr_run
$pbsserver makua.cridomain
$loglevel 7
```

Example 4-17: `mom_priv/blcr_checkpoint_script`

```
#! /usr/bin/perl
################################################################################
#
# Usage: checkpoint_script
#
# This script is invoked by pbs_mom to checkpoint a job.
#
################################################################################
use strict;
use Sys::Syslog;

# Log levels:
# 0 = none -- no logging
# 1 = fail -- log only failures
# 2 = info -- log invocations
# 3 = debug -- log all subcommands
my $logLevel = 3;

logPrint(2, "Invoked: $0 " . join(' ', @ARGV) . "\n");

my ($sessionId, $jobId, $userId, $signalNum, $checkpointDir, $checkpointName);
my $usage =
  "Usage: $0        \n";

# Note that depth is not used in this script but could control a limit to the number
of checkpoint
# image files that are preserved on the disk.
#
# Note also that a request was made to identify whether this script was invoked by the
job's
# owner or by a system administrator. While this information is known to pbs_server,
it
# is not propagated to pbs_mom and thus it is not possible to pass this to the script.

# Therefore, a workaround is to invoke qmgr and attempt to set a trivial variable.
# This will fail if the invoker is not a manager.

if (@ARGV == 7)
{
    ($sessionId, $jobId, $userId, $checkpointDir, $checkpointName, $signalNum $depth)
=
       @ARGV;
}
else { logDie(1, $usage); }

# Change to the checkpoint directory where we want the checkpoint to be created
chdir $checkpointDir
  or logDie(1, "Unable to cd to checkpoint dir ($checkpointDir): $!\n")
  if $logLevel;

my $cmd = "cr_checkpoint";
$cmd .= " --signal $signalNum" if $signalNum;
$cmd .= " --tree $sessionId";
$cmd .= " --file $checkpointName";
my $output = `$cmd 2>&1`;
my $rc = $? >> 8;
logDie(1, "Subcommand ($cmd) failed with rc=$rc:\n$output")
  if $rc && $logLevel >= 1;
logPrint(3, "Subcommand ($cmd) yielded rc=$rc:\n$output")
  if $logLevel >= 3;
exit 0;

################################################################################
# logPrint($message)
# Write a message (to syslog) and die
################################################################################
sub logPrint
{
```

```
    my ($level, $message) = @_;
    my @severity = ('none', 'warning', 'info', 'debug');

    return if $level > $logLevel;

    openlog('checkpoint_script', '', 'user');
    syslog($severity[$level], $message);
    closelog();
}

################################################################################
# logDie($message)
# Write a message (to syslog) and die
################################################################################
sub logDie
{
    my ($level, $message) = @_;
    logPrint($level, $message);
    die($message);
}
```

Example 4-18: *`mom_priv/blcr_restart_script`*

```
#! /usr/bin/perl
################################################################################
#
# Usage: restart_script
#
# This script is invoked by pbs_mom to restart a job.
#
################################################################################
use strict;
use Sys::Syslog;

# Log levels:
# 0 = none -- no logging
# 1 = fail -- log only failures
# 2 = info -- log invocations
# 3 = debug -- log all subcommands
my $logLevel = 3;

logPrint(2, "Invoked: $0 " . join(' ', @ARGV) . "\n");

my ($sessionId, $jobId, $userId, $checkpointDir, $restartName);
my $usage =
  "Usage: $0 \n";
if (@ARGV == 5)
{
    ($sessionId, $jobId, $userId, $checkpointDir, $restartName) =
       @ARGV;
}
else { logDie(1, $usage); }

# Change to the checkpoint directory where we want the checkpoint to be created
chdir $checkpointDir
  or logDie(1, "Unable to cd to checkpoint dir ($checkpointDir): $!\n")
  if $logLevel;

my $cmd = "cr_restart";
$cmd .= " $restartName";
my $output = `$cmd 2>&1`;
my $rc = $? >> 8;
logDie(1, "Subcommand ($cmd) failed with rc=$rc:\n$output")
  if $rc && $logLevel >= 1;
logPrint(3, "Subcommand ($cmd) yielded rc=$rc:\n$output")
  if $logLevel >= 3;
exit 0;

################################################################################
# logPrint($message)
# Write a message (to syslog) and die
################################################################################
sub logPrint
{
    my ($level, $message) = @_;
    my @severity = ('none', 'warning', 'info', 'debug');

    return if $level > $logLevel;
    openlog('restart_script', '', 'user');
    syslog($severity[$level], $message);
    closelog();
}

################################################################################
# logDie($message)
# Write a message (to syslog) and die
################################################################################
sub logDie
{
    my ($level, $message) = @_;

    logPrint($level, $message);
```

```
    die($message);
}
```

Related topics

- **Job Checkpoint and Restart** on page 2252

Starting a Checkpointable Job

Not every job is checkpointable. A job for which checkpointing is desirable must be started with the `-c` command line option. This option takes a comma-separated list of arguments that are used to control checkpointing behavior. The list of valid options available in the 2.4 version of TORQUE is show below.

Option	Description
none	No checkpointing (not highly useful, but included for completeness).
enabled	Specify that checkpointing is allowed, but must be explicitly invoked by either the `qhold` or `qchkpt` commands.
shutdown	Specify that checkpointing is to be done on a job at pbs_mom shutdown.
periodic	Specify that periodic checkpointing is enabled. The default interval is 10 minutes and can be changed by the `$checkpoint_interval` option in the MOM configuration file, or by specifying an interval when the job is submitted.
interval=minutes	Specify the checkpoint interval in minutes.
depth=number	Specify a number (depth) of checkpoint images to be kept in the checkpoint directory.
dir=path	Specify a checkpoint directory (default is `/var/spool/torque/checkpoint`).

Example 4-19: Sample test program

```
#include "stdio.h"
int main( int argc, char *argv[] )
{
int i;
         for (i=0; i<100; i++)
         {
                 printf("i = %d\n", i);
                 fflush(stdout);
                 sleep(1);
         }
}
```

Example 4-20: Instructions for building test program

```
> gcc -o test test.c
```

Example 4-21: Sample test script

```
#!/bin/bash ./test
```

Example 4-22: Starting the test job

```
> qstat
> qsub -c enabled,periodic,shutdown,interval=1 test.sh
77.jakaa.cridomain
> qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
77.jakaa                  test.sh          jsmith                 0 Q batch
>
```

If you have no scheduler running, you might need to start the job with **qrun**.

As this program runs, it writes its output to a file in `/var/spool/torque/spool`. This file can be observed with the command `tail -f`.

Related topics

- **Job Checkpoint and Restart** on page 2252

Checkpointing a Job

Jobs are checkpointed by issuing a **qhold** command. This causes an image file representing the state of the process to be written to disk. The directory by default is `/var/spool/torque/checkpoint`.

This default can be altered at the queue level with the qmgr command. For example, the command `qmgr -c set queue batch checkpoint_dir=/tmp` would change the checkpoint directory to `/tmp` for the queue 'batch'.

The default directory can also be altered at job submission time with the `-c dir=/tmp` command line option.

The name of the checkpoint directory and the name of the checkpoint image file become attributes of the job and can be observed with the command `qstat -f`. Notice in the output the names **checkpoint_dir** and **checkpoint_name**. The variable checkpoint_name is set when the image file is created and will not exist if no checkpoint has been taken.

A job can also be checkpointed without stopping or holding the job with the command **qchkpt**.

Related topics

- **Job Checkpoint and Restart** on page 2252

Restarting a Job

Restarting a job in the Held state

The **qrls** command is used to restart the hibernated job. If you were using the `tail -f` command to watch the output file, you will see the test program start counting again.

It is possible to use the **qalter** command to change the name of the checkpoint file associated with a job. This could be useful if there were several job checkpoints and it restarting the job from an older image was specified.

Restarting a job in the Completed state

In this case, the job must be moved to the Queued state with the **qrerun** command. Then the job must go to the Run state either by action of the scheduler or if there is no scheduler, through using the **qrun** command.

Related topics

- **Job Checkpoint and Restart** on page 2252

Acceptance Tests

A number of tests were made to verify the functioning of the BLCR implementation. See **Appendix M: BLCR Acceptance Tests** on page 2488 for a description of the testing.

Related topics

- **Job Checkpoint and Restart** on page 2252

Job Exit Status

Once a job under TORQUE has completed, the `exit_status` attribute will contain the result code returned by the job script. This attribute can be seen by submitting a `qstat -f` command to show the entire set of information associated with a job. The `exit_status` field is found near the bottom of the set of output lines.

Example 4-23: qstat -f (job failure)

```
Job Id: 179.host
    Job_Name = STDIN
    Job_Owner = user@host
    job_state = C
    queue = batchq server = host
    Checkpoint = u ctime = Fri Aug 29 14:55:55 2008
    Error_Path = host:/opt/moab/STDIN.e179
    exec_host = node1/0
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Fri Aug 29 14:55:55 2008
    Output_Path = host:/opt/moab/STDIN.o179
    Priority = 0
    qtime = Fri Aug 29 14:55:55 2008
    Rerunable = True Resource_List.ncpus = 2
    Resource_List.nodect = 1
    Resource_List.nodes = node1
    Variable_List = PBS_O_HOME=/home/user,PBS_O_LOGNAME=user,
 PBS_O_PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:,PBS_O_
SHELL=/bin/bash,PBS_O_HOST=host,
 PBS_O_WORKDIR=/opt/moab,PBS_O_QUEUE=batchq
    sched_hint = Post job file processing error; job 179.host on host node1/0Ba
 d UID for job execution REJHOST=pala.cridomain MSG=cannot find user 'user' in
password file
    etime = Fri Aug 29 14:55:55 2008
    exit_status = -1
```

The value of `Resource_List.*` is the amount of resources requested.

This code can be useful in diagnosing problems with jobs that may have unexpectedly terminated.

If TORQUE was unable to start the job, this field will contain a negative number produced by the pbs_mom. Otherwise, if the job script was successfully started, the value in this field will be the return value of the script.

Example 4-24: TORQUE supplied exit codes

Name	Value	Description
JOB_EXEC_OK	0	Job execution successful
JOB_EXEC_FAIL1	-1	Job execution failed, before files, no retry
JOB_EXEC_FAIL2	-2	Job execution failed, after files, no retry
JOB_EXEC_RETRY	-3	Job execution failed, do retry
JOB_EXEC_INITABT	-4	Job aborted on MOM initialization
JOB_EXEC_INITRST	-5	Job aborted on MOM init, chkpt, no migrate

Name	Value	Description
JOB_EXEC_INITRMG	-6	Job aborted on MOM init, chkpt, ok migrate
JOB_EXEC_BADRESRT	-7	Job restart failed
JOB_EXEC_CMDFAIL	-8	Exec() of user command failed
JOB_EXEC_STDOUTFAIL	-9	Could not create/open stdout stderr files
JOB_EXEC_OVERLIMIT_MEM	-10	Job exceeded a memory limit
JOB_EXEC_OVERLIMIT_WT	-11	Job exceeded a walltime limit
JOB_EXEC_OVERLIMIT_CPUT	-12	Job exceeded a CPU time limit

Example 4-25: Exit code from C program

```
$ cat error.c

#include
#include


int
main(int argc, char *argv)
{
   exit(256+11);
}


$ gcc -o error error.c


$ echo ./error | qsub
180.xxx.yyy


$ qstat -f
Job Id: 180.xxx.yyy
    Job_Name = STDIN
    Job_Owner = test.xxx.yyy
    resources_used.cput = 00:00:00
    resources_used.mem = 0kb
    resources_used.vmem = 0kb
    resources_used.walltime = 00:00:00
    job_state = C
    queue = batch
    server = xxx.yyy
    Checkpoint = u
    ctime = Wed Apr 30 11:29:37 2008
    Error_Path = xxx.yyy:/home/test/STDIN.e180
    exec_host = node01/0
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Wed Apr 30 11:29:37 2008
    Output_Path = xxx.yyy:/home/test/STDIN.o180
    Priority = 0
    qtime = Wed Apr 30 11:29:37 2008
    Rerunable = True
    Resource_List.neednodes = 1
    Resource_List.nodect = 1
    Resource_List.nodes = 1
    Resource_List.walltime = 01:00:00
    session_id = 14107
    substate = 59
    Variable_List = PBS_O_HOME=/home/test,PBS_O_LANG=en_US.UTF-8,
        PBS_O_LOGNAME=test,
        PBS_O_PATH=/usr/local/perltests/bin:/home/test/bin:/usr/local/s
        bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games,
        PBS_O_SHELL=/bin/bash,PBS_SERVER=xxx.yyy,
        PBS_O_HOST=xxx.yyy,PBS_O_WORKDIR=/home/test,
        PBS_O_QUEUE=batch
    euser = test
    egroup = test
    hashname = 180.xxx.yyy
    queue_rank = 8
    queue_type = E
    comment = Job started on Wed Apr 30 at 11:29
    etime = Wed Apr 30 11:29:37 2008
    exit_status = 11
    start_time = Wed Apr 30 11:29:37 2008
    start_count = 1
```

Notice that the C routine **exit** passes only the low order byte of its argument. In this case, 256+11 is really 267 but the resulting exit code is only 11 as seen in the output.

Related topics

- **Job Checkpoint and Restart** on page 2252
- **Submitting and Managing Jobs** on page 2233

Service Jobs

TORQUE service jobs are a special kind of job that is treated differently by TORQUE than normal batch jobs. TORQUE service jobs are *not* related to Moab's dynamic service jobs. A TORQUE service job cannot dynamically grow and shrink in size over time.

Jobs are marked as service jobs at the time they are submitted to Moab or TORQUE. Just like a normal job, a script file is specified with the job. In a batch job, the contents of the script file are taken by TORQUE and executed on the compute nodes. For a service job, however, the script file is assumed to respond to certain command-line arguments. Instead of just executing the script, TORQUE will use these command-line arguments to start, stop, and check on the status of the job. Listed below are the three command-line arguments that must be supported by any script submitted as part of a TORQUE service job:

- `start`: The script should take this argument and launch its service/workload. The script should remain executing/running until the service stops.
- `stop`: The script should take this argument and stop the service/workload that was earlier started.
- `status`: The script should take this argument and return, via standard out, either "running" if the service/workload is running as expected or "stopped" if the service is not running.

This feature was created with long-running services in mind. The command-line arguments should be familiar to users who interact with Unix services, as each of the service scripts found in `/etc/init.d/` also accept and respond to the arguments as explained above.

For example, if a user wants to start the Apache 2 server on a compute node, they can use a TORQUE service job and specify a script which will start, stop, and check on the status of the "httpd" daemon--possibly by using the already present `/etc/init.d/httpd` script.

> If you wish to submit service jobs only through TORQUE, no special version of Moab is required. If you wish to submit service jobs using Moab's msub, then Moab 5.4 is required.

For details, see these topics:

- **Submitting Service Jobs** on page 2267
- **Submitting Service Jobs in MCM** on page 2267
- **Managing Service Jobs** on page 2267

Submitting Service Jobs

There is a new option to **qsub**, "**-s**" which can take either a 'y' or 'n' (yes or no, respectively). When "-s y" is present, then the job is marked as a service job.

```
qsub -l walltime=100:00:00,nodes=1 -s y service_job.py
```

The example above submits a job to TORQUE with a walltime of 100 hours, one node, and it is marked as a service job. The script "service_job.py" will be used to start, stop, and check the status of the service/workload started on the compute nodes.

Moab, as of version 5.4, is able to accept the "-s y" option when msub is used for submission. Moab will then pass this information to TORQUE when the job is migrated.

Related topics

- **Service Jobs** on page 2266

Submitting Service Jobs in MCM

Submitting a service job in MCM requires the latest Adaptive Computing Suite snapshot of MCM. It also requires MCM to be started with the "`--future=2`" option.

Once MCM is started, open the **Create Workload** window and verify **Show Advanced Options** is checked. Notice that there is a **Service** checkbox that can be selected in the **Flags/Options** area. Use this to specify the job is a service job.

Related topics

- **Service Jobs** on page 2266

Managing Service Jobs

Managing a service job is done much like any other job; only a few differences exist.

Examining the job with **qstat** `-f` will reveal that the job has the `service = True` attribute. Non-service jobs will not make any mention of the "service" attribute.

Canceling a service job is done with **qdel**, mjobctl `-c`, or through any of the GUI's as with any other job. TORQUE, however, cancels the job by calling the service script with the "stop" argument instead of killing it directly. This behavior also occurs if the job runs over its wallclock and TORQUE/Moab is configured to cancel the job.

If a service job completes when the script exits after calling it with "start," or if TORQUE invokes the script with "status" and does not get back "running," it will *not* be terminated by using the "stop" argument.

Related topics

- **Service Jobs** on page 2266

Managing Nodes

This section contains information about adding and configuring compute nodes. It explains how to work with host security for systems that require dedicated access to compute nodes. It also contains information about scheduling specific cores on a node at job submission.

For details, see these topics:

- **Adding Nodes** on page 2268
- **Node Properties** on page 2269
- **Changing Node State** on page 2269
- **Host Security** on page 2272
- **Linux cpuset Support** on page 2274
- **Scheduling Cores** on page 2275

Adding Nodes

TORQUE can add and remove nodes either dynamically with **qmgr** or by manually editing the TORQUE_HOME/server_priv/nodes file (see **Initializing/Configuring TORQUE on the Server (pbs_server)** on page 2200).

Run-time node changes

TORQUE can dynamically add nodes with the qmgr command. For example, the following command will add node **node003**:

```
> qmgr -c "create node node003"
```

The above command appends the $TORQUE_HOME/server_priv/nodes file with:

```
node003
```

Nodes can also be removed with a similar command:

```
> qmgr -c "delete node node003"
```

> Typically, an administrator will want to change the state of a node instead of remove it (for details, see **Changing Node State** on page 2269).

> When you make changes to nodes – whether by using qmgr or directly editing the nodes file – you must restart pbs_server for those changes to take effect.

Related topics

- **Managing Nodes** on page 2268

Node Properties

TORQUE can associate properties with nodes to aid in identifying groups of nodes. It's typical for a site to conglomerate a heterogeneous set of resources. To identify the different sets, properties can be given to each node in a set. For example, a group of nodes that has a higher speed network connection could have the property "ib". TORQUE can set, update, or remove properties either dynamically with **qmgr** or by manually editing the `nodes` file.

Run-time node changes

TORQUE can dynamically change the properties of a node with the qmgr command. For example, note the following to give **node001** the properties of "bigmem" and "dualcore":

```
> qmgr -c "set node node001 properties = bigmem"
> qmgr -c "set node node001 properties += dualcore"
```

To relinquish a stated property, use the "-=" operator.

Manual node changes

The properties of each node are enumerated in `TORQUE_HOME/server_priv/nodes`. The feature(s) must be in a space delimited list after the node name. For example, to give **node001** the properties of "bigmem" and "dualcore" and **node002** the properties of "bigmem" and "matlab," edit the nodes file to contain the following:

`server_priv/nodes`:

```
node001 bigmem dualcore
node002 np=4 bigmem matlab
```

For changes to the nodes file to be activated, pbs_server must be restarted.

For a full description of this file, please see the *PBS Administrator Guide*.

Related topics

- **Job Submission** on page 2234
- **Managing Nodes** on page 2268

Changing Node State

A common task is to prevent jobs from running on a particular node by marking it *offline* with `pbsnodes -o nodename`. Once a node has been marked offline, the scheduler will no longer consider it available for new jobs. Simply use `pbsnodes -c nodename` when the node is returned to service.

Also useful is `pbsnodes -l`, which lists all nodes with an interesting state, such as down, unknown, or offline. This provides a quick glance at nodes that might be having a problem. (See **pbsnodes** for details.)

Related topics

- **Managing Nodes** on page 2268

Changing Node Power States

In TORQUE 5.0.1 and later, the pbsnodes -m command can modify the power state of nodes. Node cannot go from one low-power state to another low-power state. They must be brought up to the Running state and then moved to the new low-power state. The supported power states are:

State	Description
Running	• Physical machine is actively working • Power conservation is on a per-device basis • Processor power consumption controlled by P-states
Standby	• System appears off • Processor halted (OS executes a "halt" instruction) • Processor maintains CPU and system cache state • RAM refreshed to maintain memory state • Machine in low-power mode • Requires interrupt to exit state • Lowest-latency sleep state - has no effect on software
Suspend	• System appears off • Processor and support chipset have no power • OS maintains CPU, system cache, and support chipset state in memory • RAM in slow refresh • Machine in lowest-power state • Usually requires specific interrupt (keyboard, mouse) to exit state • Third lowest-latency sleep state - system must restore power to processor and support chipset
Hibernate	• System is off • Physical machine state and memory saved to disk • Requires restoration of power and machine state to exit state • Second highest-latency sleep state - system performs faster boot using saved machine state and copy of memory
Shutdown	• Equivalent to shutdown now command as root

In order to wake nodes and bring them up to a running state:

- the nodes must support, and be configured to use, Wake-on-LAN (WOL).
- the pbsnodes command must report the node's MAC address correctly.

To configure nodes to use Wake-on-LAN

1. Enable WOL in the BIOS for each node. If needed, contact your hardware manufacturer for details.
2. Use the ethtool command to determine what types of WOL packets your hardware supports. TORQUE uses the `g` packet. If the `g` packet is not listed, you cannot use WOL with TORQUE.

```
[root]# ethtool eth0
Settings for eth0:
       Supported ports: [ TP ]
       Supported link modes:   10baseT/Half 10baseT/Full
                               100baseT/Half 100baseT/Full
                               1000baseT/Full
       Supported pause frame use: No
       Supports auto-negotiation: Yes
       Advertised link modes:  10baseT/Half 10baseT/Full
                               100baseT/Half 100baseT/Full
                               1000baseT/Full
       Advertised pause frame use: No
       Advertised auto-negotiation: Yes
       Speed: 100Mb/s
       Duplex: Full
       Port: Twisted Pair
       PHYAD: 2
       Transceiver: internal
       Auto-negotiation: on
       MDI-X: off
       Supports Wake-on: pumbg
       Wake-on: p
        Current message level: 0x00000007 (7)
                              drv probe link
       Link detected: yes
```

This Ethernet interface supports the `g` WOL packet, but is currently set to use the `p` packet.

3. If your Ethernet interface supports the `g` packet, but is configured for a different packet, use ethtool -s *<interface>* wol g to configure it to use `g`.

```
[root]# ethtool -s eth0 wol g
[root]# ethtool eth0
Settings for eth0:
        Supported ports: [ TP ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Speed: 100Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 2
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: off
        Supports Wake-on: pumbg
        Wake-on: g
        Current message level: 0x00000007 (7)
                               drv probe link
        Link detected: yes
```

Now the power state of your nodes can be modified and they can be woken up from power-saving states.

Related topics

- **pbsnodes** on page 2356

Host Security

Enabling PAM with TORQUE

TORQUE is able to take advantage of the authentication services provided through Pluggable Authentication Modules (PAM) to help administrators manage access to compute nodes by users. The PAM module available in TORQUE is located in the PAM security directory. This module, when used in conjunction with other PAM modules, restricts access to the compute node unless the user has a job currently running on the node. The following configurations are examples only. For more information about PAM, see the PAM (Pluggable Authentication Modules) documentation from LinuxDocs.

To enable TORQUE PAM configure TORQUE using the `--with-pam` option. Using `--with-pam` is sufficient but if your PAM security modules are not in the default `/lib/security` or `/lib64/security` directory, you can specify the location using `--with-pam=`*<DIR>* where *<DIR>* is the directory where you want the modules to be installed. When TORQUE is installed the files `pam_pbssimpleauth.la` and `pam_pbssimpleauth.so` appear in `/lib/security`, `/lib64/security`, or the directory designated on the configuration line.

PAM is very flexible and policies vary greatly from one site to another. The following example restricts users trying to access a node using SSH. Administrators need to assess their own installations and decide how to apply the TORQUE PAM restrictions.

In this example, after installing TORQUE with PAM enabled, you would add the following two lines to /etc/pam.d/sshd:

```
account required pam_pbssimpleauth.so
account required pam_access.so
```

In /etc/security/access.conf make sure all users who access the compute node are added to the configuration. This is an example which allows the users root, george, allen, and michael access.

```
-:ALL EXCEPT root george allen michael torque:ALL
```

With this configuration, if user george has a job currently running on the compute node, george can use ssh to login to the node. If there are currently no jobs running, george is disconnected when attempting to login.

TORQUE PAM is good at keeping users out who do not have jobs running on a compute node. However, it does not have the ability to force a user to log out once they are in. To accomplish this use epilogue or prologue scripts to force users off the system.

Legacy TORQUE PAM configuration

There is an alternative PAM configuration for TORQUE that has been available since 2006. It can be found in the `contrib/pam_authuser` directory of the source tree. Adaptive Computing does not currently support this method but the instructions are given here for those who are currently using it and for those who wish to use it.

For systems requiring dedicated access to compute nodes (for example, users with sensitive data), TORQUE prologue and epilogue scripts provide a vehicle to leverage the authentication provided by linux-PAM modules. (See **Appendix G: Prologue and Epilogue Scripts** on page 2469 for more information.)

To allow only users with running jobs (and root) to access compute nodes

1. Untar `contrib/pam_authuser.tar.gz` (found in the src tar ball).
2. Compile `pam_authuser.c` with `make` and `make install` on every compute node.
3. Edit `/etc/system-auth` as described in `README.pam_authuser`, again on every compute node.
4. Either make a tarball of the epilogue* and prologue* scripts (to preserve the symbolic link) and untar it in the `mom_priv` directory, or just copy epilogue* and prologue* to `mom_priv/`.

The prologue* scripts are Perl scripts that add the user of the job to /etc/authuser. The epilogue* scripts then remove the first occurrence of the user from /etc/authuser. File locking is employed in all scripts to eliminate the chance of race conditions. There is also some commented code in the epilogue* scripts, which, if uncommented, kills all processes owned by the user (using pkill), provided that the user doesn't have another valid job on the same node.

prologue and **epilogue** scripts were added to the `pam_authuser` tarball in version 2.1 of TORQUE.

Related topics

- **Managing Nodes** on page 2268

Linux cpuset Support

cpuset overview

Linux kernel 2.6 cpusets are logical, hierarchical groupings of CPUs and units of memory. Once created, individual processes can be placed within a cpuset. The processes will only be allowed to run/access the specified CPUs and memory. cpusets are managed in a virtual file system mounted at `/dev/cpuset`. New cpusets are created by simply making new directories. cpusets gain CPUs and memory units by simply writing the unit number to files within the cpuset.

cpuset support

> All nodes using cpusets must have the `hwloc` library version 1.2 or higher installed.

When started, pbs_mom will create an initial top-level cpuset at `/dev/cpuset/torque`. This cpuset contains all CPUs and memory of the host machine. If this "torqueset" already exists, it will be left unchanged to allow the administrator to override the default behavior. All subsequent cpusets are created within the torqueset.

When a job is started, the jobset is created at `/dev/cpuset/torque/$jobid` and populated with the CPUs listed in the `exec_host job` attribute. Also created are individual tasksets for each CPU within the jobset. This happens before prologue, which allows it to be easily modified, and it happens on all nodes.

The top-level batch script process is executed in the jobset. Tasks launched through the TM interface (pbsdsh and PW's mpiexec) will be executed within the appropriate taskset.

On job exit, all tasksets and the jobset are deleted.

cpuset configuration

To configure cpuset

1. As root, mount the virtual filesystem for cpusets:

```
mkdir /dev/cpuset
mount -t cpuset none /dev/cpuset
```

> Do this for each MOM that is to use cpusets.

2. Because cpuset usage is a build-time option in TORQUE, you must add `--enable-cpuset` to your configure options:

```
./configure --enable-cpuset
```

3. Use this configuration for the MOMs across your system.

cpuset advantages / disadvantages

Presently, any job can request a single CPU and proceed to use everything available in the machine. This is occasionally done to circumvent policy, but most often is simply an error on the part of the user. cpuset support will easily constrain the processes to not interfere with other jobs.

Jobs on larger NUMA systems may see a performance boost if jobs can be intelligently assigned to specific CPUs. Jobs may perform better if striped across physical processors, or contained within the fewest number of memory controllers.

TM tasks are constrained to a single core, thus a multi-threaded process could seriously suffer.

Related topics

- **Managing Nodes** on page 2268
- **Geometry Request Configuration** on page 2275

Scheduling Cores

In TORQUE 2.4 and later, you can request specific cores on a node at job submission by using geometry requests. To use this feature, specify the **procs_bitmap** resource request of `qsub-l` (see **qsub**) at job submission.

For details about scheduling cores, see these topics:

- **Geometry Request Configuration** on page 2275
- **Geometry Request Usage** on page 2276
- **Geometry Request Considerations** on page 2276

Geometry Request Configuration

A Linux kernel of 2.6 or later is required to use geometry requests, because this feature uses Linux cpusets in its implementation. In order to use this feature, the cpuset directory has to be mounted. For more information on how to mount the cpuset directory, see **Linux cpuset Support** on page 2274. If the operating environment is suitable for geometry requests, configure TORQUE with the `--enable-geometry-requests` option.

```
> ./configure --prefix=/home/john/torque --enable-geometry-requests
```

TORQUE is configured to install to `/home/john/torque` and to enable the geometry requests feature.

> The geometry request feature uses a subset of the cpusets feature. When you configure TORQUE using `--enable-cpuset` and `--enable-geometry-requests` at the same time, and use `-l procs_bitmap=X`, the job will get the requested cpuset. Otherwise, the job is treated as if only `--enable-cpuset` was configured.

Related topics

- **Scheduling Cores** on page 2275

Geometry Request Usage

Once enabled, users can submit jobs with a geometry request by using the `procs_bitmap=`*<string>* resource request. **procs_bitmap** requires a numerical string made up of 1's and 0's. A 0 in the bitmap means the job cannot run on the core that matches the 0's index in the bitmap. The index is in reverse order of the number of cores available. If a job is submitted with `procs_bitmap=1011`, then the job requests a node with four free cores, and uses only cores one, two, and four.

> The geometry request feature requires a node that has all cores free. A job with a geometry request cannot run on a node that has cores that are busy, even if the node has more than enough cores available to run the job.

```
qsub -l procs_bitmap=0011 ossl.sh
```

The job **ossl.sh** is submitted with a geometry request of **0011**.

In the above example, the submitted job can run only on a node that has four cores. When a suitable node is found, the job runs exclusively on cores one and two.

Related topics

- **Scheduling Cores** on page 2275

Geometry Request Considerations

As previously stated, jobs with geometry requests require a node with all of its cores available. After the job starts running on the requested cores, the node cannot run other jobs, even if the node has enough free cores to meet the requirements of the other jobs. Once the geometry requesting job is done, the node is available to other jobs again.

Related topics

- **Scheduling Cores** on page 2275

Scheduling Accelerator Hardware

TORQUE works with accelerators (such as NVIDIA GPUs and Intel MICs) and can collect and report metrics from them or submit workload to them. This feature requires the use of the Moab scheduler. See **Accelerators** on page 794 for information on configuring accelerators in TORQUE.

Setting Server Policies

This section explains how to set up and configure your queue. It lists the queue attributes and describes how to set up a routing queue. This section also explains how to set up TORQUE to run in high availability mode. For details, see these topics:

- **Queue Configuration** on page 2277
- **Server High Availability** on page 2291

Queue Configuration

Under TORQUE, queue configuration is accomplished using the **Server High Availability** command. With this tool, the first step is to create the queue. This is accomplished using the `create` subcommand of qmgr as in the following example:

```
> qmgr -c "create queue batch queue_type=execution"
```

Once created, the queue must be configured to be operational. At a minimum, this includes setting the options **started** and **enabled**. Further configuration is possible using any combination of the attributes listed in what follows.

For Boolean attributes, *T*, *t*, *1*, *Y*, and *y* are all synonymous with "TRUE," and *F*, *f*, *0*, *N*, and *n* all mean "FALSE."

For **queue_type**, *E* and *R* are synonymous with "Execution" and "Routing" (respectively).

See these topics for more details:

- **Queue Attributes** on page 2277
- **Example Queue Configuration** on page 2288
- **Setting a Default Queue** on page 2289
- **Mapping a Queue to Subset of Resources** on page 2289
- **Creating a Routing Queue** on page 2289

Related topics

- **Appendix B: Server Parameters** on page 2417
- **qalter** on page 2360 - command which can move jobs from one queue to another

Queue Attributes

This section lists the following queue attributes:

- **acl_groups** on page 2278
- **acl_group_enable** on page 2279
- **acl_group_sloppy** on page 2279

This section also lists some queue resource limits (see **Assigning queue resource limits** on page 2287).

> For Boolean attributes, *T*, *t*, *1*, *Y*, and *y* are all synonymous with "TRUE," and *F*, *f*, *0*, *N*, and *n* all mean "FALSE."

acl_groups	
Format	<GROUP>[@<HOST>][+<USER>[@<HOST>]]...
Default	---

acl_groups

Description	Specifies the list of groups which may submit jobs to the queue. If acl_group_enable is set to true, only users with a primary group listed in acl_groups may utilize the queue. If the PBSACLUSEGROUPLIST variable is set in the pbs_server environment, acl_groups checks against all groups of which the job user is a member.
Example	`> qmgr -c "set queue batch acl_groups=staff"` `> qmgr -c "set queue batch acl_groups+=ops@h1"` `> qmgr -c "set queue batch acl_groups+=staff@h1"` Used in conjunction with **acl_group_enable**.

acl_group_enable

Format	<BOOLEAN>
Default	FALSE
Description	If **TRUE**, constrains TORQUE to only allow jobs submitted from groups specified by the **acl_groups** parameter.
Example	`qmgr -c "set queue batch acl_group_enable=true"`

acl_group_sloppy

Format	<BOOLEAN>
Default	FALSE
Description	If **TRUE**, **acl_groups** will be checked against all groups of which the job users is a member.
Example	---

acl_hosts

Format	<HOST>[+<HOST>]...
Default	---

acl_hosts	
Description	Specifies the list of hosts that may submit jobs to the queue.
Example	`qmgr -c "set queue batch acl_hosts=h1+h1+h1"` Used in conjunction with **acl_host_enable**.

acl_host_enable	
Format	<BOOLEAN>
Default	FALSE
Description	If **TRUE**, constrains TORQUE to only allow jobs submitted from hosts specified by the **acl_hosts** parameter.
Example	`qmgr -c "set queue batch acl_host_enable=true"`

acl_logic_or	
Format	<BOOLEAN>
Default	FALSE
Description	If **TRUE**, user and group acls are logically OR'd together, meaning that either acl may be met to allow access. If FALSE or unset, then both acls are AND'd, meaning that both acls must be satisfied.
Example	`qmgr -c "set queue batch acl_logic_or=true"`

acl_users	
Format	<USER>[@<HOST>][+<USER>[@<HOST>]]...
Default	---
Description	Specifies the list of users who may submit jobs to the queue. If **acl_user_enable** is set to **TRUE**, only users listed in acl_users may use the queue.

acl_users	
Example	`> qmgr -c "set queue batch acl_users=john"` `> qmgr -c "set queue batch acl_users+=steve@h1"` `> qmgr -c "set queue batch acl_users+=stevek@h1"` Used in conjunction with **acl_user_enable**.

acl_user_enable	
Format	<BOOLEAN>
Default	FALSE
Description	If **TRUE**, constrains TORQUE to only allow jobs submitted from users specified by the **acl_users** parameter.
Example	`qmgr -c "set queue batch acl_user_enable=true"`

disallowed_types	
Format	<type>[+<type>]...
Default	---
Description	Specifies classes of jobs that are not allowed to be submitted to this queue. Valid types are interactive, batch, rerunable, nonrerunable, fault_tolerant (as of version 2.4.0 and later), fault_intolerant (as of version 2.4.0 and later), and job_array (as of version 2.4.1 and later).
Example	`qmgr -c "set queue batch disallowed_types = interactive"` `qmgr -c "set queue batch disallowed_types += job_array"`

enabled	
Format	<BOOLEAN>
Default	FALSE
Description	Specifies whether the queue accepts new job submissions.

enabled	
Example	`qmgr -c "set queue batch enabled=true"`

features_required	
Format	feature1[feature2[,feature3...]]
Default	---
Description	Specifies that all jobs in this queue will require these features in addition to any they may have requested. A feature is a synonym for a property.
Example	`qmgr -c 's q batch features_required=fast'`

keep_completed	
Format	<INTEGER>
Default	0
Description	Specifies the number of seconds jobs should be held in the Completed state after exiting. For more information, see **Keeping Completed Jobs** on page 2251.
Example	`qmgr -c "set queue batch keep_completed=120"`

kill_delay	
Format	<INTEGER>
Default	2
Description	Specifies the number of seconds between sending a SIGTERM and a SIGKILL to a job in a specific queue that you want to cancel. It is possible that the job script, and any child processes it spawns, can receive several SIGTERM signals before the SIGKILL signal is received. All MOMs must be configured with `$exec_with_exec true` in order for **kill_delay** to work, even when relying on default **kill_delay** settings.

kill_delay	
Example	`qmgr -c "set queue batch kill_delay=30"`

max_queuable	
Format	<INTEGER>
Default	unlimited
Description	Specifies the maximum number of jobs allowed in the queue at any given time (includes idle, running, and blocked jobs).
Example	`qmgr -c "set queue batch max_queuable=20"`

max_running	
Format	<INTEGER>
Default	unlimited
Description	Specifies the maximum number of jobs in the queue allowed to run at any given time.
Example	`qmgr -c "set queue batch max_running=20"`

max_user_queuable	
Format	<INTEGER>
Default	unlimited
Description	Specifies the maximum number of jobs, per user, allowed in the queue at any given time (includes idle, running, and blocked jobs). Version 2.1.3 and greater.
Example	`qmgr -c "set queue batch max_user_queuable=20"`

max_user_run	
Format	<INTEGER>
Default	unlimited
Description	Specifies the maximum number of jobs, per user, in the queue allowed to run at any given time.
Example	`qmgr -c "set queue batch max_user_run=10"`

priority	
Format	<INTEGER>
Default	0
Description	Specifies the priority value associated with the queue.
Example	`qmgr -c "set queue batch priority=20"`

queue_type	
Format	One of *e*, *execution*, *r*, or *route* (see **Creating a Routing Queue** on page 2289)
Default	---
Description	Specifies the queue type. This value must be explicitly set for all queues.
Example	`qmgr -c "set queue batch queue_type=execution"`

required_login_property	
Format	<STRING>
Default	---

required_login_property	
Description	Adds the specified login property as a requirement for all jobs in this queue.
Example	`qmgr -c 's q <queuename> required_login_property=INDUSTRIAL'`

resources_available	
Format	<STRING>
Default	---
Description	Specifies to cumulative resources available to all jobs running in the queue. See **qsub will not allow the submission of jobs requesting many processors** on page 2326 for more information.
Example	`qmgr -c "set queue batch resources_available.nodect=20"` You must restart pbs_server for changes to take effect. Also, resources_available is constrained by the smallest of queue.resources_available and server.resources_available.

resources_default	
Format	<STRING>
Default	---
Description	Specifies default resource requirements for jobs submitted to the queue.
Example	`qmgr -c "set queue batch resources_default.walltime=3600"`

resources_max	
Format	<STRING>
Default	---

resources_max

Description	Specifies the maximum resource limits for jobs submitted to the queue.
Example	`qmgr -c "set queue batch resources_max.nodect=16"`

resources_min

Format	<STRING>
Default	---
Description	Specifies the minimum resource limits for jobs submitted to the queue.
Example	`qmgr -c "set queue batch resources_min.nodect=2"`

route_destinations

Format	<queue>[@<host>]
Default	---
Description	Specifies the potential destination queues for jobs submitted to the associated routing queue. This attribute is only valid for routing queues (see **Creating a Routing Queue** on page 2289).
Example	`> qmgr -c "set queue route route_destinations=fast"` `> qmgr -c "set queue route route_destinations+=slow"` `> qmgr -c "set queue route route_destinations+=medium@hostname"` To set multiple queue specifications, use multiple commands: `> qmgr -c 's s route_destinations=batch'` `> qmgr -c 's s route_destinations+=long'` `> qmgr -c 's s route_destinations+=short'`

started

Format	<BOOLEAN>

started	
Default	FALSE
Description	Specifies whether jobs in the queue are allowed to execute.
Example	`qmgr -c "set queue batch started=true"`

Assigning queue resource limits

Administrators can use resources limits to help direct what kind of jobs go to different queues. There are four queue attributes where resource limits can be set: **resources_available**, **resources_default**, **resources_max**, and **resources_min**. The list of supported resources that can be limited with these attributes are *arch*, *mem*, *nodect*, *nodes*, *procct*, *pvmem*, *vmem*, and *walltime*.

Resource	Format	Description
arch	string	Specifies the administrator defined system architecture required.
mem	size	Amount of physical memory used by the job. (Ignored on Darwin, Digital Unix, Free BSD, HPUX 11, IRIX, NetBSD, and SunOS. Also ignored on Linux if number of nodes is not 1. Not implemented on AIX and HPUX 10.)
ncpus	integer	Sets the number of processors in one task where a task cannot span nodes. You cannot request both ncpus and nodes in the same queue.
nodect	integer	Sets the number of nodes available. By default, TORQUE will set the number of nodes available to the number of nodes listed in the `$TORQUE_HOME/server_priv/nodes` file. nodect can be set to be greater than or less than that number. Generally, it is used to set the node count higher than the number of physical nodes in the cluster.
nodes	integer	Specifies the number of nodes.
procct	integer	Sets limits on the total number of execution slots (procs) allocated to a job. The number of procs is calculated by summing the products of all node and ppn entries for a job. For example `qsub -l nodes=2:ppn=2+3:ppn=4 job.sh` would yield a procct of 16. 2*2 (2:ppn=2) + 3*4 (3:ppn=4).
pvmem	size	Amount of virtual memory used by any single process in a job.

Resource	Format	Description
vmem	size	Amount of virtual memory used by all concurrent processes in the job.
walltime	seconds, or [[HH:]MM:]SS	Amount of real time during which a job can be in a running state.

size

The size format specifies the maximum amount in terms of bytes or words. It is expressed in the form *integer[suffix]*. The suffix is a multiplier defined in the following table ("b" means bytes [the default] and "w" means words). The size of a word is calculated on the execution server as its word size.

Suffix		Multiplier
b	w	1
kb	kw	1024
mb	mw	1,048,576
gb	gw	1,073,741,824
tb	tw	1,099,511,627,776

Related topics

- **Queue Configuration** on page 2277
- **Example Queue Configuration** on page 2288

Example Queue Configuration

The following series of **qmgr** commands will create and configure a queue named batch:

```
qmgr -c "create queue batch queue_type=execution"
qmgr -c "set queue batch started=true"
qmgr -c "set queue batch enabled=true"
qmgr -c "set queue batch resources_default.nodes=1"
qmgr -c "set queue batch resources_default.walltime=3600"
```

This queue will accept new jobs and, if not explicitly specified in the job, will assign a nodecount of 1 and a walltime of 1 hour to each job.

Related topics

- **Queue Configuration** on page 2277

Setting a Default Queue

By default, a job must explicitly specify which queue it is to run in. To change this behavior, the server parameter **default_queue** may be specified as in the following example:

```
qmgr -c "set server default_queue=batch"
```

Related topics

- **Queue Configuration** on page 2277

Mapping a Queue to Subset of Resources

TORQUE does not currently provide a simple mechanism for mapping queues to nodes. However, schedulers such as Moab and Maui can provide this functionality.

The simplest method is using `default_resources.neednodes` on an execution queue, setting it to a particular node attribute. Maui/Moab will use this information to ensure that jobs in that queue will be assigned nodes with that attribute. For example, suppose we have some nodes bought with money from the chemistry department, and some nodes paid by the biology department.

```
$TORQUE_HOME/server_priv/nodes:
node01 np=2 chem
node02 np=2 chem
node03 np=2 bio
node04 np=2 bio
qmgr:
set queue chem resources_default.neednodes=chem
set queue bio resources_default.neednodes=bio
```

> This example does not preclude other queues from accessing those nodes. One solution is to use some other generic attribute with all other nodes and queues.

More advanced configurations can be made with standing reservations and QoSs.

Related topics

- **Queue Configuration** on page 2277

Creating a Routing Queue

A routing queue will steer a job to a destination queue based on job attributes and queue constraints. It is set up by creating a queue of **queue_type** "Route" with a **route_destinations** attribute set, as in the following example.

```
qmgr

# routing queue
create queue route
set queue route queue_type = Route
set queue route route_destinations = reg_64
set queue route route_destinations += reg_32
set queue route route_destinations += reg
set queue route enabled = True
set queue route started = True

# queue for jobs using 1-15 nodes
create queue reg
set queue reg queue_type = Execution
set queue reg resources_min.ncpus = 1
set queue reg resources_min.nodect = 1
set queue reg resources_default.ncpus = 1
set queue reg resources_default.nodes = 1
set queue reg enabled = True
set queue reg started = True

# queue for jobs using 16-31 nodes
create queue reg_32
set queue reg_32 queue_type = Execution
set queue reg_32 resources_min.ncpus = 31
set queue reg_32 resources_min.nodes = 16
set queue reg_32 resources_default.walltime = 12:00:00
set queue reg_32 enabled = True
set queue reg_32 started = True

# queue for jobs using 32+ nodes
create queue reg_64
set queue reg_64 queue_type = Execution
set queue reg_64 resources_min.ncpus = 63
set queue reg_64 resources_min.nodes = 32
set queue reg_64 resources_default.walltime = 06:00:00
set queue reg_64 enabled = True
set queue reg_64 started = True

# have all jobs go through the routing queue
set server default_queue = batch
set server resources_default.ncpus = 1
set server resources_default.walltime = 24:00:00
  ...
```

In this example, the compute nodes are dual processors and default walltimes are set according to the number of processors/nodes of a job. Jobs with 32 nodes (63 processors) or more will be given a default walltime of 6 hours. Also, jobs with 16-31 nodes (31-62 processors) will be given a default walltime of 12 hours. All other jobs will have the server default walltime of 24 hours.

The ordering of the route_destinations is important. In a routing queue, a job is assigned to the first possible destination queue based on the **resources_max**, **resources_min**, **acl_users**, and **acl_groups** attributes. In the preceding example, the attributes of a single processor job would first be checked against the reg_64 queue, then the reg_32 queue, and finally the reg queue.

Adding the following settings to the earlier configuration elucidates the queue resource requirements:

```
qmgr

set queue reg resources_max.ncpus = 30
set queue reg resources_max.nodect = 15
set queue reg_16 resources_max.ncpus = 62
set queue reg_16 resources_max.nodect = 31
```

The time of enforcement of server and queue defaults is important in this example. TORQUE applies server and queue defaults differently in job centric and queue centric modes. For job centric mode, TORQUE waits to apply the server and queue defaults until the job is assigned to its final execution queue. For queue centric mode, it enforces server defaults before it is placed in the routing queue. In either mode, queue defaults override the server defaults. TORQUE defaults to job centric mode. To set queue centric mode, set `queue_centric_limits`, as in what follows:

```
qmgr

set server queue_centric_limits = true
```

An artifact of job centric mode is that if a job does not have an attribute set, the server and routing queue defaults are not applied when queue resource limits are checked. Consequently, a job that requests 32 nodes (not ncpus=32) will not be checked against a min_resource.ncpus limit. Also, for the preceding example, a job without any attributes set will be placed in the reg_64 queue, since the server ncpus default will be applied after the job is assigned to an execution queue.

> Routine queue defaults are not applied to job attributes in versions 2.1.0 and before.

> If the error message "`qsub: Job rejected by all possible destinations`" is reported when submitting a job, it may be necessary to add queue location information, (i.e., in the routing queue's **route_destinations** attribute, change "batch" to "batch@localhost").

Related topics

- **Queue Configuration** on page 2277
- **Queue Attributes** on page 2277

Server High Availability

You can now run TORQUE in a redundant or high availability mode. This means that there can be multiple instances of the server running and waiting to take over processing in the event that the currently running server fails.

> The high availability feature is available in the 2.3 and later versions of TORQUE. TORQUE 2.4 includes several enhancements to high availability (see **Server High Availability** on page 2291).

> Contact Adaptive Computing before attempting to implement any type of high availability.

For more details, see these sections:

- **Redundant server host machines** on page 2292
- **Server High Availability** on page 2291
- **Enhanced high availability with Moab** on page 2293
- **How commands select the correct server host** on page 2294

Redundant server host machines

High availability enables Moab HPC Suite to continue running even if pbs_server is brought down. This is done by running multiple copies of pbs_server which have their `torque/server_priv` directory mounted on a shared file system.

> Do not use symbolic links when sharing the TORQUE home directory or server_priv directories. A workaround for this is to use `mount --rbind /path/to/share /var/spool/torque`. Also, it is highly recommended that you only share the server_priv and not the entire *$TORQUEHOMEDIR*.

The `torque/server_name` must include the host names of all nodes that run pbs_server. All MOM nodes also must include the host names of all nodes running pbs_server in their `torque/server_name` file. The syntax of the `torque/server_name` is a comma delimited list of host names.

For example:

```
host1,host2,host3
```

> When configuring high availability, do not use $pbsserver to specify the host names. You must use the *$TORQUEHOMEDIR*`/server_name` file.

All instances of pbs_server need to be started with the `--ha` command line option that allows the servers to run at the same time. Only the first server to start will complete the full startup. The second server to start will block very early in the startup when it tries to lock the file `torque/server_priv/server.lock`. When the second server cannot obtain the lock, it will spin in a loop and wait for the lock to clear. The sleep time between checks of the lock file is one second.

Notice that not only can the servers run on independent server hardware, there can also be multiple instances of the pbs_server running on the same machine. This was not possible before as the second one to start would always write an error and quit when it could not obtain the lock.

Enabling high availability

To use high availability, you must start each instance of pbs_server with the `--ha` option.

Prior to version 4.0, TORQUE with HA was configured with an `--enable-high-availability` option. That option is no longer required.

Three server options help manage high availability. The server parameters are **lock_file**, **lock_file_update_time**, and **lock_file_check_time**.

The lock_file option allows the administrator to change the location of the lock file. The default location is `torque/server_priv`. If the lock_file option is used, the new location must be on the shared partition so all servers have access.

The lock_file_update_time and lock_file_check_time parameters are used by the servers to determine if the primary server is active. The primary pbs_server will update the lock file based on the lock_file_update_time (default value of 3 seconds). All backup pbs_servers will check the lock file as indicated by the lock_file_check_time parameter (default value of 9 seconds). The lock_file_update_time must be less than the lock_file_check_time. When a failure occurs, the backup pbs_server takes up to the lock_file_check_time value to take over.

```
> qmgr -c "set server lock_file_check_time=5"
```

In the above example, after the primary pbs_server goes down, the backup pbs_server takes up to 5 seconds to take over. It takes additional time for all MOMs to switch over to the new pbs_server.

> The clock on the primary and redundant servers must be synchronized in order for high availability to work. Use a utility such as NTP to ensure your servers have a synchronized time.

> Do not use anything but a plain simple NFS fileshare that is not used by anybody or anything else (i.e., only Moab can use the fileshare).

> Do not use any general-purpose NAS, do not use any parallel file system, and do not use company-wide shared infrastructure to set up Moab high availability using "native" high availability.

Enhanced high availability with Moab

When TORQUE is run with an external scheduler such as Moab, and the pbs_server is not running on the same host as Moab, pbs_server needs to know where to find the scheduler. To do this, use the `-l` option as demonstrated in the example below (the port is required and the default is 15004).

```
> pbs_server -l <moabhost:port>
```

If Moab is running in HA mode, add a `-l` option for each redundant server.

```
> pbs_server -l <moabhost1:port> -l <moabhost2:port>
```

If pbs_server and Moab run on the same host, use the `--ha` option as demonstrated in the example below.

```
> pbs_server --ha
```

The root user of each Moab host must be added to the **operators** and **managers** lists of the server. This enables Moab to execute root level operations in TORQUE.

How commands select the correct server host

The various commands that send messages to pbs_server usually have an option of specifying the server name on the command line, or if none is specified will use the default server name. The default server name comes either from the environment variable *PBS_DEFAULT* or from the file `torque/server_name`.

When a command is executed and no explicit server is mentioned, an attempt is made to connect to the first server name in the list of hosts from PBS_DEFAULT or `torque/server_name`. If this fails, the next server name is tried. If all servers in the list are unreachable, an error is returned and the command fails.

Note that there is a period of time after the failure of the current server during which the new server is starting up where it is unable to process commands. The new server must read the existing configuration and job information from the disk, so the length of time that commands cannot be received varies. Commands issued during this period of time might fail due to timeouts expiring.

Job names

Job names normally contain the name of the host machine where pbs_server is running. When job names are constructed, only the server name in *$PBS_DEFAULT* or the first name from the server specification list, `$TORQUE_HOME/server_name`, is used in building the job name.

Persistence of the pbs_server process

The system administrator must ensure that pbs_server continues to run on the server nodes. This could be as simple as a *cron* job that counts the number of pbs_server's in the process table and starts some more if needed.

High availability of the NFS server

> Before installing a specific NFS HA solution please contact Adaptive Computing Support for a detailed discussion on NFS HA type and implementation path.

One consideration of this implementation is that it depends on NFS file system also being redundant. NFS can be set up as a redundant service. See the following.

- Setting Up A Highly Available NFS Server
- Making NFS Work On Your Network
- Sourceforge Linux NFS FAQ
- NFS v4 main site

There are also other ways to set up a shared file system. See the following:

- Red Hat Global File System
- Data sharing with a GFS storage cluster

Installing TORQUE in high availability mode

The following procedure demonstrates a TORQUE installation in high availability (HA) mode.

Requirements

- gcc (GCC) 4.1.2
- BASH shell
- Servers configured the following way:
    - 2 main servers with identical architecture:
        - `server1` — Primary server running TORQUE with a shared file system (this example uses NFS)
        - `server2` — Secondary server running with TORQUE with a shared file system (this example uses NFS)
    - `fileServer` — Shared file system (this example uses NFS)
    - Compute nodes

To install TORQUE in HA mode

1. Stop all firewalls or update your firewall to allow traffic from TORQUE services.

   ```
   > service iptables stop
   > chkconfig iptables off
   ```

   If you are unable to stop the firewall due to infrastructure restriction, open the following ports:

   - `15001[tcp,udp]`
   - `15002[tcp,udp]`
   - `15003[tcp,udp]`

2. Disable SELinux

   ```
   > vi /etc/sysconfig/selinux

   SELINUX=disabled
   ```

3. Update your main `~/.bashrc` profile to ensure you are always referencing the applications to be installed on all servers.

   ```
   # TORQUE
   export TORQUEHOME=/var/spool/torque

   # Library Path

   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${TORQUEHOME}/lib

   # Update system paths
   export PATH=${TORQUEHOME}/bin:${TORQUEHOME}/sbin:$ {PATH}
   ```

4. Verify `server1` and `server2` are resolvable via either DNS or looking for an entry in the `/etc/hosts` file.

5. Configure the NFS Mounts by following these steps:

   a. Create mount point folders on `fileServer`.

   ```
   fileServer# mkdir -m 0755 /var/spool/torque
   fileServer# mkdir -m 0750 /var/spool/torque/server_priv
   ```

   b. Update `/etc/exports` on `fileServer`. The IP addresses should be that of `server2`.

   ```
   /var/spool/torque/server_priv 192.168.0.0/255.255.255.0(rw,sync,no_root_squash)
   ```

   c. Update the list of NFS exported file systems.

   ```
   fileServer# exportfs -r
   ```

6. If the NFS daemons are not already running on `fileServer`, start them.

   ```
   > systemctl restart rpcbind.service
   > systemctl start nfs-server.service
   > systemctl start nfs-lock.service
   > systemctl start nfs-idmap.service
   ```

7. Mount the exported file systems on `server1` by following these steps:

   a. Create the directory reference and mount them.

   ```
   server1# mkdir /var/spool/torque/server_priv
   ```

   Repeat this process for `server2`.

   b. Update `/etc/fstab` on `server1` to ensure that NFS mount is performed on startup.

   ```
   fileServer:/var/spool/torque/server_priv /var/spool/torque/server_priv nfs
   rsize= 8192,wsize=8192,timeo=14,intr
   ```

   Repeat this step for `server2`.

8. Install TORQUE by following these steps:

   a. Download and extract TORQUE 5.0.1 on `server1`.

   ```
   server1# wget http://github.com/adaptivecomputing/torque/ branches/5.0.1/torque-
   5.0.1.tar.gz
   server1# tar -xvzf torque-5.0.1.tar.gz
   ```

   b. Navigate to the TORQUE directory and compile TORQUE on server1.

   ```
   server1# configure
   server1# make
   server1# make install
   server1# make packages
   ```

   c. If the installation directory is shared on both head nodes, then run `make install` on `server1`.

   ```
   server1# make install
   ```

If the installation directory is not shared, repeat step 8a-b (downloading and installing TORQUE) on `server2`.

9. Start trqauthd.

```
server1# /etc/init.d/trqauthd start
```

10. Configure TORQUE for HA.

    a. List the host names of all nodes that run pbs_server in the `torque/server_name` file. You must also include the host names of all nodes running pbs_server in the `torque/server_name` file of each MOM node. The syntax of `torque/server_name` is a comma-delimited list of host names.

    ```
    server1
    server2
    ```

    b. Create a simple queue configuration for TORQUE job queues on server1.

    ```
    server1# pbs_server -t create
    server1# qmgr -c “set server scheduling=true”
    server1# qmgr -c “create queue batch queue_type=execution”
    server1# qmgr -c “set queue batch started=true”
    server1# qmgr -c “set queue batch enabled=true”
    server1# qmgr -c “set queue batch resources_default.nodes=1”
    server1# qmgr -c “set queue batch resources_default.walltime=3600”
    server1# qmgr -c “set server default_queue=batch”
    ```

    Because `server_priv/*` is a shared drive, you do not need to repeat this step on `server2`.

    c. Add the root users of TORQUE to the TORQUE configuration as an operator and manager.

    ```
    server1# qmgr -c “set server managers += root@server1”
    server1# qmgr -c “set server managers += root@server2”
    server1# qmgr -c “set server operators += root@server1”
    server1# qmgr -c “set server operators += root@server2”
    ```

    Because `server_priv/*` is a shared drive, you do not need to repeat this step on Server 2.

    d. You must update the lock file mechanism for TORQUE in order to determine which server is the primary. To do so, use the `lock_file_update_time` and `lock_file_check_time` parameters. The primary pbs_server will update the lock file based on the specified `lock_file_update_time` (default value of 3 seconds). All backup pbs_servers will check the lock file as indicated by the `lock_file_check_time` parameter (default value of 9 seconds). The `lock_file_update_time` must be less than the `lock_file_check_time`. When a failure occurs, the backup pbs_server takes up to the `lock_file_check_time` value to take over.

    ```
    server1# qmgr -c “set server lock_file_check_time=5”
    server1# qmgr -c “set server lock_file_update_time=3”
    ```

> Because `server_priv/*` is a shared drive, you do not need to repeat this step on `server2`.

e. List the servers running pbs_server in the TORQUE `acl_hosts` file.

```
server1# qmgr -c "set server acl_hosts += server1"
server1# qmgr -c "set server acl_hosts += server2"
```

> Because `server_priv/*` is a shared drive, you do not need to repeat this step on `server2`.

f. Restart the running pbs_server in HA mode.

```
server1# qterm
```

g. Start the pbs_server on the secondary server.

```
server1# pbs_server --ha -l server2:port
server2# pbs_server --ha -l server1:port
```

11. Check the status of TORQUE in HA mode.

```
server1# qmgr -c "p s"
server2# qmgr -c "p s"
```

The commands above returns all settings from the active TORQUE server from either node.

Drop one of the pbs_servers to verify that the secondary server picks up the request.

```
server1# qterm
server2# qmgr -c "p s"
```

Stop the pbs_server on `server2` and restart pbs_server on `server1` to verify that both nodes can handle a request from the other.

12. Install a pbs_mom on the compute nodes.

a. Copy the install scripts to the compute nodes and install.

b. Navigate to the shared source directory of TORQUE and run the following:

```
node1# torque-package-mom-linux-x86_64.sh --install
node2# torque-package-clients-linux-x86_64.sh --install
```

Repeat this for each compute node. Verify that the `/var/pool/ torque/server-name` file shows all your compute nodes.

c. On `server1` or `server2`, configure the nodes file to identify all available MOMs. To do so, edit the `/var/spool/torque/server_priv/nodes` file.

```
node1 np=2
node2 np=2
```

> Change the `np` flag to reflect number of available processors on that node.

d. Recycle the pbs_servers to verify that they pick up the MOM configuration.

```
server1# qterm; pbs_server --ha -l server2:port
server2# qterm; pbs_server --ha -l server1:port
```

e. Start the pbs_mom on each execution node.

```
node5# pbs_mom
node6# pbs_mom
```

Installing TORQUE in high availability mode on headless nodes

The following procedure demonstrates a TORQUE installation in high availability (HA) mode on nodes with no local hard drive.

Requirements

- gcc (GCC) 4.1.2
- BASH shell
- Servers (these cannot be two VMs on the same hypervisor) configured the following way:
  - 2 main servers with identical architecture
    - `server1` — Primary server running TORQUE with a file system share (this example uses NFS)
    - `server2` — Secondary server running with TORQUE with a file system share (this example uses NFS)
  - Compute nodes
  - `fileServer` — A shared file system server (this example uses NFS)

To install TORQUE in HA mode on a node with no local hard drive

1. Stop all firewalls or update your firewall to allow traffic from TORQUE services.

```
> service iptables stop
> chkconfig iptables off
```

If you are unable to stop the firewall due to infrastructure restriction, open the following ports:

- `15001[tcp,udp]`
- `15002[tcp,udp]`
- `15003[tcp,udp]`

2. Disable SELinux

```
> vi /etc/sysconfig/selinux

SELINUX=disabled
```

3. Update your main `~/.bashrc` profile to ensure you are always referencing the applications to be installed on all servers.

```
# TORQUE
export TORQUEHOME=/var/spool/torque

# Library Path

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${TORQUEHOME}/lib

# Update system paths
export PATH=${TORQUEHOME}/bin:${TORQUEHOME}/sbin:$ {PATH}
```

4. Verify `server1` and `server2` are resolvable via either DNS or looking for an entry in the `/etc/hosts` file.

5. Configure the NFS Mounts by following these steps:

   a. Create mount point folders on `fileServer`.

   ```
   fileServer# mkdir -m 0755 /var/spool/torque
   ```

   b. Update `/etc/exports` on `fileServer`. The IP addresses should be that of `server2`.

   ```
   /var/spool/torque/ 192.168.0.0/255.255.255.0(rw,sync,no_root_squash)
   ```

   c. Update the list of NFS exported file systems.

   ```
   fileServer# exportfs -r
   ```

6. If the NFS daemons are not already running on `fileServer`, start them.

```
> systemctl restart rpcbind.service
> systemctl start nfs-server.service
> systemctl start nfs-lock.service
> systemctl start nfs-idmap.service
```

7. Mount the exported file systems on `server1` by following these steps:

   a. Create the directory reference and mount them.

   ```
   server1# mkdir /var/spool/torque
   ```

   Repeat this process for `server2`.

   b. Update `/etc/fstab` on `server1` to ensure that NFS mount is performed on startup.

   ```
   fileServer:/var/spool/torque/server_priv /var/spool/torque/server_priv nfs
   rsize= 8192,wsize=8192,timeo=14,intr
   ```

   Repeat this step for `server2`.

8. Install TORQUE by following these steps:

a. Download and extract TORQUE 5.0.1 on `server1`.

```
server1# wget http://github.com/adaptivecomputing/torque/ branches/5.0.1/torque-5.0.1.tar.gz
server1# tar -xvzf torque-5.0.1.tar.gz
```

b. Navigate to the TORQUE directory and compile TORQUE with the HA flag on server1.

```
server1# configure --prefix=/var/spool/torque
server1# make
server1# make install
server1# make packages
```

c. If the installation directory is shared on both head nodes, then run `make install` on `server1`.

```
server1# make install
```

If the installation directory is not shared, repeat step 8a-b (downloading and installing TORQUE) on `server2`.

9. Start trqauthd.

```
server1# /etc/init.d/trqauthd start
```

10. Configure TORQUE for HA.

a. List the host names of all nodes that run pbs_server in the `torque/server_name` file. You must also include the host names of all nodes running pbs_server in the `torque/server_name` file of each MOM node. The syntax of `torque/server_name` is a comma-delimited list of host names.

```
server1,server2
```

b. Create a simple queue configuration for TORQUE job queues on server1.

```
server1# pbs_server -t create
server1# qmgr -c "set server scheduling=true"
server1# qmgr -c "create queue batch queue_type=execution"
server1# qmgr -c "set queue batch started=true"
server1# qmgr -c "set queue batch enabled=true"
server1# qmgr -c "set queue batch resources_default.nodes=1"
server1# qmgr -c "set queue batch resources_default.walltime=3600"
server1# qmgr -c "set server default_queue=batch"
```

Because `TORQUEHOME` is a shared drive, you do not need to repeat this step on `server2`.

c. Add the root users of TORQUE to the TORQUE configuration as an operator and manager.

```
server1# qmgr -c "set server managers += root@server1"
server1# qmgr -c "set server managers += root@server2"
server1# qmgr -c "set server operators += root@server1"
server1# qmgr -c "set server operators += root@server2"
```

Because `TORQUEHOME` is a shared drive, you do not need to repeat this step on `server2`.

d. You must update the lock file mechanism for TORQUE in order to determine which server is the primary. To do so, use the `lock_file_update_time` and `lock_file_check_time` parameters. The primary pbs_server will update the lock file based on the specified `lock_file_update_time` (default value of 3 seconds). All backup pbs_servers will check the lock file as indicated by the `lock_file_check_time` parameter (default value of 9 seconds). The `lock_file_update_time` must be less than the `lock_file_check_time`. When a failure occurs, the backup pbs_server takes up to the `lock_file_check_time` value to take over.

```
server1# qmgr -c "set server lock_file_check_time=5"
server1# qmgr -c "set server lock_file_update_time=3"
```

> Because `TORQUEHOME` is a shared drive, you do not need to repeat this step on `server2`.

e. List the servers running pbs_server in the TORQUE `acl_hosts` file.

```
server1# qmgr -c "set server acl_hosts += server1"
server1# qmgr -c "set server acl_hosts += server2"
```

> Because `TORQUEHOME` is a shared drive, you do not need to repeat this step on `server2`.

f. Restart the running pbs_server in HA mode.

```
server1# qterm
```

g. Start the pbs_server on the secondary server.

```
server1# pbs_server --ha -l server2:port
server2# pbs_server --ha -l server1:port
```

11. Check the status of TORQUE in HA mode.

```
server1# qmgr -c "p s"
server2# qmgr -c "p s"
```

The commands above returns all settings from the active TORQUE server from either node.

Drop one of the pbs_servers to verify that the secondary server picks up the request.

```
server1# qterm
server2# qmgr -c "p s"
```

Stop the pbs_server on `server2` and restart pbs_server on `server1` to verify that both nodes can handle a request from the other.

12. Install a pbs_mom on the compute nodes.

a. On `server1` or `server2`, configure the nodes file to identify all available MOMs. To do so, edit the `/ var/spool/torque/server_priv/nodes` file.

```
node1 np=2
node2 np=2
```

> Change the `np` flag to reflect number of available processors on that node.

b. Recycle the pbs_servers to verify that they pick up the MOM configuration.

```
server1# qterm; pbs_server --ha -l server2:port
server2# qterm; pbs_server --ha -l server1:port
```

c. Start the pbs_mom on each execution node.

```
server1# pbs_mom -d <mom-server1>
server2# pbs_mom -d <mom-server2>
```

Example setup of high availability

1. The machines running pbs_server must have access to a shared `server_priv/` directory (usually an NFS share on a MoM).
2. All MoMs must have the same content in their `server_name` file. This can be done manually or via an NFS share. The `server_name` file contains a comma-delimited list of the hosts that run pbs_server.

```
# List of all servers running pbs_server
server1,server2
```

3. The machines running pbs_server must be listed in **acl_hosts**.

```
> qmgr -c "set server acl_hosts += server1"
> qmgr -c "set server acl_hosts += server2"
```

4. Start pbs_server with the --ha option.

```
[root@server1]$ pbs_server --ha

[root@server2]$ pbs_server --ha
```

Related topics

- **Setting Server Policies** on page 2277
- **Queue Configuration** on page 2277

Setting min_threads and max_threads

There are two threadpools in TORQUE, one for background tasks and one for incoming requests from the MOMs and through the API (client commands, Moab, and so forth). The **min_threads** on page 2429 and **max_threads** on page 2429 parameters control the number of total threads used for both, not for each individually. The incoming requests' threadpool has three-quarters of **min_threads** for its minimum, and three-quarters of **max_threads** for its maximum, with the background pool receiving the other one-quarter.

Additionally, pbs_server no longer allows incoming requests to pile up indefinitely. When the threadpool is too busy for incoming requests, it indicates such, returning PBSE_SERVER_BUSY with the accompanying message that "Pbs Server is currently too busy to service this request. Please retry this request." The threshold for this message, if the request is from a manager, is that at least two threads be available in the threadpool. If the request comes from a non-manager, 5% of the threadpool must be available for the request to be serviced. Note that availability is calculated based on the maximum threads and not based on the current number of threads allocated.

If an undesirably large number of requests are given a busy response, one option is to increase the number of maximum threads for the threadpool. If the load on the server is already very high, then this is probably not going to help, but if the CPU load is lower, then it may help. Remember that by default the threadpool shrinks down once the extra threads are no longer needed. This is controlled via the **thread_idle_seconds** on page 2435 server parameter.

> Any change in the **min_threads**, **max_threads**, or **thread_idle_seconds** parameters requires a restart of pbs_server to take effect.

Integrating Schedulers for TORQUE

Selecting the cluster scheduler is an important decision and significantly affects cluster utilization, responsiveness, availability, and intelligence. The default TORQUE scheduler, pbs_sched, is very basic and will provide poor utilization of your cluster's resources. Other options, such as Maui Scheduler or Moab Workload Manager, are highly recommended. If you are using Maui or Moab, see **Moab-TORQUE Integration Guide** on page 1206. If using pbs_sched, simply start the pbs_sched daemon.

> If you are installing Moab Cluster Manager, TORQUE and Moab were configured at installation for interoperability and no further action is required.

Configuring Data Management

This section contains information about SCP-based data management with TORQUE. It describes how to use TORQUE with NFS and other networked filesystems. It also outlines file staging requirements. For details, see these topics:

- **SCP Setup** on page 2304
- **NFS and Other Networked Filesystems** on page 2307
- **File stage-in/stage-out** on page 2308

SCP Setup

To use SCP-based data management, TORQUE must be authorized to migrate data to any of the compute nodes. If this is not already enabled within the cluster, this can be achieved with the process described below. This process enables uni-directional access for a particular user from a *source* host to a *destination* host.

> These directions were written using OpenSSH version 3.6 and may not transfer correctly to older versions.

To set up TORQUE for SCP, follow the directions in each of these topics:

- **Generating SSH Key on Source Host** on page 2305
- **Copying Public SSH Key to each Destination Host** on page 2305
- **Configuring the SSH Daemon on Each Destination Host** on page 2306
- **Validating Correct SSH Configuration** on page 2306
- **Enabling Bi-directional SCP Access** on page 2306
- **Compiling TORQUE to Support SCP** on page 2307
- **Troubleshooting** on page 2307

Related topics

- **Configuring Data Management** on page 2304

Generating SSH Key on Source Host

On the source host as the transfer user, execute the following:

```
> ssh-keygen -t rsa
```

This will prompt for a passphrase (optional) and create two files (`id_rsa` and `id_rsa.pub`) inside `~/.ssh/`.

Related topics

- **SCP Setup** on page 2304
- **Copying Public SSH Key to each Destination Host** on page 2305

Copying Public SSH Key to each Destination Host

Transfer public key to each destination host as the transfer user:

Easy key copy:

```
ssh-copy-id [-i [identity_file]] [user@]machine
```

Manual steps to copy keys:

```
> scp ~/.ssh/id_rsa.pub destHost:~ (enter password)
```

Create an authorized_keys file on each destination host:

```
> ssh destHost (enter password)
> cat id_rsa.pub >> .ssh/authorized_keys
```

If the .ssh directory does not exist, create it with 700 privileges (`mkdir .ssh`; `chmod 700 .ssh`):

```
> chmod 700 .ssh/authorized_keys
```

Related topics

- **Generating SSH Key on Source Host** on page 2305
- **SCP Setup** on page 2304

Configuring the SSH Daemon on Each Destination Host

Some configuration of the SSH daemon may be required on the destination host. (Because this is not always the case, see **Validating Correct SSH Configuration** on page 2306 and test the changes made to this point. If the tests fail, proceed with this step and then try testing again.) Typically, this is done by editing the `/etc/ssh/sshd_config` file (root access needed). To verify correct configuration, see that the following attributes are set (not commented):

```
RSAAuthentication yes
PubkeyAuthentication yes
```

If configuration changes were required, the SSH daemon will need to be restarted (root access needed):

```
> /etc/init.d/sshd restart
```

Related topics

- **SCP Setup** on page 2304

Validating Correct SSH Configuration

If all is properly configured, the following command issued on the *source* host should succeed and not prompt for a password:

```
> scp destHost:/etc/motd /tmp
```

If this is your first time accessing *destination* from *source*, it may ask you if you want to add the fingerprint to a file of known hosts. If you specify yes, this message should no longer appear and should not interfere with scp copying via TORQUE. Also, it is important that the full hostname appear in the `known_hosts` file. To do this, use the full hostname for *destHost*, as in `machine.domain.org` instead of just `machine`.

Related topics

- **SCP Setup** on page 2304

Enabling Bi-directional SCP Access

The preceding steps allow *source* access to destination without prompting for a password. The reverse, however, is not true. Repeat the steps, but this time using the *destination* as the *source*, etc. to enable bi-directional SCP access (i.e. *source* can send to *destination* and *destination* can send to *source* without password prompts.)

Related topics

- SCP Setup on page 2304

Compiling TORQUE to Support SCP

> In TORQUE 2.1 and later, SCP is the default remote copy protocol. These instructions are only necessary for earlier versions.

TORQUE must be re-configured (and then rebuilt) to use SCP by passing in the `--with-scp` flag to the configure script:

```
> ./configure --prefix=xxx --with-scp
> make
```

> If special SCP flags are required in your local setup, these can be specified using the $rcpcmd parameter.

Related topics

- SCP Setup on page 2304

Troubleshooting

If, after following all of the instructions in this section (see SCP Setup on page 2304), TORQUE is still having problems transferring data with SCP, set the `PBSDEBUG` environment variable and restart the pbs_mom for details about copying. Also check the MOM log files for more details.

Related topics

- SCP Setup on page 2304

NFS and Other Networked Filesystems

When a batch job starts, its `stdin` file (if specified) is copied from the submission directory on the remote submission host. This file is placed in the `$PBSMOMHOME` directory on the mother superior node (i.e., `/usr/spool/PBS/spool`). As the job runs, stdout and stderr files are generated and placed in this directory using the naming convention *$JOBID.OU* and *$JOBID.ER*.

When the job completes, the MOM copies the files into the directory from which the job was submitted. By default, this file copying will be accomplished using a remote copy facility such as *rcp* or *scp*.

If a shared file system such as NFS, DFS, or AFS is available, a site can specify that the MOM should take advantage of this by specifying the `$usecp` directive inside the MOM configuration file (located in the `$PBSMOMHOME/mom_priv` directory) using the following format:

```
$usecp <HOST>:<SRCDIR> <DSTDIR>
```

<HOST> can be specified with a leading wildcard ('*') character. The following example demonstrates this directive:

```
mom_priv/config

# /home is NFS mounted on all hosts
$usecp *:/home /home
# submission hosts in domain fte.com should map '/data' directory on submit host to
# '/usr/local/data' on compute host
$usecp *.fte.com:/data /usr/local/data
```

If for any reason the MOM daemon is unable to copy the output or error files to the submission directory, these files are instead copied to the `undelivered` directory also located in `$PBSMOMHOME`.

Related topics

- **Configuring Data Management** on page 2304

File stage-in/stage-out

File staging requirements are specified using the `stagein` and `stageout` directives of the **qsub** command. Stagein requests occur before the job starts execution, while stageout requests happen after a job completes.

On completion of the job, all staged-in and staged-out files are removed from the execution system. The `file_list` is in the form `local_file@hostname:remote_file[,...]` regardless of the direction of the copy. The name `local_file` is the name of the file on the system where the job executed. It may be an absolute path or relative to the home directory of the user. The name `remote_file` is the destination name on the host specified by hostname. The name may be absolute or relative to the user's home directory on the destination host. The use of wildcards in the file name is not recommended.

The file names map to a remote copy program (rcp/scp/cp, depending on configuration) called on the execution system in the following manner:

For stagein: `rcp/scp hostname:remote_file local_file`

For stageout: `rcp/scp local_file hostname:remote_file`

Examples

```
# stage /home/john/input_source.txt from node13.fsc to /home/john/input_
destination.txt on master compute node
> qsub -l nodes=1,walltime=100 -W stagein=input_
source.txt@node13.fsc:/home/john/input_destination.txt
```

```
# stage /home/bill/output_source.txt on master compute node to /tmp/output_
destination.txt on node15.fsc
> qsub -l nodes=1,walltime=100 -W stageout=/tmp/output_
source.txt@node15.fsc:/home/bill/output_destination.txt
```

```
$ fortune >xxx;echo cat xxx|qsub -W stagein=xxx@`hostname`:xxx
199.myhost.mydomain
$ cat STDIN*199
Anyone who has had a bull by the tail knows five or six more things
than someone who hasn't.
-- Mark Twain
```

Related topics

- **Configuring Data Management** on page 2304

MPI (Message Passing Interface) Support

A message passing library is used by parallel jobs to augment communication between the tasks distributed across the cluster. TORQUE can run with any message passing library and provides limited integration with some MPI libraries.

For more information, see these topics:

- **MPICH** on page 2309
- **Open MPI** on page 2310

MPICH

One of the most popular MPI libraries is MPICH available from Argonne National Lab. If using this release, you may want to consider also using the mpiexec tool for launching MPI applications. Support for mpiexec has been integrated into TORQUE.

MPIExec Overview

mpiexec is a replacement program for the script *mpirun*, which is part of the *mpich* package. It is used to initialize a parallel job from within a PBS batch or interactive environment. mpiexec uses the task manager library of PBS to spawn copies of the executable on the nodes in a PBS allocation.

Reasons to use mpiexec rather than a script (mpirun) or an external daemon (mpd):

- Starting tasks with the task manager (TM) interface is much faster than invoking a separate rsh * once for each process.
- Resources used by the spawned processes are accounted correctly with mpiexec, and reported in the PBS logs, because all the processes of a parallel job remain under the control of PBS, unlike when using mpirun-like scripts.
- Tasks that exceed their assigned limits of CPU time, wallclock time, memory usage, or disk space are killed cleanly by PBS. It is quite hard for processes to escape control of the resource manager when using mpiexec.
- You can use mpiexec to enforce a security policy. If all jobs are forced to spawn using mpiexec and the PBS execution environment, it is not necessary to enable rsh or ssh access to the compute nodes in the cluster.

For more information, see the mpiexec homepage.

MPIExec Troubleshooting

Although problems with mpiexec are rare, if issues do occur, the following steps may be useful:

- Determine current version using `mpiexec --version` and review the change log available on the MPI homepage to determine if the reported issue has already been corrected.
- Send email to the mpiexec mailing list at mpiexec@osc.edu.
- Browse the mpiexec user list archives for similar problems and resolutions.
- Read the FAQ contained in the `README` file and the mpiexec man pages contained within the mpiexec distribution.
- Increase the logging of mpiexec operation with `mpiexec --verbose` (reports messages to `stderr`).
- Increase logging of the master and slave resource manager execution daemons associated with the job (with TORQUE, use `$loglevel` to 5 or higher `in $TORQUEROOT/mom_priv/config` and look for '**tm**' messages after associated `join job` messages).
- Use `tracejob` (included with TORQUE) or `qtracejob` (included with OSC's pbstools package) to isolate failures within the cluster.
- If the message '`exec: Error: get_hosts: pbs_connect: Access from host not allowed, or unknown host`' appears, this indicates that mpiexec cannot communicate with the pbs_server daemon. In most cases, this indicates that the `$TORQUEROOT/server_name` file points to the wrong server or the node cannot resolve the server's name. The **qstat** command can be run on the node to test this.

General MPI Troubleshooting

When using MPICH, some sites have issues with orphaned MPI child processes remaining on the system after the master MPI process has been terminated. To address this, TORQUE epilogue scripts can be created that properly clean up the orphaned processes (see **Appendix G: Prologue and Epilogue Scripts** on page 2469).

Related topics

- **MPI (Message Passing Interface) Support** on page 2309

Open MPI

Open MPI is a new MPI implementation that combines technologies from multiple projects to create the best possible library. It supports the TM interface for integration with TORQUE. More information is available in the FAQ.

Related topics

- **MPI (Message Passing Interface) Support** on page 2309

Resources

A primary task of any resource manager is to monitor the state, health, configuration, and utilization of managed resources. TORQUE is specifically designed to monitor compute hosts for use in a batch environment. TORQUE is not designed to monitor non-compute host resources such as software licenses, networks, file systems, and so forth, although these resources can be integrated into the cluster using some scheduling systems.

With regard to monitoring compute nodes, TORQUE reports about a number of attributes broken into three major categories:

- **Configuration** on page 2311
- **Utilization** on page 2312
- **Node states** on page 2312

Configuration

Configuration includes both detected hardware configuration and specified batch attributes.

Attribute	Description	Details
Architecture (arch)	operating system of the node	The value reported is a derivative of the operating system installed.
Node Features (properties)	arbitrary string attributes associated with the node	No node features are specified by default. If required, they are set using the `nodes` file located in the `TORQUE_HOME/server_priv` directory. They may specify any string and are most commonly used to allow users to request certain subsets of nodes when submitting jobs.
Local Disk (size)	configured local disk	By default, local disk space is not monitored. If the MOM configuration size [fs=<FS>] parameter is set, TORQUE will report, in kilobytes, configured disk space within the specified directory.
Memory (physmem)	local memory/RAM	Local memory/RAM is monitored and reported in kilobytes.

Attribute	Description	Details
Processors (ncpus/np)	real/virtual processors	The number of processors detected by TORQUE is reported via the *ncpus* attribute. However, for scheduling purposes, other factors are taken into account. In its default configuration, TORQUE operates in "dedicated" mode with each node possessing a single virtual processor. In dedicated mode, each job task will consume one virtual processor and TORQUE will accept workload on each node until all virtual processors on that node are in use. While the number of virtual processors per node defaults to 1, this may be configured using the nodes file located in the `TORQUE_HOME/server_priv` directory. An alternative to dedicated mode is "timeshared" mode. If TORQUE's time-shared mode is enabled, TORQUE will accept additional workload on each node until the node's *maxload* limit is reached.
Swap (tot-mem)	virtual memory/Swap	Virtual memory/Swap is monitored and reported in kilobytes.

Utilization

Utilization includes information regarding the amount of node resources currently in use as well as information about who or what is consuming it.

Attribute	Description	Details
Disk (size)	local disk availability	By default, local disk space is not monitored. If the MOM configuration size [fs=<FS>] parameter is set, TORQUE will report configured and currently available disk space within the specified directory in kilobytes.
Memory (availmem)	real memory/RAM	Available real memory/RAM is monitored and reported in kilobytes.
Network (netload)	local network adapter usage	Reports total number of bytes transferred in or out by the network adapter.
Processor Utilization (loadave)	node's cpu load average	Reports the node's 1 minute bsd load average.

Node states

State information includes administrative status, general node health information, and general usage status.

Attribute	Description	Details
Idle Time (idletime)	time since local keyboard/mouse activity has been detected	Time in seconds since local keyboard/mouse activity has been detected.
State (state)	monitored/admin node state	A node can be in one or more of the following states: • *busy* - node is full and will not accept additional work • *down* - node is failing to report, is detecting local failures with node • *free* - node is ready to accept additional work • *job-exclusive* - all available virtual processors are assigned to jobs • *job-sharing* - node has been allocated to run multiple shared jobs and will remain in this state until jobs are complete • *offline* - node has been instructed by an admin to no longer accept work • *reserve* - node has been reserved by the server • *time-shared* - node always allows multiple jobs to run concurrently • *unknown* - node has not been detected

Accounting Records

TORQUE maintains accounting records for batch jobs in the following directory:

`$TORQUEROOT/server_priv/accounting/<TIMESTAMP>`

`$TORQUEROOT` defaults to `/usr/spool/PBS` and *<TIMESTAMP>* is in the format: *YYYYMMDD*.

These records include events, time stamps, and information on resources requested and used.

Records for four different event types are produced and are described in the following table:

Record marker	Record type	Description
A	abort	Job has been aborted by the server
C	checkpoint	Job has been checkpointed and held
D	delete	Job has been deleted

Record marker	Record type	Description
E	exit	Job has exited (either successfully or unsuccessfully)
Q	queue	Job has been submitted/queued
R	rerun	Attempt to rerun the job has been made
S	start	Attempt to start the job has been made (if the job fails to properly start, it may have multiple job start records)
T	restart	Attempt to restart the job (from checkpoint) has been made (if the job fails to properly start, it may have multiple job start records)

Accounting Variables

The following table offers accounting variable descriptions. Descriptions for accounting variables not indicated in the table, particularly those prefixed with **Resources_List**, are available at **Job Submission** on page 2234.

Variable	Description
ctime	Time job was created
etime	Time job became eligible to run
qtime	Time job was queued
start	Time job started to run

A sample record in this file can look like the following:

```
08/26/2014 17:07:44;Q;11923.napali;queue=batch
08/26/2014 17:07:50;S;11923.napali;user=dbeer group=company jobname=STDIN queue=batch
ctime=1409094464 qtime=1409094464 etime=1409094464 start=1409094470 owner=dbeer@napali
exec_host=napali/0+napali/1+napali/2+napali/3+napali/4+napali/5+torque-devtest-
03/0+torque-devtest-03/1+torque-devtest-03/2+torque-devtest-03/3+torque-devtest-
03/4+torque-devtest-03/5 Resource_List.neednodes=2:ppn=6 Resource_List.nodect=2
Resource_List.nodes=2:ppn=6
08/26/2014 17:08:04;E;11923.napali;user=dbeer group=company jobname=STDIN queue=batch
ctime=1409094464 qtime=1409094464 etime=1409094464 start=1409094470 owner=dbeer@napali
exec_host=napali/0+napali/1+napali/2+napali/3+napali/4+napali/5+torque-devtest-
03/0+torque-devtest-03/1+torque-devtest-03/2+torque-devtest-03/3+torque-devtest-
03/4+torque-devtest-03/5 Resource_List.neednodes=2:ppn=6 Resource_List.nodect=2
Resource_List.nodes=2:ppn=6 session=11352 total_execution_slots=12 unique_node_count=2
end=1409094484 Exit_status=265 resources_used.cput=00:00:00 resources_used.mem=82700kb
resources_used.vmem=208960kb resources_used.walltime=00:00:14 Error_Path=/dev/pts/11
Output_Path=/dev/pts/11
```

The value of `Resource_List.*` is the amount of resources requested, and the value of `resources_used.*` is the amount of resources actually used.

total_execution_slots and *unique_node_count* display additional information regarding the job resource usage.

Job Logging

New in TORQUE 2.5.3 is the ability to log job information for completed jobs. The information stored in the log file is the same information produced with the command **qstat** `-f`. The log file data is stored using an XML format. Data can be extracted from the log using the utility `showjobs` found in the `contrib/` directory of the TORQUE source tree. Custom scripts that can parse the XML data can also be used.

For details about job logging, see these topics:

- **Job Log Location and Name** on page 2315
- **Enabling Job Logs** on page 2316

Job Log Location and Name

When job logging is enabled (see **Enabling Job Logs** on page 2316), the job log is kept at `$TORQUE_HOME/job_logs`. The naming convention for the job log is the same as for the server log or MOM log. The log name is created from the current year/month/day.

For example, if today's date is 26 October, 2010 the log file is named 20101026.

A new log file is created each new day that data is written to the log.

Related topics

- **Enabling Job Logs** on page 2316
- **Job Logging** on page 2315

Enabling Job Logs

There are five new server parameters used to enable job logging. These parameters control what information is stored in the log and manage the log files.

Parameter	Description
record_job_info	This must be set to true in order for job logging to be enabled. If not set to true, the remaining server parameters are ignored.
record_job_script	If set to true, this adds the contents of the script executed by a job to the log.
job_log_file_max_size	This specifies a soft limit (in kilobytes) for the job log's maximum size. The file size is checked every five minutes and if the *current day* file size is greater than or equal to this value, it is rolled from *<filename>* to *<filename.1>* and a new empty log is opened. If the current day file size exceeds the maximum size a second time, the *<filename.1>* log file is rolled to *<filename.2>*, the current log is rolled to *<filename.1>*, and a new empty log is opened. Each new log causes all other logs to roll to an extension that is one greater than its current number. Any value less than 0 is ignored by pbs_server (meaning the log will not be rolled).
job_log_file_roll_depth	This sets the maximum number of new log files that are kept in a day if the **job_log_file_max_size** parameter is set. For example, if the roll depth is set to 3, no file can roll higher than *<filename.3>*. If a file is already at the specified depth, such as *<filename.3>*, the file is deleted so it can be replaced by the incoming file roll, *<filename.2>*.
job_log_keep_days	This maintains logs for the number of days designated. If set to 4, any log file older than 4 days old is deleted.

Related topics

- **Job Log Location and Name** on page 2315
- **Job Logging** on page 2315

Troubleshooting

There are a few general strategies that can be followed to determine the cause of unexpected behavior. These are a few of the tools available to help determine where problems occur. See these topics for details:

Host Resolution

The TORQUE server host must be able to perform both forward and reverse name lookup on itself and on all compute nodes. Likewise, each compute node must be able to perform forward and reverse name lookup on itself, the TORQUE server host, and all other compute nodes. In many cases, name resolution is handled by configuring the node's `/etc/hosts` file although *DNS* and *NIS* services may also be used. Commands such as `nslookup` or `dig` can be used to verify proper host resolution.

> Invalid host resolution may exhibit itself with compute nodes reporting as down within the output of **pbsnodes**`-a` and with failure of the **momctl** `-d3` command.

Related topics

- **Troubleshooting** on page 2316

Firewall Configuration

Be sure that, if you have firewalls running on the server or node machines, you allow connections on the appropriate ports for each machine. TORQUE pbs_mom daemons use UDP ports 1023 and below if privileged ports are configured (privileged ports is the default). The pbs_server and pbs_mom daemons use TCP and UDP ports 15001-15004 by default.

Firewall based issues are often associated with server to MOM communication failures and messages such as 'premature end of message' in the log files.

Also, the `tcpdump` program can be used to verify the correct network packets are being sent.

Related topics

- **Troubleshooting** on page 2316

TORQUE Log Files

pbs_server and pbs_mom log files

The pbs_server keeps a daily log of all activity in the `TORQUE_HOME/server_logs` directory. The pbs_mom also keeps a daily log of all activity in the `TORQUE_HOME/mom_logs/` directory. These logs contain information on communication between server and MOM as well as information on jobs as they enter the queue and as they are dispatched, run, and terminated. These logs can be very helpful in determining general job failures. For MOM logs, the verbosity of the logging can be adjusted by setting the $loglevel parameter in the `mom_priv/config` file. For server logs, the verbosity of the logging can be adjusted by setting the server log_level attribute in **qmgr**.

For both pbs_mom and pbs_server daemons, the log verbosity level can also be adjusted by setting the environment variable **PBSLOGLEVEL** to a value between 0 and 7. Further, to dynamically change the log level of a running daemon, use the `SIGUSR1` and `SIGUSR2` signals to increase and decrease the active loglevel by one. Signals are sent to a process using the `kill` command.

For example, ``kill -USR1 `pgrep pbs_mom` `` would raise the log level up by one.

The current loglevel for pbs_mom can be displayed with the command **momctl** `-d3`.

trqauthd log files

As of TORQUE 4.1.3, trqauthd logs its events in the `$TORQUE_HOME/client_logs` directory. It names the log files in the format <*YYYYMMDD*>, creating a new log daily as events occur.

> You might see some peculiar behavior if you mount the `client_logs` directory for shared access via network-attached storage.
>
> When trqauthd first gets access on a particular day, it writes an "open" message to the day's log file. It also writes a "close" message to the last log file it accessed prior to that, which is usually the previous day's log file, but not always. For example, if it is Monday and no client commands were executed over the weekend, trqauthd writes the "close" message to Friday's file.
>
> Since the various trqauthd binaries on the submit hosts (and potentially, the compute nodes) each write an "open" and "close" message on the first access of a new day, you'll see multiple (seemingly random) accesses when you have a shared log.

The trqauthd records the following events along with the date and time of the occurrence:

- When trqauthd successfully starts. It logs the event with the IP address and port.
- When a user successfully authenticates with trqauthd.
- When a user fails to authenticate with trqauthd.
- When trqauthd encounters any unexpected errors.

Example 4-26: trqauthd logging sample

```
2012-10-05 15:05:51.8404 Log opened
2012-10-05 15:05:51.8405 TORQUE authd daemon started and listening on IP:port
101.0.1.0:12345
2012-10-10 14:48:05.5688 User hfrye at IP:port abc:12345 logged in
```

Related topics

- **Troubleshooting** on page 2316

Using "tracejob" to Locate Job Failures

Overview

The *tracejob* utility extracts job status and job events from accounting records, MOM log files, server log files, and scheduler log files. Using it can help identify where, how, a why a job failed. This tool takes a job id as a parameter as well as arguments to specify which logs to search, how far into the past to search, and other conditions.

Syntax

```
tracejob [-a|s|l|m|q|v|z] [-c count] [-w size] [-p path] [ -n <DAYS>] [-f
filter_type] <JOBID>
```

```
-p  :  path to PBS_SERVER_HOME
-w  :  number of columns of your terminal
-n  :  number of days in the past to look for job(s) [default 1]
-f  :  filter out types of log entries, multiple -f's can be specified
       error, system, admin, job, job_usage, security, sched, debug,
       debug2, or absolute numeric hex equivalent
-z  :  toggle filtering excessive messages
-c  :  what message count is considered excessive
-a  :  don't use accounting log files
-s  :  don't use server log files
-l  :  don't use scheduler log files
-m  :  don't use MOM log files
-q  :  quiet mode - hide all error messages
-v  :  verbose mode - show more error messages
```

Example

```
> tracejob -n 10 1131

Job: 1131.icluster.org

03/02/2005 17:58:28  S   enqueuing into batch, state 1 hop 1
03/02/2005 17:58:28  S   Job Queued at request of dev@icluster.org, owner =
                         dev@icluster.org, job name = STDIN, queue = batch
03/02/2005 17:58:28  A   queue=batch
03/02/2005 17:58:41  S   Job Run at request of dev@icluster.org
03/02/2005 17:58:41  M   evaluating limits for job
03/02/2005 17:58:41  M   phase 2 of job launch successfully completed
03/02/2005 17:58:41  M   saving task (TMomFinalizeJob3)
03/02/2005 17:58:41  M   job successfully started
03/02/2005 17:58:41  M   job 1131.koa.icluster.org reported successful start on 1 node
(s)
03/02/2005 17:58:41  A   user=dev group=dev jobname=STDIN queue=batch ctime=1109811508

                         qtime=1109811508 etime=1109811508 start=1109811521
                         exec_host=icluster.org/0 Resource_List.neednodes=1 Resource_
List.nodect=1
                         Resource_List.nodes=1 Resource_List.walltime=00:01:40
03/02/2005 18:02:11  M   walltime 210 exceeded limit 100
03/02/2005 18:02:11  M   kill_job
03/02/2005 18:02:11  M   kill_job found a task to kill
03/02/2005 18:02:11  M   sending signal 15 to task
03/02/2005 18:02:11  M   kill_task: killing pid 14060 task 1 with sig 15
03/02/2005 18:02:11  M   kill_task: killing pid 14061 task 1 with sig 15
03/02/2005 18:02:11  M   kill_task: killing pid 14063 task 1 with sig 15
03/02/2005 18:02:11  M   kill_job done
03/02/2005 18:04:11  M   kill_job
03/02/2005 18:04:11  M   kill_job found a task to kill
03/02/2005 18:04:11  M   sending signal 15 to task
03/02/2005 18:06:27  M   kill_job
03/02/2005 18:06:27  M   kill_job done
03/02/2005 18:06:27  M   performing job clean-up
03/02/2005 18:06:27  A   user=dev group=dev jobname=STDIN queue=batch ctime=1109811508

                         qtime=1109811508 etime=1109811508 start=1109811521
                         exec_host=icluster.org/0 Resource_List.neednodes=1 Resource_
List.nodect=1
                         Resource_List.nodes=1 Resource_List.walltime=00:01:40
session=14060
                         end=1109811987 Exit_status=265 resources_used.cput=00:00:00
                         resources_used.mem=3544kb resources_used.vmem=10632kb
                         resources_used.walltime=00:07:46

...
```

The `tracejob` command operates by searching the pbs_server accounting records and the pbs_server, MOM, and scheduler logs. To function properly, it must be run on a node and as a user which can access these files. By default, these files are all accessible by the user root and only available on the cluster management node. In particular, the files required by `tracejob` are located in the following directories:

`TORQUE_HOME/server_priv/accounting`

`TORQUE_HOME/server_logs`

`TORQUE_HOME/mom_logs`

`TORQUE_HOME/sched_logs`

> `tracejob` may only be used on systems where these files are made available. Non-root users may be able to use this command if the permissions on these directories or files are changed appropriately.

> The value of `Resource_List.*` is the amount of resources requested, and the value of `resources_used.*` is the amount of resources actually used.

Related topics

- **Troubleshooting** on page 2316

Using GDB to Locate Job Failures

If either the pbs_mom or pbs_server fail unexpectedly (and the log files contain no information on the failure) gdb can be used to determine whether or not the program is crashing. To start pbs_mom or pbs_server under GDB export the environment variable `PBSDEBUG=yes` and start the program (i.e., `gdb pbs_mom` and then issue the `run` subcommand at the gdb prompt).

GDB may run for some time until a failure occurs and at which point, a message will be printed to the screen and a gdb prompt again made available. If this occurs, use the gdb `where` subcommand to determine the exact location in the code. The information provided may be adequate to allow local diagnosis and correction. If not, this output may be sent to the mailing list or to help for further assistance.

> See the PBSCOREDUMP parameter for enabling creation of core files (see **Using "tracejob" to Locate Job Failures** on page 2319).

Related topics

- **Troubleshooting** on page 2316

Other Diagnostic Options

When *PBSDEBUG* is set, some client commands will print additional diagnostic information.

```
$ export PBSDEBUG=yes
$ cmd
```

To debug different kinds of problems, it can be useful to see where in the code time is being spent. This is called profiling and there is a Linux utility "gprof" that will output a listing of routines and the amount of time spent in these routines. This does require that the code be compiled with special options to instrument the code and to produce a file, gmon.out, that will be written at the end of program execution.

The following listing shows how to build TORQUE with profiling enabled. Notice that the output file for pbs_mom will end up in the `mom_priv` directory because its startup code changes the default directory to this location.

```
# ./configure "CFLAGS=-pg -lgcov -fPIC"
# make -j5
# make install
# pbs_mom ... do some stuff for a while ...
# momctl -s
# cd /var/spool/torque/mom_priv
# gprof -b `which pbs_mom` gmon.out |less
#
```

Another way to see areas where a program is spending most of its time is with the valgrind program. The advantage of using valgrind is that the programs do not have to be specially compiled.

```
# valgrind --tool=callgrind pbs_mom
```

Related topics

- **Troubleshooting** on page 2316

Stuck Jobs

If a job gets stuck in TORQUE, try these suggestions to resolve the issue:

- Use the **qdel** command to cancel the job.
- Force the MOM to send an obituary of the job ID to the server.

  ```
  > qsig -s 0 <JOBID>
  ```

- You can try clearing the stale jobs by using the **momctl** command on the compute nodes where the jobs are still listed.

  ```
  > momctl -c 58925 -h compute-5-20
  ```

- Setting the **qmgr** server setting `mom_job_sync` to *True* might help prevent jobs from hanging.

  ```
  > qmgr -c "set server mom_job_sync = True"
  ```

  To check and see if this is already set, use:

  ```
  > qmgr -c "p s"
  ```

- If the suggestions above cannot remove the stuck job, you can try **qdel** `-p`. However, since the `-p` option purges all information generated by the job, this is not a recommended option unless the above suggestions fail to remove the stuck job.

  ```
  > qdel -p <JOBID>
  ```

- The last suggestion for removing stuck jobs from compute nodes is to restart the pbs_mom.

For additional troubleshooting, run a tracejob on one of the stuck jobs. You can then create an online support ticket with the full server log for the time period displayed in the trace job.

Related topics

- **Troubleshooting** on page 2316

Frequently Asked Questions (FAQ)

Cannot connect to server: error=15034

This error occurs in TORQUE clients (or their APIs) because TORQUE cannot find the `server_name` file and/or the PBS_DEFAULT environment variable is not set. The `server_name` file or PBS_DEFAULT variable indicate the pbs_server's hostname that the client tools should communicate with. The `server_name` file is usually located in TORQUE's local state directory. Make sure the file exists, has proper permissions, and that the version of TORQUE you are running was built with the proper directory settings. Alternatively you can set the PBS_DEFAULT environment variable. Restart TORQUE daemons if you make changes to these settings.

Deleting 'stuck' jobs

To manually delete a "stale" job which has no process, and for which the mother superior is still alive, sending a sig 0 with qsig will often cause MOM to realize the job is stale and issue the proper JobObit notice. Failing that, use **momctl** `-c` to forcefully cause MOM to purge the job. The following process should never be necessary:

- Shut down the MOM on the mother superior node.
- Delete all files and directories related to the job from `TORQUE_HOME/mom_priv/jobs`.
- Restart the MOM on the mother superior node.

If the mother superior MOM has been lost and cannot be recovered (i.e. hardware or disk failure), a job running on that node can be purged from the output of **qstat** using the **qdel** on page 2371 `-p` command or can be removed manually using the following steps:

To remove job X

1. Shut down pbs_server.

```
> qterm
```

2. Remove job spool files.

```
> rm TORQUE_HOME/server_priv/jobs/X.SC TORQUE_HOME/server_priv/jobs/X.JB
```

3. Restart pbs_server

```
> pbs_server
```

Which user must run TORQUE?

TORQUE (pbs_server & pbs_mom) must be started by a user with root privileges.

Scheduler cannot run jobs - rc: 15003

For a scheduler, such as Moab or Maui, to control jobs with TORQUE, the scheduler needs to be run be a user in the server operators / managers list (see **qmgr**). The default for the server operators / managers list is root@localhost. For TORQUE to be used in a grid setting with Silver, the scheduler needs to be run as root.

PBS_Server: pbsd_init, Unable to read server database

If this message is displayed upon starting pbs_server it means that the local database cannot be read. This can be for several reasons. The most likely is a version mismatch. Most versions of TORQUE can read each other's databases. However, there are a few incompatibilities between OpenPBS and TORQUE. Because of enhancements to TORQUE, it cannot read the job database of an OpenPBS server (job structure sizes have been altered to increase functionality). Also, a compiled in 32-bit mode cannot read a database generated by a 64-bit pbs_server and vice versa.

To reconstruct a database (excluding the job database)

1. First, print out the old data with this command:

```
%> qmgr -c "p s"
#
# Create queues and set their attributes.
#
#
# Create and define queue batch
# create queue batch
set queue batch queue_type = Execution
set queue batch acl_host_enable = False
set queue batch resources_max.nodect = 6
set queue batch resources_default.nodes = 1
set queue batch resources_default.walltime = 01:00:00
set queue batch resources_available.nodect = 18
set queue batch enabled = True
set queue batch started = True
#
# Set server attributes.
#
set server scheduling = True
set server managers = griduser@oahu.icluster.org
set server managers += scott@*.icluster.org
set server managers += wightman@*.icluster.org
set server operators = griduser@oahu.icluster.org
set server operators += scott@*.icluster.org
set server operators += wightman@*.icluster.org
set server default_queue = batch
set server log_events = 511
set server mail_from = adm
set server resources_available.nodect = 80
set server node_ping_rate = 300
set server node_check_rate = 600
set server tcp_timeout = 6
```

2. Copy this information somewhere.

3. Restart pbs_server with the following command:

```
> pbs_server -t create
```

4. When you are prompted to overwrite the previous database, enter *y*, then enter the data exported by the qmgr command as in this example:

```
> cat data | qmgr
```

5. Restart pbs_server without the flags:

```
> qterm
> pbs_server
```

This will reinitialize the database to the current version.

> Reinitializing the server database will reset the next jobid to 1

qsub will not allow the submission of jobs requesting many processors

TORQUE's definition of a node is context sensitive and can appear inconsistent. The **qsub** `-l nodes=<X>` expression can at times indicate a request for X processors and other time be interpreted as a request for X nodes. While qsub allows multiple interpretations of the keyword nodes, aspects of the TORQUE server's logic are not so flexible. Consequently, if a job is using `-l nodes` to specify processor count and the requested number of processors exceeds the available number of physical nodes, the server daemon will reject the job.

To get around this issue, the server can be told it has an inflated number of nodes using the `resources_available` attribute. To take effect, this attribute should be set on both the server and the associated queue as in the example below. (See **resources_available** for more information.)

```
> qmgr
Qmgr: set server resources_available.nodect=2048
Qmgr: set queue batch resources_available.nodect=2048
```

The pbs_server daemon will need to be restarted before these changes will take effect.

qsub reports 'Bad UID for job execution'

```
[guest@login2]$ qsub test.job
qsub: Bad UID for job execution
```

Job submission hosts must be explicitly specified within TORQUE or enabled via RCmd security mechanisms in order to be trusted. In the example above, the host '`login2`' is not configured to be trusted. This process is documented in **Configuring job submission hosts** on page 2216.

Why does my job keep bouncing from running to queued?

There are several reasons why a job will fail to start. Do you see any errors in the MOM logs? Be sure to increase the loglevel on MOM if you don't see anything. Also be sure TORQUE is configured with `--enable-syslog` and look in `/var/log/messages` (or wherever your syslog writes).

Also verify the following on all machines:

- DNS resolution works correctly with matching forward and reverse
- Time is synchronized across the head and compute nodes
- User accounts exist on all compute nodes
- User home directories can be mounted on all compute nodes
- Prologue scripts (if specified) exit with 0

If using a scheduler such as Moab or Maui, use a scheduler tool such as checkjob to identify job start issues.

How do I use PVM with TORQUE?

- Start the master pvmd on a compute node and then add the slaves
- mpiexec can be used to launch slaves using rsh or ssh (use export `PVM_RSH=/usr/bin/ssh` to use ssh)

> Access can be managed by rsh/ssh without passwords between the batch nodes, but denying it from anywhere else, including the interactive nodes. This can be done with xinetd and sshd configuration (root is allowed to ssh everywhere). This way, the pvm daemons can be started and killed from the job script.

The problem is that this setup allows the users to bypass the batch system by writing a job script that uses rsh/ssh to launch processes on the batch nodes. If there are relatively few users and they can more or less be trusted, this setup can work.

My build fails attempting to use the TCL library

TORQUE builds can fail on TCL dependencies even if a version of TCL is available on the system. TCL is only utilized to support the xpbsmon client. If your site does not use this tool (most sites do not use xpbsmon), you can work around this failure by rerunning `configure` with the `--disable-gui` argument.

My job will not start, failing with the message 'cannot send job to mom, state=PRERUN'

If a node crashes or other major system failures occur, it is possible that a job may be stuck in a corrupt state on a compute node. TORQUE 2.2.0 and higher automatically handle this when the `mom_job_sync` parameter is set via **qmgr** (the default). For earlier versions of TORQUE, set this parameter and restart the pbs_mom daemon.

This error can also occur if not enough free space is available on the partition that holds TORQUE.

How do I determine what version of TORQUE I am using?

There are times when you want to find out what version of TORQUE you are using. An easy way to do this is to run the following command:

```
qmgr

> qmgr -c "p s" | grep pbs_ver
```

How do I resolve autogen.sh errors that contain "error: possibly undefined macro: AC_MSG_ERROR"?

Verify the `pkg-config` package is installed.

How do I resolve compile errors with libssl or libcrypto for TORQUE 4.0 on Ubuntu 10.04?

When compiling TORQUE 4.0 on Ubuntu 10.04 the following errors might occur:

```
libtool: link: gcc -Wall -pthread -g -D_LARGEFILE64_SOURCE -o .libs/trqauthd trq_auth_
daemon.o trq_main.o -ldl -lssl -lcrypto -L/home/adaptive/torques/torque-
4.0.0/src/lib/Libpbs/.libs /home/adaptive/torques/torque-
4.0.0/src/lib/Libpbs/.libs/libtorque.so -lpthread -lrt -pthread
/usr/bin/ld: cannot find -lssl
collect2: ld returned 1 exit status
make[3]: *** [trqauthd] Error 1

libtool: link: gcc -Wall -pthread -g -D_LARGEFILE64_SOURCE -o .libs/trqauthd trq_auth_
daemon.o trq_main.o -ldl -lssl -lcrypto -L/home/adaptive/torques/torque-
4.0.0/src/lib/Libpbs/.libs /home/adaptive/torques/torque-
4.0.0/src/lib/Libpbs/.libs/libtorque.so -lpthread -lrt -pthread
/usr/bin/ld: cannot find -lcrypto
collect2: ld returned 1 exit status
make[3]: *** [trqauthd] Error 1
```

To resolve the compile issue, use these commands:

```
> cd /usr/lib
> ln -s /lib/libcrypto.so.0.9. libcrypto.so
> ln -s /lib/libssl.so.0.9.8 libssl.so
```

Why are there so many error messages in the client logs (trqauthd logs) when I don't notice client commands failing?

If a client makes a connection to the server and the trqauthd connection for that client command is authorized *before* the client's connection, the trqauthd connection is rejected. The connection is retried, but if all retry attempts are rejected, trqauthd logs a message indicating a failure. Some client commands then open a new connection to the server and try again. The client command fails only if all its retries fail.

Related topics

- **Troubleshooting** on page 2316

Compute Node Health Check

TORQUE provides the ability to perform health checks on each compute node. If these checks fail, a failure message can be associated with the node and routed to the scheduler. Schedulers (such as Moab) can forward this information to administrators by way of scheduler triggers, make it available through scheduler diagnostic commands, and automatically mark the node down until the issue is resolved. (See the RMMSGIGNORE parameter in **Appendix A: Moab Parameters** on page 902 for more information.)

Additionally, Michael Jennings at LBNL has authored an open-source bash node health check script project. It offers an easy way to perform some of the most common node health checking tasks, such as verifying network and filesystem functionality. More information is available on the project's page.

For more information about node health checks, see these topics:

Related topics

- **Troubleshooting** on page 2316

Configuring MOMs to Launch a Health Check

The health check feature is configured via the `mom_priv/config` file using the parameters described below:

Parameter	Format	Default	Description
$node_ check_ script	<STRING>	N/A	(Required) Specifies the fully qualified pathname of the health check script to run
$node_ check_inter- val	<INTEGER>	1	(Optional) Specifies the number of MOM intervals between health checks (by default, each MOM interval is 45 seconds long - this is controlled via the **$status_update_time** on page 2450 node para- meter. The integer may be followed by a list of event names (cur- rently supported are `jobstart` and `jobend`). (For more information, see **pbs_mom**.)

Related topics

- **Compute Node Health Check** on page 2328

Creating the Health Check Script

The health check script is executed directly by the pbs_mom daemon under the root user id. It must be accessible from the compute node and may be a script or compile executable program. It may make any needed system calls and execute any combination of system utilities but should not execute resource manager client commands. Also, as of TORQUE 1.0.1, the pbs_mom daemon blocks until the health check is completed and does not possess a built-in timeout. Consequently, it is advisable to keep the launch script execution time short and verify that the script will not block even under failure conditions.

If the script detects a failure, it should return the keyword **ERROR** to stdout followed by an error message. When a failure is detected, the ERROR keyword should be printed to stdout before any other data. The message (up to 1024 characters) immediately following the ERROR keyword must all be contained on the same line. The message is assigned to the node attribute 'message' of the associated node.

Related topics

- **Compute Node Health Check** on page 2328

Adjusting Node State Based on the Health Check Output

If the health check reports an error, the node attribute "message" is set to the error string returned. Cluster schedulers can be configured to adjust a given node's state based on this information. For example, by default, Moab sets a node's state to down if a node error message is detected. The node health script continues to run at the configured interval (see **Configuring MOMs to Launch a Health Check** on page 2329 for more information), and if it does not generate the error message again during one of its later executions, Moab picks that up at the beginning of its next iteration and restores the node to an online state.

Related topics

- **Compute Node Health Check** on page 2328

Example Health Check Script

As mentioned, the health check can be a shell script, PERL, Python, C-executable, or anything which can be executed from the command line capable of setting STDOUT. The example below demonstrates a very simple health check:

```
#!/bin/sh
/bin/mount | grep global
if [ $? != "0" ]
  then
    echo "ERROR cannot locate filesystem global"
fi
```

Related topics

- **Compute Node Health Check** on page 2328

Debugging

TORQUE supports a number of diagnostic and debug options including the following:

PBSDEBUG environment variable - If set to 'yes', this variable will prevent pbs_server, pbs_mom, and/or pbs_sched from backgrounding themselves allowing direct launch under a debugger. Also, some client commands will provide additional diagnostic information when this value is set.

PBSLOGLEVEL environment variable - Can be set to any value between 0 and 7 and specifies the logging verbosity level (default = 0)

PBSCOREDUMP environment variable - If set, it will cause the offending resource manager daemon to create a core file if a *SIGSEGV*, *SIGILL*, *SIGFPE*, *SIGSYS*, or *SIGTRAP* signal is received. The core dump will be placed in the daemon's home directory (`$PBSHOME/mom_priv` for pbs_mom and `$PBSHOME/server_priv` for pbs_server).

To enable core dumping in a Red Hat system, you must add the following line to the `/etc/init.d/pbs_mom` and `/etc/init.d/pbs_server` scripts:

```
export DAEMON_COREFILE_LIMIT=unlimited
```

NDEBUG #define - if set at build time, will cause additional low-level logging information to be output to stdout for pbs_server and pbs_mom daemons.

tracejob reporting tool - can be used to collect and report logging and accounting information for specific jobs (for more information, see **Using "tracejob" to Locate Job Failures** on page 2319).

PBSLOGLEVEL and *PBSCOREDUMP* must be added to the `$PBSHOME/pbs_environment` file, not just the current environment. To set these variables, add a line to the pbs_environment file as either "variable=value" or just "variable". In the case of "variable=value", the environment variable is set up as the value specified. In the case of "variable", the environment variable is set based upon its value in the current environment.

TORQUE error codes

Error code name	Number	Description
PBSE_FLOOR	15000	No error
PBSE_UNKJOBID	15001	Unknown job identifier
PBSE_NOATTR	15002	Undefined attribute
PBSE_ATTRRO	15003	Attempt to set READ ONLY attribute
PBSE_IVALREQ	15004	Invalid request
PBSE_UNKREQ	15005	Unknown batch request
PBSE_TOOMANY	15006	Too many submit retries
PBSE_PERM	15007	No permission
PBSE_IFF_NOT_FOUND	15008	"pbs_iff" not found; unable to authenticate
PBSE_MUNGE_NOT_FOUND	15009	"munge" executable not found; unable to authenticate
PBSE_BADHOST	15010	Access from host not allowed

Error code name	Number	Description
PBSE_JOBEXIST	15011	Job already exists
PBSE_SYSTEM	15012	System error occurred
PBSE_INTERNAL	15013	Internal server error occurred
PBSE_REGROUTE	15014	Parent job of dependent in rte queue
PBSE_UNKSIG	15015	Unknown signal name
PBSE_BADATVAL	15016	Bad attribute value
PBSE_MODATRRUN	15017	Cannot modify attribute in run state
PBSE_BADSTATE	15018	Request invalid for job state
PBSE_UNKQUE	15020	Unknown queue name
PBSE_BADCRED	15021	Invalid credential in request
PBSE_EXPIRED	15022	Expired credential in request
PBSE_QUNOENB	15023	Queue not enabled
PBSE_QACESS	15024	No access permission for queue
PBSE_BADUSER	15025	Bad user - no password entry
PBSE_HOPCOUNT	15026	Max hop count exceeded
PBSE_QUEEXIST	15027	Queue already exists
PBSE_ATTRTYPE	15028	Incompatible queue attribute type
PBSE_QUEBUSY	15029	Queue busy (not empty)
PBSE_QUENBIG	15030	Queue name too long
PBSE_NOSUP	15031	Feature/function not supported

Error code name	Number	Description
PBSE_QUENOEN	15032	Cannot enable queue,needs add def
PBSE_PROTOCOL	15033	Protocol (ASN.1) error
PBSE_BADATLST	15034	Bad attribute list structure
PBSE_NOCONNECTS	15035	No free connections
PBSE_NOSERVER	15036	No server to connect to
PBSE_UNKRESC	15037	Unknown resource
PBSE_EXCQRESC	15038	Job exceeds queue resource limits
PBSE_QUENODFLT	15039	No default queue defined
PBSE_NORERUN	15040	Job not rerunnable
PBSE_ROUTEREJ	15041	Route rejected by all destinations
PBSE_ROUTEEXPD	15042	Time in route queue expired
PBSE_MOMREJECT	15043	Request to MOM failed
PBSE_BADSCRIPT	15044	(qsub) Cannot access script file
PBSE_STAGEIN	15045	Stage-In of files failed
PBSE_RESCUNAV	15046	Resources temporarily unavailable
PBSE_BADGRP	15047	Bad group specified
PBSE_MAXQUED	15048	Max number of jobs in queue
PBSE_CKPBSY	15049	Checkpoint busy, may be retries
PBSE_EXLIMIT	15050	Limit exceeds allowable
PBSE_BADACCT	15051	Bad account attribute value

Error code name	Number	Description
PBSE_ALRDYEXIT	15052	Job already in exit state
PBSE_NOCOPYFILE	15053	Job files not copied
PBSE_CLEANEDOUT	15054	Unknown job id after clean init
PBSE_NOSYNCMSTR	15055	No master in sync set
PBSE_BADDEPEND	15056	Invalid dependency
PBSE_DUPLIST	15057	Duplicate entry in list
PBSE_DISPROTO	15058	Bad DIS based request protocol
PBSE_EXECTHERE	15059	Cannot execute there
PBSE_SISREJECT	15060	Sister rejected
PBSE_SISCOMM	15061	Sister could not communicate
PBSE_SVRDOWN	15062	Requirement rejected -server shutting down
PBSE_CKPSHORT	15063	Not all tasks could checkpoint
PBSE_UNKNODE	15064	Named node is not in the list
PBSE_UNKNODEATR	15065	Node-attribute not recognized
PBSE_NONODES	15066	Server has no node list
PBSE_NODENBIG	15067	Node name is too big
PBSE_NODEEXIST	15068	Node name already exists
PBSE_BADNDATVAL	15069	Bad node-attribute value
PBSE_MUTUALEX	15070	State values are mutually exclusive
PBSE_GMODERR	15071	Error(s) during global modification of nodes

Error code name	Number	Description
PBSE_NORELYMOM	15072	Could not contact MOM
PBSE_NOTSNODE	15073	No time-shared nodes
PBSE_JOBTYPE	15074	Wrong job type
PBSE_BADACLHOST	15075	Bad ACL entry in host list
PBSE_MAXUSERQUED	15076	Maximum number of jobs already in queue for user
PBSE_BADDISALLOWTYPE	15077	Bad type in "disallowed_types" list
PBSE_NOINTERACTIVE	15078	Interactive jobs not allowed in queue
PBSE_NOBATCH	15079	Batch jobs not allowed in queue
PBSE_NORERUNABLE	15080	Rerunable jobs not allowed in queue
PBSE_NONONRERUNABLE	15081	Non-rerunable jobs not allowed in queue
PBSE_UNKARRAYID	15082	Unknown array ID
PBSE_BAD_ARRAY_REQ	15083	Bad job array request
PBSE_TIMEOUT	15084	Time out
PBSE_JOBNOTFOUND	15085	Job not found
PBSE_NOFAULTTOLERANT	15086	Fault tolerant jobs not allowed in queue
PBSE_NOFAULTINTOLERANT	15087	Only fault tolerant jobs allowed in queue
PBSE_NOJOBARRAYS	15088	Job arrays not allowed in queue
PBSE_RELAYED_TO_MOM	15089	Request was relayed to a MOM
PBSE_MEM_MALLOC	15090	Failed to allocate memory for memmgr
PBSE_MUTEX	15091	Failed to allocate controlling mutex (lock/unlock)

Error code name	Number	Description
PBSE_TRHEADATTR	15092	Failed to set thread attributes
PBSE_THREAD	15093	Failed to create thread
PBSE_SELECT	15094	Failed to select socket
PBSE_SOCKET_FAULT	15095	Failed to get connection to socket
PBSE_SOCKET_WRITE	15096	Failed to write data to socket
PBSE_SOCKET_READ	15097	Failed to read data from socket
PBSE_SOCKET_CLOSE	15098	Socket closed
PBSE_SOCKET_LISTEN	15099	Failed to listen in on socket
PBSE_AUTH_INVALID	15100	Invalid auth type in request
PBSE_NOT_IMPLEMENTED	15101	Functionality not yet implemented
PBSE_QUENOTAVAILABLE	15102	Queue is not available

Related topics

- **Troubleshooting** on page 2316

Appendices

The appendices provide tables of commands, parameters, configuration options, error codes, the Quick Start Guide, and so forth.

Appendix A: Commands Overview

Client commands

Command	Description
momctl	Manage/diagnose MOM (node execution) daemon
pbsdsh	Launch tasks within a parallel job
pbsnodes	View/modify batch status of compute nodes
qalter	Modify queued batch jobs
qchkpt	Checkpoint batch jobs
qdel	Delete/cancel batch jobs
qgpumode	Specifies new mode for GPU
qgpureset	Reset the GPU
qhold	Hold batch jobs
qmgr	Manage policies and other batch configuration
qmove on page 2380	Move batch jobs

Command	Description
qorder on page 2381	Exchange order of two batch jobs in any queue
qrerun	Rerun a batch job
qrls	Release batch job holds
qrun	Start a batch job
qsig	Send a signal to a batch job
qstat	View queues and jobs
qsub	Submit jobs
qterm	Shutdown pbs server daemon
tracejob	Trace job actions and states recorded in TORQUE logs (see **Using "tracejob" to Locate Job Failures** on page 2319)

Binary executables

Command	Description
pbs_iff	Interprocess authentication service
pbs_mom	Start MOM (node execution) daemon
pbs_server	Start server daemon
pbs_track	Tell pbs_mom to track a new process

Related topics

- **Appendix C: Node Manager (MOM) Configuration** on page 2435
- **Appendix B: Server Parameters** on page 2417

momctl

(PBS MOM Control)

Synopsis

```
momctl -c { <JOBID> | all }
momctl -C
momctl -d { <INTEGER> | <JOBID> }
momctl -f <FILE>
momctl -h <HOST>[,<HOST>]...
momctl -p <PORT_NUMBER>
momctl -q <ATTRIBUTE>
momctl -r { <FILE> | LOCAL:<FILE> }
momctl -s
```

Overview

The momctl command allows remote shutdown, reconfiguration, diagnostics, and querying of the pbs_mom daemon.

Format

-c — Clear	
Format	{ <JOBID> | *all* }
Default	---
Description	Makes the MOM unaware of the job's existence. It does not clean up any processes associated with the job.
Example	`momctl - node1 -c 15406`

-C — Cycle	
Format	---
Default	---
Description	Cycle pbs_mom(s)
Example	`momctl - node1 -C` Cycle pbs_mom on node1.

-d — Diagnose	
Format	{ <INTEGER> | <JOBID> }
Default	0
Description	Diagnose MOM(s) (For more details, see **Diagnose detail** on page 2343 below.)
Example	`momctl - node1 -d 2` Print level 2 and lower diagnose information for the MOM on node1.

-f — Host File	
Format	<FILE>
Default	---
Description	A file containing only comma or whitespace (space, tab, or new line) delimited hostnames
Example	`momctl -f hosts.txt -d` Print diagnose information for the MOMs running on the hosts specified in `hosts.txt`.

-h — Host List	
Format	<HOST>[,<HOST>]...
Default	localhost
Description	A comma separated list of hosts
Example	`momctl -h node1,node2,node3 -d` Print diagnose information for the MOMs running on node1, node2, and node3.

-p — Port	
Format	<PORT_NUMBER>

-p — Port	
Default	TORQUE's default port number
Description	The port number for the specified MOM(s)
Example	`momctl -p 5455 -h node1 -d` Request diagnose information over port 5455 on node1.

-q — Query	
Format	<ATTRIBUTE>
Default	---
Description	Query <ATTRIBUTE> on specified MOM, where *<ATTRIBUTE>* is a property listed by **pbsnodes** `-a` (see **Query attributes** on page 2342 for a list of attributes)
Example	`momctl -q physmem` Print the amount of physmem on localhost.

-r — Reconfigure	
Format	{ <FILE> | LOCAL:<FILE> }
Default	---
Description	Reconfigure MOM(s) with remote or local config file, <FILE>. This does not work if $remote_reconfig is not set to true when the MOM is started.
Example	`momctl -r /home/user1/new.config -h node1` Reconfigure MOM on node1 with `/home/user1/new.cofig` on node1.

-s — Shutdown	
Format	
Default	---

-s — Shutdown	
Description	Shutdown pbs_mom
Example	`momctl -s` Terminates pbs_mom process on localhost.

Query attributes

Attribute	Description
arch	node hardware architecture
availmem	available RAM
loadave	1 minute load average
ncpus	number of CPUs available on the system
netload	total number of bytes transferred over all network interfaces
nsessions	number of sessions active
nusers	number of users active
physmem	configured RAM
sessions	list of active sessions
totmem	configured RAM plus configured swap

Diagnose detail

Level	Description
0	Display the following information: • Local hostname • Expected server hostname • Execution version • MOM home directory • MOM config file version (if specified) • Duration MOM has been executing • Duration since last request from pbs_server daemon • Duration since last request to pbs_server daemon • RM failure messages (if any) • Log verbosity level • Local job list
1	All information for level 0 plus the following: • Interval between updates sent to server • Number of initialization messages sent to pbs_server daemon • Number of initialization messages received from pbs_server daemon • Prolog/epilog alarm time • List of trusted clients
2	All information from level 1 plus the following: • PID • Event alarm status
3	All information from level 2 plus the following: • syslog enabled

Example 4-27: MOM diagnostics

```
momctl -d 1

Host: nsrc/nsrc.fllcl.com    Server: 10.10.10.113    Version: torque_1.1.0p4
HomeDirectory:          /usr/spool/PBS/mom_priv
ConfigVersion:          147
MOM active:             7390 seconds
Last Msg From Server:   7389 seconds (CLUSTER_ADDRS)
Server Update Interval: 20 seconds
Server Update Interval: 20 seconds
Init Msgs Received:     0 hellos/1 cluster-addrs
Init Msgs Sent:         1 hellos
LOGLEVEL:               0 (use SIGUSR1/SIGUSR2 to adjust)
Prolog Alarm Time:      300 seconds
Trusted Client List:    12.14.213.113,127.0.0.1
JobList:                NONE

diagnostics complete
```

Example 4-28: System shutdown

```
> momctl -s -f /opt/clusterhostfile

shutdown request successful on node001
shutdown request successful on node002
shutdown request successful on node003
shutdown request successful on node004
shutdown request successful on node005
shutdown request successful on node006
```

pbs_mom

Start a pbs batch execution mini-server.

Synopsis

```
pbs_mom [-a alarm] [-A alias] [-C chkdirectory] [-c config] [-d directory] [-h
hostname]
 [-L logfile] [-M MOMport] [-R RPPport] [-p|-r] [-P purge] [-w] [-x]
```

Description

The pbs_mom command is located within the `TORQUE_HOME` directory and starts the operation of a batch Machine Oriented Mini-server (MOM) on the execution host. To ensure that the pbs_mom command is not runnable by the general user community, the server will only execute if its real and effective uid is zero.

The first function of pbs_mom is to place jobs into execution as directed by the server, establish resource usage limits, monitor the job's usage, and notify the server when the job completes. If they exist, pbs_mom will execute a prologue script before executing a job and an epilogue script after executing the job.

The second function of pbs_mom is to respond to resource monitor requests. This was done by a separate process in previous versions of PBS but has now been combined into one process. It provides information about the status of running jobs, memory available, etc.

The last function of pbs_mom is to respond to task manager requests. This involves communicating with running tasks over a TCP socket as well as communicating with other MOMs within a job (a.k.a. a "sisterhood").

pbs_mom will record a diagnostic message in a log file for any error occurrence. The log files are maintained in the mom_logs directory below the home directory of the server. If the log file cannot be opened, the diagnostic message is written to the system console.

Options

Flag	Name	Description
-a	alarm	Used to specify the alarm timeout in seconds for computing a resource. Every time a resource request is processed, an alarm is set for the given amount of time. If the request has not completed before the given time, an alarm signal is generated. The default is 5 seconds.
-A	alias	Used to specify this multimom's alias name. The alias name needs to be the same name used in the mom.hierarchy file. It is only needed when running multiple MOMs on the same machine. For more information, see **TORQUE Multi-MOM** on page 2231.
-C	chkdirectory	Specifies The path of the directory used to hold checkpoint files. (Currently this is only valid on Cray systems.) The default directory is `TORQUE_HOME/spool/checkpoint` (see the **-d** option). The directory specified with the -C option must be owned by root and accessible (rwx) only by root to protect the security of the checkpoint files.
-c	config	Specifies an alternative configuration file, see description below. If this is a relative file name it will be relative to `TORQUE_HOME/mom_priv`, (see the **-d** option). If the specified file cannot be opened, pbs_mom will abort. If the **-C** option is not supplied, pbs_mom will attempt to open the default configuration file "config" in `TORQUE_HOME/mom_priv`. If this file is not present, pbs_mom will log the fact and continue.
-d	directory	Specifies the path of the directory which is the home of the server's working files, `TORQUE_HOME`. This option is typically used along with **-M** when debugging MOM. The default directory is given by `$PBS_SERVER_HOME` which is typically `/usr/spool/PBS`.
-h	hostname	Set MOM's hostname. This can be useful on multi-homed networks.
-L	logfile	Specify an absolute path name for use as the log file. If not specified, MOM will open a file named for the current date in the `TORQUE_HOME/mom_logs` directory (see the **-d** option).
-M	port	Specifies the port number on which the mini-server (MOM) will listen for batch requests.

Flag	Name	Description
-p	n/a	Specifies the impact on jobs which were in execution when the mini-server shut down. On any restart of MOM, the new mini-server will not be the parent of any running jobs, MOM has lost control of her offspring (not a new situation for a mother). With the -p option, MOM will allow the jobs to continue to run and monitor them indirectly via polling. This flag is redundant in that this is the default behavior when starting the server. The -p option is mutually exclusive with the **-R** and **-q** options.
-P	purge	Specifies the impact on jobs which were in execution when the mini-server shut down. With the -P option, it is assumed that either the entire system has been restarted or the MOM has been down so long that it can no longer guarantee that the pid of any running process is the same as the recorded job process pid of a recovering job. Unlike the **-p** option, no attempt is made to try and preserve or recover running jobs. All jobs are terminated and removed from the queue.
-q	n/a	Specifies the impact on jobs which were in execution when the mini-server shut down. With the -q option, MOM will allow the processes belonging to jobs to continue to run, but will not attempt to monitor them. The -q option is mutually exclusive with the **-p** and **-R** options.
-R	port	Specifies the port number on which the mini-server (MOM) will listen for resource monitor requests, task manager requests and inter-MOM messages. Both a UDP and a TCP port of this number will be used.
-r	n/a	Specifies the impact on jobs which were in execution when the mini-server shut down. With the -r option, MOM will kill any processes belonging to jobs, mark the jobs as terminated, and notify the batch server which owns the job. The -r option is mutually exclusive with the **-p** and **-q** options. Normally the mini-server is started from the system boot file without the **-p** or the -r option. The mini-server will make no attempt to signal the former session of any job which may have been running when the mini-server terminated. It is assumed that on reboot, all processes have been killed. If the -r option is used following a reboot, process IDs (pids) may be reused and MOM may kill a process that is not a batch session.
-w	wait_for_server	When started with -w, pbs_moms wait until they get their MOM hierarchy file from pbs_server to send their first update, or until 10 minutes pass. This reduces network traffic on startup and can bring up clusters faster.
-x	n/a	Disables the check for privileged port resource monitor connections. This is used mainly for testing since the privileged port is the only mechanism used to prevent any ordinary user from connecting.

Configuration file

The configuration file, located at `mom_priv/config` by default, can be specified on the command line at program start with the **-C** flag. The use of this file is to provide several types of run time information to pbs_mom: static resource names and values, external resources provided by a program to be run on request via a shell escape, and values to pass to internal set up functions at initialization (and re-initialization).

See the **Parameters** on page 2435 page for a full list of pbs_mom parameters.

Each item type is on a single line with the component parts separated by white space. If the line starts with a hash mark (pound sign, *#*), the line is considered to be a comment and is skipped.

Static Resources

For static resource names and values, the configuration file contains a list of resource names/values pairs, one pair per line and separated by white space. An example of static resource names and values could be the number of tape drives of different types and could be specified by:

- tape3480 4
- tape3420 2
- tapedat 1
- tape8mm 1

Shell Commands

If the first character of the value is an exclamation mark (*!*), the entire rest of the line is saved to be executed through the services of the system(3) standard library routine.

The shell escape provides a means for the resource monitor to yield arbitrary information to the scheduler. Parameter substitution is done such that the value of any qualifier sent with the query, as explained below, replaces a token with a percent sign (*%*) followed by the name of the qualifier. For example, here is a configuration file line which gives a resource name of "escape":

```
escape !echo %xxx %yyy
```

If a query for "escape" is sent with no qualifiers, the command executed would be `echo %xxx %yyy`.

If one qualifier is sent, `escape[xxx=hi there]`, the command executed would be `echo hi there %yyy`.

If two qualifiers are sent, `escape[xxx=hi][yyy=there]`, the command executed would be `echo hi there`.

If a qualifier is sent with no matching token in the command line, `escape[zzz=snafu]`, an error is reported.

Resources

Resource Manager queries can be made with **momctl** `–q` options to retrieve and set pbs_mom options. Any configured static resource may be retrieved with a request of the same name. These are resource requests not otherwise documented in the PBS ERS.

Request	Description
cycle	Forces an immediate MOM cycle.
status_update_time	Retrieve or set the $status_update_time parameter.
check_poll_time	Retrieve or set the $check_poll_time parameter.
configversion	Retrieve the config version.
jobstartblocktime	Retrieve or set the $jobstartblocktime parameter.
enablemomrestart	Retrieve or set the $enablemomrestart parameter.
loglevel	Retrieve or set the $loglevel parameter.
down_on_error	Retrieve or set the EXPERIMENTAL $down_on_error parameter.
diag0 - diag4	Retrieves varied diagnostic information.
rcpcmd	Retrieve or set the $rcpcmd parameter.
version	Retrieves the pbs_mom version.

Health check

The health check script is executed directly by the pbs_mom daemon under the root user id. It must be accessible from the compute node and may be a script or compiled executable program. It may make any needed system calls and execute any combination of system utilities but should not execute resource manager client commands. Also, the pbs_mom daemon blocks until the health check is completed and does not possess a built-in timeout. Consequently, it is advisable to keep the launch script execution time short and verify that the script will not block even under failure conditions.

If the script detects a failure, it should return the keyword "Error" to `stdout` followed by an error message. The message (up to 256 characters) immediately following the Error string will be assigned to the node attribute message of the associated node.

If the script detects a failure when run from "jobstart", then the job will be rejected. You can use this behavior with an advanced scheduler, such as Moab Workload Manager, to cause the job to be routed to another node. TORQUE currently ignores Error messages by default, but you can configure an advanced scheduler to react appropriately.

If the experimental `$down_on_error` MOM setting is enabled, the MOM will set itself to state down and report to pbs_server. Additionally, the experimental `$down_on_error server` attribute can be enabled which has the same effect but moves the decision to pbs_server. It is redundant to have MOM's

`$down_on_error` and pbs_servers `down_on_error` features enabled. See "down_on_error" in pbs_server_attributes(7B).

Files

File	Description
$PBS_SERVER_HOME/server_name	Contains the hostname running pbs_server
$PBS_SERVER_HOME/mom_priv	The default directory for configuration files, typically (`/usr/spool/pbs`)/`mom_priv`
$PBS_SERVER_HOME/mom_logs	Directory for log files recorded by the server
$PBS_SERVER_HOME/mom_priv/-prologue	The administrative script to be run before job execution
$PBS_SERVER_HOME/mom_priv/e-pilogue	The administrative script to be run after job execution

Signal handling

pbs_mom handles the following signals:

Signal	Description
SIGHUP	Causes pbs_mom to re-read its configuration file, close and reopen the log file, and reinitialize resource structures.
SIGALRM	Results in a log file entry. The signal is used to limit the time taken by certain children processes, such as the prologue and epilogue.
SIGINT and SIGTERM	Results in pbs_mom exiting without terminating any running jobs. This is the action for the following signals as well: SIGXCPU, SIGXFSZ, SIGCPULIM, and SIGSHUTDN.
SIGUSR1, SIGUSR2	Causes the MOM to increase and decrease logging levels, respectively.
SIGPIPE, SIGINFO	Are ignored.
SIGBUS, SIGFPE, SIGILL, SIGTRAP, and SIGSYS	Cause a core dump if the PBSCOREDUMP environmental variable is defined.

All other signals have their default behavior installed.

Exit status

If the pbs_mom command fails to begin operation, the server exits with a value greater than zero.

Related topics

- pbs_server(8B)

Non-Adaptive Computing topics

- pbs_scheduler_basl(8B)
- pbs_scheduler_tcl(8B)
- PBS External Reference Specification
- PBS Administrators Guide

pbs_server

(*PBS Server*) pbs batch system manager

Synopsis

```
pbs_server [-a active] [-c] [-d config_path] [-f force overwrite] [-p port] [-
A acctfile]
[-l location] [-L logfile] [-S scheduler_port]
[-H hostname] [-t type] [--ha]
[-n don't send hierarchy] [--about] [-v] [--version]
```

Description

The pbs_server command starts the operation of a batch server on the local host. Typically, this command will be in a local boot file such as `/etc/rc.local`. If the batch server is already in execution, pbs_server will exit with an error. To ensure that the pbs_server command is not runnable by the general user community, the server will only execute if its real and effective uid is zero.

The server will record a diagnostic message in a log file for any error occurrence. The log files are maintained in the server_logs directory below the home directory of the server. If the log file cannot be opened, the diagnostic message is written to the system console.

As of TORQUE 4.0, the pbs_server is multi-threaded which leads to quicker response to client commands, is more robust, and allows for higher job throughput.

Options

Option	Name	Description
-A	acctfile	Specifies an absolute path name of the file to use as the accounting file. If not specified, the file name will be the current date in the `PBS_HOME/server_priv/accounting` directory.
-a	active	Specifies if scheduling is active or not. This sets the server attribute scheduling. If the option argument is "true" ("True", "t", "T", or "1"), the server is active and the PBS job scheduler will be called. If the argument is "false" ("False", "f", "F", or "0), the server is idle, and the scheduler will not be called and no jobs will be run. If this option is not specified, the server will retain the prior value of the scheduling attribute.
-c	wait_for_ moms	This directs pbs_server to send the MOM hierarchy only to MOMs that request it for the first 10 minutes. After 10 minutes, it attempts to send the MOM hierarchy to MOMs that haven't requested it already. This greatly reduces traffic on start up.
-d	config_dir- ectory	Specifies the path of the directory which is home to the server's configuration files, PBS_HOME. A host may have multiple servers. Each server must have a different configuration directory. The default configuration directory is given by the symbol `$PBS_ SERVER_HOME` which is typically `var/spool/torque`.
-f	force over- write	Forces an overwrite of the server database. This can be useful to bypass the `yes/no` prompt when running something like pbs_server `-t create` and can ease installation and configuration of TORQUE via scripts.
-H	hostname	Causes the server to start under a different hostname as obtained from gethostname (2). Useful for servers with multiple network interfaces to support connections from clients over an interface that has a hostname assigned that differs from the one that is returned by gethost name(2).
--ha	high_avail- ability	Starts server in high availability mode (for details, see **Server High Availability** on page 2291).
-L	logfile	Specifies an absolute path name of the file to use as the log file. If not specified, the file will be the current date in the `PBS_HOME/server_logs` directory (see the **-d** option).
-l	location	Specifies where to find Moab when it does not reside on the same host as TORQUE.
-n	no send	This directs pbs_server to not send the hierarchy to all the MOMs on startup. Instead, the hierarchy is only sent if a MOM requests it. This flag works only in conjunction with the local MOM hierarchy feature.

Option	Name	Description
-p	port	Specifies the port number on which the server will listen for batch requests. If multiple servers are running on a single host, each must have its own unique port number. This option is for use in testing with multiple batch systems on a single host.
-S	scheduler_ port	Specifies the port number to which the server should connect when contacting the scheduler. The argument scheduler_conn is of the same syntax as under the -M option.
-t	type	Specifies the impact on jobs which were in execution, running, when the server shut down. If the running job is not rerunnable or restartable from a checkpoint image, the job is aborted. If the job is rerunnable or restartable, then the actions described below are taken. When the type argument is: • *hot* – All jobs are requeued except non-rerunnable jobs that were executing. Any rerunnable job which was executing when the server went down will be run immediately. This returns the server to the same state as when it went down. After those jobs are restarted, then normal scheduling takes place for all remaining queued jobs. If a job cannot be restarted immediately because of a missing resource, such as a node being down, the server will attempt to restart it periodically for up to 5 minutes. After that period, the server will revert to a normal state, as if warm started, and will no longer attempt to restart any remaining jobs which were running prior to the shutdown. • *warm* – All rerunnable jobs which were running when the server went down are requeued. All other jobs are maintained. New selections are made for which jobs are placed into execution. Warm is the default if `-t` is not specified. • *cold* – All jobs are deleted. Positive confirmation is required before this direction is accepted. • *create* – The server will discard any existing configuration files, queues and jobs, and initialize configuration files to the default values. The server is idled.

Files

File	Description
TORQUE_HOME/server_ priv	Default directory for configuration files, typically `/usr/spool/pbs/server_ priv`
TORQUE_HOME/server_ logs	Directory for log files recorded by the server

Signal handling

On receipt of the following signals, the server performs the defined action:

Action	Description
SIGHUP	The current server log and accounting log are closed and reopened. This allows for the prior log to be renamed and a new log started from the time of the signal.
SIGINT	Causes an orderly shutdown of pbs_server.
SIGUSR1, SIGURS2	Causes server to increase and decrease logging levels, respectively.
SIGTERM	Causes an orderly shutdown of pbs_server.
SIGSHUTDN	On systems (Unicos) where SIGSHUTDN is defined, it also causes an orderly shutdown of the server.
SIGPIPE	This signal is ignored.

All other signals have their default behavior installed.

Exit status

If the server command fails to begin batch operation, the server exits with a value greater than zero.

Related topics

- pbs_mom(8B)
- pbsnodes(8B)
- qmgr(1B)
- qrun(8B)
- qsub(1B)
- qterm(8B)

Non-Adaptive Computing topics

- pbs_connect(3B)
- pbs_sched_basl(8B)
- pbs_sched_tcl(8B)
- qdisable(8B)
- qenable(8B)
- qstart(8B)
- qstop(8B)
- PBS External Reference Specification

pbs_track

Starts a new process and informs pbs_mom to start tracking it.

Synopsis

```
pbs_track -j <JOBID> [-b] <executable> [args]
```

Description

The pbs_track command tells a pbs_mom daemon to monitor the lifecycle and resource usage of the process that it launches using exec(). The pbs_mom is told about this new process via the Task Manager API, using tm_adopt(). The process must also be associated with a job that already exists on the pbs_mom.

By default, pbs_track will send its PID to TORQUE via tm_adopt(). It will then perform an exec(), causing <executable> to run with the supplied arguments. pbs_track will not return until the launched process has completed because it becomes the launched process.

This command can be considered related to the **pbsdsh** command which uses the tm_spawn() API call. The pbsdsh command asks a pbs_mom to launch and track a new process on behalf of a job. When it is not desirable or possible for the pbs_mom to spawn processes for a job, pbs_track can be used to allow an external entity to launch a process and include it as part of a job.

This command improves integration with TORQUE and SGI's MPT MPI implementation.

Options

Option	Description
-j <JOBID>	Job ID the new process should be associated with.
-b	Instead of having pbs_track send its PID to TORQUE, it will fork() first, send the child PID to TORQUE, and then execute from the forked child. This essentially "backgrounds" pbs_track so that it will return after the new process is launched.

Operands

The pbs_track command accepts a path to a program/executable (<executable>) and, optionally, one or more arguments to pass to that program.

Exit status

Because the pbs_track command becomes a new process (if used without **-b**), its exit status will match that of the new process. If the **-b** option is used, the exit status will be zero if no errors occurred before launching the new process.

If pbs_track fails, whether due to a bad argument or other error, the exit status will be set to a non-zero value.

Related topics

- pbsdsh(1B)

Non-Adaptive Computing topics

- tm_spawn(3B)

pbsdsh

The pbsdsh command distributes tasks to nodes under pbs.

> Some limitations exist in the way that pbsdsh can be used. Please note the following situations are not currently supported:
>
> - Running multiple instances of pbsdsh concurrently within a single job.
> - Launching a large number of processes in succession (causes pbdsdsh to hang).

Synopsis

```
pbsdsh [-c copies] [-o] [-s] [-u] [-v] program [args]
pbsdsh [-n node] [-o] [-s] [-u] [-v] program [args]
pbsdsh [-h nodename] [-o] [-v] program [args]
```

Description

Executes (spawns) a normal Unix program on one or more nodes under control of the Portable Batch System, PBS. Pbsdsh uses the Task Manager API (see tm_spawn(3)) to distribute the program on the allocated nodes.

When run without the **-c** or the **-n** option, pbsdsh will spawn the program on all nodes allocated to the PBS job. The spawns take place concurrently – all execute at (about) the same time.

Users will find the PBS_TASKNUM, PBS_NODENUM, and the PBS_VNODENUM environmental variables useful. They contain the TM task id, the node identifier, and the cpu (virtual node) identifier.

> Note that under particularly high workloads, the pbsdsh command may not function properly.

Options

Option	Name	Description
-c	copies	The program is spawned on the first Copies nodes allocated. This option is mutually exclusive with **-n**.
-h	hostname	The program is spawned on the node specified.

Option	Name	Description
-n	node	The program is spawned on one node which is the n-th node allocated. This option is mutually exclusive with **-c**.
-o	---	Directs tasks stdout and stderr to the corresponding streams of pbsdsh. Otherwise, tasks stdout and/or stderr go to the job.
-s	---	If this option is given, the program is run in turn on each node, one after the other.
-u	---	The program is run once on each node (unique). This ignores the number of allocated processors on a given node.
-v	---	Verbose output about error conditions and task exit status is produced.

Operands

The first operand, program, is the program to execute.

Additional operands are passed as arguments to the program.

Standard error

The pbsdsh command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the command, the exit status will be a value of zero.

If the pbsdsh command fails to process any operand, or fails to contact the MOM daemon on the localhost the command exits with a value greater than zero.

Related topics

- qsub(1B)

Non-Adaptive Computing topics

- tm_spawn(3B)

pbsnodes

PBS node manipulation.

Synopsis

```
pbsnodes [-{a|x}] [-q] [-s server] [node|:property]
pbsnodes -l [-q] [-s server] [state] [nodename|:property ...]
pbsnodes -m <running|standby|suspend|hibernate|shutdown> <host list>
pbsnodes [-{c|d|o|r}] [-q] [-s server] [-n -l] [-N "note"] [node|:property]
```

Description

The pbsnodes command is used to mark nodes down, free or offline. It can also be used to list nodes and their state. Node information is obtained by sending a request to the PBS job server. Sets of nodes can be operated on at once by specifying a node property prefixed by a colon. (For more information, see Node states.)

Nodes do not exist in a single state, but actually have a set of states. For example, a node can be simultaneously "busy" and "offline". The "free" state is the absence of all other states and so is never combined with other states.

In order to execute pbsnodes with other than the **-a** or **-l** options, the user must have PBS Manager or Operator privilege.

Options

Option	Description
-a	All attributes of a node or all nodes are listed. This is the default if no flag is given.
-x	Same as **-a**, but the output has an XML-like format.
-c	Clear OFFLINE from listed nodes.
-d	Print MOM diagnosis on the listed nodes. Not yet implemented. Use **momctl** instead.

Option	Description
-m	Set the hosts in the specified host list to the requested power state. If a compute node does not support the energy-saving power state you request, the command returns an error and leaves the state unchanged. In order for the command to wake a node from a low-power state, Wake-on-LAN (WOL) must be enabled for the node. In order for the command to wake a node from a low-power state, Wake-on-LAN must be enabled for the node and it must support the g WOL packet. For more information, see **Changing Node Power States** on page 2270. The allowable power states are: • **Running**: The node is up and running. • **Standby**: CPU is halted but still powered. Moderate power savings but low latency entering and leaving this state. • **Suspend**: Also known as Suspend-to-RAM. Machine state is saved to RAM. RAM is put into self-refresh mode. Much more significant power savings with longer latency entering and leaving state. • **Hibernate**: Also known as Suspend-to-disk. Machine state is saved to disk and then powered down. Significant power savings but very long latency entering and leaving state. • **Shutdown**: Equivalent to shutdown now command as root. The host list is a space-delimited list of node host names.

Option	Description

```
pbsnodes -m shutdown node01 node02 node03 node04
```

With this command, pbs_server tells the pbs_mom associated with nodes01-04 to shut down the node.

The pbsnodes output shows the current power state of nodes. In this example, note that pbsnodes returns the MAC addresses of the nodes.

```
pbsnodes
nuc1
     state = free
     power_state = Running
     np = 4
     ntype = cluster
     status = rectime=1395765676,macaddr=0b:25:22:92:7b:26
,cpuclock=Fixed,varattr=,jobs=,state=free,netload=1242652020,gres=,loadave=0.16,ncpus=6,physmem
=16435852kb,availmem=24709056kb,totmem=33211016kb,idletime=4636,nusers=3,nsessions=12,sessions=
2758 998 1469 2708 2797 2845 2881 2946 4087 4154 4373 6385,uname=Linux bdaw 3.2.0-60-generic
#91-Ubuntu SMP Wed Feb 19 03:54:44 UTC 2014 x86_64,opsys=linux
     note = This is a node note
     mom_service_port = 15002
     mom_manager_port = 15003

nuc2
     state = free
     power_state = Running
     np = 4
     ntype = cluster
     status = rectime=1395765678,macaddr=2c:a8:6b:f4:b9:35
,cpuclock=OnDemand:800MHz,varattr=,jobs=,state=free,netload=12082362,gres=,loadave=0.00,ncpus=4
,physmem=16300576kb,availmem=17561808kb,totmem=17861144kb,idletime=67538,nusers=2,nsessions=7,s
essions=2189 2193 2194 2220 2222 2248 2351,uname=Linux nuc2 2.6.32-431.el6.x86_64 #1 SMP Fri
Nov 22 03:15:09 UTC 2013 x86_64,opsys=linux
     mom_service_port = 15002
     mom_manager_port = 15003
```

Option	Description
-	Add the OFFLINE state. This is different from being marked DOWN. OFFLINE prevents new jobs from running on the specified nodes. This gives the administrator a tool to hold a node out of service without changing anything else. The OFFLINE state will never be set or cleared automatically by pbs_server; it is purely for the manager or operator.
-	Purge the node record from pbs_server. Not yet implemented.
-	Reset the listed nodes by clearing OFFLINE and adding DOWN state. pbs_server will ping the node and, if they communicate correctly, free the node.

Option	Description
-l	List node names and their state. If no state is specified, only nodes in the DOWN, OFFLINE, or UNKNOWN states are listed. Specifying a state string acts as an output filter. Valid state strings are "active", "all", "busy", "down", "free", "job-exclusive", "job-sharing", "offline", "reserve", "state-unknown", "time-shared", and "up". • Using *all* displays all nodes and their attributes. • Using *active* displays all nodes which are job-exclusive, job-sharing, or busy. • Using *up* displays all nodes in an "up state". Up states include job-exclusive, job-sharing, reserve, free, busy and time-shared. • All other strings display the nodes which are currently in the state indicated by the string.
-N	Specify a "note" attribute. This allows an administrator to add an arbitrary annotation to the listed nodes. To clear a note, use `-N ""` or `-N n`.
-n	Show the "note" attribute for nodes that are DOWN, OFFLINE, or UNKNOWN. This option requires **-l**.
-q	Suppress all error messages.
-s	Specify the PBS server's hostname or IP address.

Related topics

- pbs_server(8B)

Non-Adaptive Computing topics

- PBS External Reference Specification

qalter

Alter batch job.

Synopsis

```
qalter [-a date_time][-A account_string][-c interval][-e path_name]
 [-h hold_list][-j join_list][-k keep_list][-l resource_list]
 [-m mail_options][-M mail_list][-n][-N name][-o path_name]
 [-p priority][-r y|n][-S path_name_list][-u user_list]
 [-v variable_list][-W additional_attributes]
```

```
[-t array_range]
job_identifier ...
```

Description

The qalter command modifies the attributes of the job or jobs specified by `job_identifier` on the command line. Only those attributes listed as options on the command will be modified. If any of the specified attributes cannot be modified for a job for any reason, none of that job's attributes will be modified.

The qalter command accomplishes the modifications by sending a Modify Job batch request to the batch server which owns each job.

Options

Option	Name	Description
-a	date_time	Replaces the time at which the job becomes eligible for execution. The date_time argument syntax is: `[[[[CC]YY]MM]DD]hhmm[.SS]` If the month, MM, is not specified, it will default to the current month if the specified day DD, is in the future. Otherwise, the month will be set to next month. Likewise, if the day, DD, is not specified, it will default to today if the time hhmm is in the future. Otherwise, the day will be set to tomorrow. This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun.
-A	account_string	Replaces the account string associated with the job. This attribute cannot be altered once the job has begun execution.

Option	Name	Description
-c	checkpoint_ interval	Replaces the interval at which the job will be checkpointed. If the job executes upon a host which does not support checkpointing, this option will be ignored. The interval argument is specified as: • *n* – No checkpointing is to be performed. • *s* – Checkpointing is to be performed only when the server executing the job is shutdown. • *c* – Checkpointing is to be performed at the default minimum cpu time for the queue from which the job is executing. • *c=minutes* – Checkpointing is performed at intervals of the specified amount of time in minutes. Minutes are the number of minutes of CPU time used, not necessarily clock time. This value must be greater than zero. If the number is less than the default checkpoint time, the default time will be used. This attribute can be altered once the job has begun execution, but the new value does not take effect unless the job is rerun.
-e	path_name	Replaces the path to be used for the standard error stream of the batch job. The path argument is of the form: `[hostname:]path_name` where *hostname* is the name of a host to which the file will be returned and *path_name* is the path name on that host in the syntax recognized by POSIX 1003.1. The argument will be interpreted as follows: • *path_name* – Where *path_name* is not an absolute path name, then the qalter command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component. • *hostname:path_name* – Where *path_name* is not an absolute path name, then the qalter command will not expand the path name. The execution server will expand it relative to the home directory of the user on the system specified by hostname. This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun.

Option	Name	Description
-h	hold_list	Updates the types of holds on the job. The hold_list argument is a string of one or more of the following characters: • *u* – Add the USER type hold. • *s* – Add the SYSTEM type hold if the user has the appropriate level of privilege. (Typically reserved to the batch administrator.) • *o* – Add the OTHER (or OPERATOR) type hold if the user has the appropriate level of privilege. (Typically reserved to the batch administrator and batch operator.) • *n* – Set to none and clear the hold types which could be applied with the user's level of privilege. Repetition of characters is permitted, but "n" may not appear in the same option argument with the other three characters. This attribute can be altered once the job has begun execution, but the hold will not take effect unless the job is rerun.
-j	join	Declares which standard streams of the job will be merged together. The join argument value may be the characters "oe" and "eo", or the single character "n". An argument value of oe directs that the standard output and standard error streams of the job will be merged, intermixed, and returned as the standard output. An argument value of eo directs that the standard output and standard error streams of the job will be merged, intermixed, and returned as the standard error. A value of n directs that the two streams will be two separate files. This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun. If using either the -e or the -o option and the `-j eo|oe` option, the `-j` option takes precedence and all standard error and output messages go to the chosen output file.

Option	Name	Description
-k	keep	Defines which if either of standard output or standard error of the job will be retained on the execution host. If set for a stream, this option overrides the path name for that stream. The argument is either the single letter "e", "o", or "n", or one or more of the letters "e" and "o" combined in either order. • *n* – No streams are to be retained. • *e* – The standard error stream is to retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by: `job_name.esequence` where job_name is the name specified for the job, and sequence is the sequence number component of the job identifier. • *o* – The standard output stream is to be retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by: `job_name.osequence` where job_name is the name specified for the job, and sequence is the sequence number component of the job identifier. • *eo* – Both the standard output and standard error streams will be retained. • *oe* – Both the standard output and standard error streams will be retained. This attribute cannot be altered once the job has begun execution.
-l	resource_ list	Modifies the list of resources that are required by the job. The resource_list argument is in the following syntax: `resource_name[=[value]][,resource_name[=[value]],...]` For the complete list of resources that can be modified, see **Requesting Resources** on page 2237. If a requested modification to a resource would exceed the resource limits for jobs in the current queue, the server will reject the request. If the job is running, only certain resources can be altered. Which resources can be altered in the run state is system dependent. A user may only lower the limit for those resources.
-m	mail_ options	Replaces the set of conditions under which the execution server will send a mail message about the job. The mail_options argument is a string which consists of the single character "n", or one or more of the characters "a", "b", and "e". If the character "n" is specified, no mail will be sent. For the letters "a", "b", and "e": • *a* – Mail is sent when the job is aborted by the batch system. • *b* – Mail is sent when the job begins execution. • *e* – Mail is sent when the job ends.

<table>
<tr><th>Option</th><th>Name</th><th>Description</th></tr>
<tr><td>-M</td><td>user_list</td><td>Replaces the list of users to whom mail is sent by the execution server when it sends mail about the job.
The user_list argument is of the form:
<code>user[@host][,user[@host],...]</code></td></tr>
<tr><td>-n</td><td>node-exclusive</td><td>Sets or unsets exclusive node allocation on a job. Use the y and n options to enable or disable the feature. This affects only cpusets and compatible schedulers.
<code>> qalter ... -n y #enables exclusive node allocation on a job</code>
<code>> qalter ... -n n #disables exclusive node allocation on a job</code></td></tr>
<tr><td>-N</td><td>name</td><td>Renames the job. The name specified may be up to and including 15 characters in length. It must consist of printable, nonwhite space characters with the first character alphabetic.</td></tr>
<tr><td>-o</td><td>path</td><td>Replaces the path to be used for the standard output stream of the batch job. The path argument is of the form:
<code>[hostname:]path_name</code>
where hostname is the name of a host to which the file will be returned and path_name is the path name on that host in the syntax recognized by POSIX. The argument will be interpreted as follows:
path_name – Where path_name is not an absolute path name, then the qalter command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component.
hostname:path_name – Where path_name is not an absolute path name, then the qalter command will not expand the path name. The execution server will expand it relative to the home directory of the user on the system specified by hostname.
This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun.</td></tr>
<tr><td>-p</td><td>priority</td><td>Replaces the priority of the job. The priority argument must be an integer between -1024 and +1023 inclusive.
This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun.</td></tr>
<tr><td>-r</td><td>[y/n]</td><td>Declares whether the job is rerunable (see the qrerun command). The option argument c is a single character. PBS recognizes the following characters: y and n. If the argument is "y", the job is marked rerunable.
If the argument is "n", the job is marked as not rerunable.</td></tr>
</table>

Option	Name	Description
-S	path	Declares the shell that interprets the job script. The option argument path_list is in the form: `path[@host][,path[@host],...]` Only one path may be specified for any host named. Only one path may be specified without the corresponding host name. The path selected will be the one with the host name that matched the name of the execution host. If no matching host is found, then the path specified (without a host) will be selected. If the `-S` option is not specified, the option argument is the null string, or no entry from the path_list is selected, the execution will use the login shell of the user on the execution host. This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun.
-t	array_ range	The array_range argument is an integer id or a range of integers. Multiple ids or id ranges can be combined in a comma delimited list. Examples: `-t 1-100` or `-t 1,10,50-100` If an array range isn't specified, the command tries to operate on the entire array. The command acts on the array (or specified range of the array) just as it would on an individual job. An optional "slot limit" can be specified to limit the amount of jobs that can run concurrently in the job array. The default value is unlimited. The slot limit must be the last thing specified in the array_request and is delimited from the array by a percent sign (%). `qalter weatherSimulationArray[] -t %20` Here, the array weatherSimulationArray[] is configured to allow a maximum of 20 concurrently running jobs. Slot limits can be applied at job submit time with **qsub**, or can be set in a global server parameter policy with **max_slot_limit**.
-u	user_list	Replaces the user name under which the job is to run on the execution system. The user_list argument is of the form: `user[@host][,user[@host],...]` Only one user name may be given for per specified host. Only one of the user specifications may be supplied without the corresponding host specification. That user name will be used for execution on any host not named in the argument list. This attribute cannot be altered once the job has begun execution.
-W	additional_ attributes	The `-W` option allows for the modification of additional job attributes. Note if white space occurs anywhere within the option argument string or the equal sign, "=", occurs within an attribute_value string, then the string must be enclosed with either single or double quote marks. To see the attributes PBS currently supports within the `-W` option, see Table 4-4: -W additional_attributes on page 2367.

Table 4-4: -W additional_attributes

<table>
<tr><th>Attribute</th><th>Description</th></tr>
<tr><td>depend=dependency_list</td><td>
Redefines the dependencies between this and other jobs. The dependency_list is in the form:
<code>type[:argument[:argument...][,type:argument...]</code>
The argument is either a numeric count or a PBS job id according to type. If argument is a count, it must be greater than 0. If it is a job id and is not fully specified in the form: <code>seq_number.server.name</code>, it will be expanded according to the default server rules. If argument is null (the preceding colon need not be specified), the dependency of the corresponding type is cleared (unset).

synccount:count – This job is the first in a set of jobs to be executed at the same time. Count is the number of additional jobs in the set.
syncwith:jobid – This job is an additional member of a set of jobs to be executed at the same time. In the above and following dependency types, jobid is the job identifier of the first job in the set.
after:jobid [:jobid...] – This job may be scheduled for execution at any point after jobs jobid have started execution.
afterok:jobid [:jobid...] – This job may be scheduled for execution only after jobs jobid have terminated with no errors. See the csh warning under "Extended Description".
afternotok:jobid [:jobid...] – This job may be scheduled for execution only after jobs jobid have terminated with errors. See the csh warning under "Extended Description".
afterany:jobid [:jobid...] – This job may be scheduled for execution after jobs jobid have terminated, with or without errors.
on:count – This job may be scheduled for execution after count dependencies on other jobs have been satisfied. This dependency is used in conjunction with any of the 'before' dependencies shown below. If job A has on:2, it will wait for two jobs with 'before' dependencies on job A to be fulfilled before running.
before:jobid [:jobid...] – When this job has begun execution, then jobs jobid... may begin.
beforeok:jobid [:jobid...] – If this job terminates execution without errors, then jobs jobid... may begin. See the csh warning under "Extended Description".
beforenotok:jobid [:jobid...] – If this job terminates execution with errors, then jobs jobid... may begin. See the csh warning under "Extended Description".
beforeany:jobid [:jobid...] – When this job terminates execution, jobs jobid... may begin.
If any of the before forms are used, the job referenced by jobid must have been submitted with a dependency type of on.
If any of the before forms are used, the jobs referenced by jobid must have the same owner as the job being altered. Otherwise, the dependency will not take effect.

Error processing of the existence, state, or condition of the job specified to qalter is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the job will be deleted by the server. Mail will be sent to the job submitter stating the error.
</td></tr>
</table>

Attribute	Description
group_list=g_list	Alters the group name under which the job is to run on the execution system. The g_list argument is of the form: `group[@host][,group[@host],...]` Only one group name may be given per specified host. Only one of the group specifications may be supplied without the corresponding host specification. That group name will used for execution on any host not named in the argument list.
stagein=file_list **stageout=file_list**	Alters which files are staged (copied) in before job start or staged out after the job completes execution. The file_list is in the form: `local_file@hostname:remote_file[,...]` The name local_file is the name on the system where the job executes. It may be an absolute path or a path relative to the home directory of the user. The name remote_file is the destination name on the host specified by hostname. The name may be absolute or relative to the user's home directory on the destination host.

Operands

The qalter command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Standard error

Any error condition, either in processing the options or the operands, or any error received in reply to the batch requests will result in an error message being written to standard error.

Exit status

Upon successful processing of all the operands presented to the qalter command, the exit status will be a value of zero.

If the qalter command fails to process any operand, the command exits with a value greater than zero.

Copyright

Related topics

- qdel
- qhold
- qrls
- qsub

Non-Adaptive Computing topics

- Batch Environment Services
- qmove
- touch

qchkpt

Checkpoint pbs batch jobs.

Synopsis

```
qchkpt <JOBID>[ <JOBID>] ...
```

Description

The qchkpt command requests that the PBS MOM generate a checkpoint file for a running job.

This is an extension to POSIX.2d.

The qchkpt command sends a Chkpt Job batch request to the server as described in the general section.

Options

None.

Operands

The qchkpt command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Examples

```
> qchkpt 3233 request a checkpoint for job 3233
```

Standard error

The qchkpt command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qchkpt command, the exit status will be a value of zero.

If the qchkpt command fails to process any operand, the command exits with a value greater than zero.

Related topics

- qhold(1B)
- qrls(1B)
- qalter(1B)
- qsub(1B)

Non-Adaptive Computing topics

- pbs_alterjob(3B)
- pbs_holdjob(3B),
- pbs_rlsjob(3B)
- pbs_job_attributes(7B)
- pbs_resources_unicos8(7B)

qdel

(delete job)

Synopsis

```
qdel [{-a <asynchronous delete>|-m <message>|-p|-W <delay>|-t <array_range>}]
<JOBID>[ <JOBID>]... | 'all' | 'ALL'
```

Description

The qdel command deletes jobs in the order in which their job identifiers are presented to the command. A job is deleted by sending a Delete Job batch request to the batch server that owns the job. A job that has been deleted is no longer subject to management by batch services.

A batch job may be deleted by its owner, the batch operator, or the batch administrator.

A batch job being deleted by a server will be sent a SIGTERM signal following by a SIGKILL signal. The time delay between the two signals is an attribute of the execution queue from which the job was run (set table by the administrator). This delay may be overridden by the **-W** option.

See the PBS ERS section 3.1.3.3, "Delete Job Request", for more information.

Options

Option	Name	Description
-a	asynchronous delete	Performs an asynchronous delete. The server responds to the user before contacting the MOM. The option `qdel -a all` performs `qdel all` due to restrictions from being single-threaded.
-W	delay	Specifies the wait delay between the sending of the SIGTERM and SIGKILL signals. The argument is the length of time in seconds of the delay.
-p	purge	Forcibly purges the job from the server. This should only be used if a running job will not exit because its allocated nodes are unreachable. The admin should make every attempt at resolving the problem on the nodes. If a job's mother superior recovers after purging the job, any epilogue scripts may still run. This option is only available to a batch operator or the batch administrator.
-m	message	Specify a comment to be included in the email. The argument message specifies the comment to send. This option is only available to a batch operator or the batch administrator.
-t	array_range	The array_range argument is an integer id or a range of integers. Multiple ids or id ranges can be combined in a comma delimited list (examples: -t 1-100 or -t 1,10,50-100). If an array range isn't specified, the command tries to operate on the entire array. The command acts on the array (or specified range of the array) just as it would on an individual job.

Operands

The qdel command accepts one or more job_identifier operands of the form:

`sequence_number[.server_name][@server]`

or

`all`

Examples

```
> qdel 1324
> qdel 1324-3 To delete one job of a job array
> qdel all To delete all jobs (Version 2.3.0 and later)
```

Standard error

The qdel command will write a diagnostic messages to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qdel command, the exit status will be a value of zero.

If the qdel command fails to process any operand, the command exits with a value greater than zero.

Related topics

- qsub(1B)
- qsig(1B)

Non-Adaptive Computing topics

- pbs_deljob(3B)

qgpumode

(GPU mode)

Synopsis

```
qgpumode -H host -g gpuid -m mode
```

Description

The qgpumode command specifies the mode for the GPU. This command triggers an immediate update of the pbs_server.

For additional information about options for configuring GPUs, see **NVIDIA GPUs** on page 796.

Options

Option	Description
-H	Specifies the host where the GPU is located.
-g	Specifies the ID of the GPU. This varies depending on the version of the Nvidia driver used. For driver 260.x, it is 0, 1, and so on. For driver 270.x, it is the PCI bus address, i.e., 0:5:0.

Option	Description
-m	Specifies the new mode for the GPU: • **0 (Default/Shared)**: Default/shared compute mode. Multiple threads can use `cudaSetDevice()` with this device. • **1 (Exclusive Thread)**: Compute-exclusive-thread mode. Only one thread in one process is able to use `cudaSetDevice()` with this device. • **2 (Prohibited)**: Compute-prohibited mode. No threads can use `cudaSetDevice()` with this device. • **3 (Exclusive Process)**: Compute-exclusive-process mode. Many threads in one process are able to use `cudaSetDevice()` with this device. `qgpumode -H node01 -g 0 -m 1` *This puts the first GPU on node01 into mode 1 (exclusive)* `qgpumode -H node01 -g 0 -m 0` *This puts the first GPU on node01 into mode 0 (shared)*

Related topics

- **qgpureset** on page 2374

qgpureset

(reset GPU)

Synopsis

```
qgpureset -H host -g gpuid -p -v
```

Description

The qgpureset command resets the GPU.

Options

Option	Description
-H	Specifies the host where the GPU is located.
-g	Specifies the ID of the GPU. This varies depending on the version of the Nvidia driver used. For driver 260.x, it is 0, 1, and so on. For driver 270.x, it is the PCI bus address, i.e., 0:5:0.

Option	Description
-p	Specifies to reset the GPU's permanent ECC error count.
-v	Specifies to reset the GPU's volatile ECC error count.

Related topics

- **qgpumode** on page 2373

qhold

(hold job)

Synopsis

```
qhold [{-h <HOLD LIST>|-t <array_range>}] <JOBID>[ <JOBID>] ...
```

Description

The qhold command requests that the server place one or more holds on a job. A job that has a hold is not eligible for execution. There are three supported holds: USER, OTHER (also known as operator), and SYSTEM.

A user may place a USER hold upon any job the user owns. An "operator", who is a user with "operator privilege," may place ether an USER or an OTHER hold on any job. The batch administrator may place any hold on any job.

If no **-h** option is given, the USER hold will be applied to the jobs described by the job_identifier operand list.

If the job identified by job_identifier is in the queued, held, or waiting states, then the hold type is added to the job. The job is then placed into held state if it resides in an execution queue.

If the job is in running state, then the following additional action is taken to interrupt the execution of the job. If checkpoint/restart is supported by the host system, requesting a hold on a running job will (1) cause the job to be checkpointed, (2) the resources assigned to the job will be released, and (3) the job is placed in the held state in the execution queue.

If checkpoint/restart is not supported, qhold will only set the requested hold attribute. This will have no effect unless the job is rerun with the **qrerun** command.

Options

Option	Name	Description
-h	hold_list	The hold_list argument is a string consisting of one or more of the letters "u", "o", or "s" in any combination. The hold type associated with each letter is: • *u* – USER • *o* – OTHER • *s* – SYSTEM
-t	array_range	The array_range argument is an integer id or a range of integers. Multiple ids or id ranges can be combined in a comma delimited list (examples: -t 1-100 or -t 1,10,50-100) . If an array range isn't specified, the command tries to operate on the entire array. The command acts on the array (or specified range of the array) just as it would on an individual job.

Operands

The qhold command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Example

```
> qhold -h u 3233 place user hold on job 3233
```

Standard error

The qhold command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qhold command, the exit status will be a value of zero.

If the qhold command fails to process any operand, the command exits with a value greater than zero.

Related topics

- qrls(1B)
- qalter(1B)
- qsub(1B)

Non-Adaptive Computing topics

- pbs_alterjob(3B)
- pbs_holdjob(3B)

- pbs_rlsjob(3B)
- pbs_job_attributes(7B)
- pbs_resources_unicos8(7B)

qmgr

(PBS Queue Manager) PBS batch system manager.

Synopsis

```
qmgr [-a] [-c command] [-e] [-n] [-z] [server...]
```

Description

The qmgr command provides an administrator interface to query and configure batch system parameters (see **Appendix B: Server Parameters** on page 2417).

The command reads directives from standard input. The syntax of each directive is checked and the appropriate request is sent to the batch server or servers.

The list or print subcommands of qmgr can be executed by general users. Creating or deleting a queue requires PBS Manager privilege. Setting or unsetting server or queue attributes requires PBS Operator or Manager privilege.

> By default, the user root is the only PBS Operator and Manager. To allow other users to be privileged, the server attributes operators and managers will need to be set (i.e., as root, issue '`qmgr -c 'set server managers += <USER1>@<HOST>`'). See **TORQUE/PBS Integration Guide - RM Access Control** on page 1209.

If qmgr is invoked without the **-c** option and standard output is connected to a terminal, qmgr will write a prompt to standard output and read a directive from standard input.

Commands can be abbreviated to their minimum unambiguous form. A command is terminated by a new line character or a semicolon, ";", character. Multiple commands may be entered on a single line. A command may extend across lines by escaping the new line character with a back-slash "\".

Comments begin with the "#" character and continue to end of the line. Comments and blank lines are ignored by qmgr.

Options

Option	Name	Description
-a	---	Abort qmgr on any syntax errors or any requests rejected by a server.
-c	command	Execute a single command and exit qmgr.

Option	Name	Description
-e	---	Echo all commands to standard output.
-n	---	No commands are executed, syntax checking only is performed.
-z	---	No errors are written to standard error.

Operands

The *server* operands identify the name of the batch server to which the administrator requests are sent. Each *server* conforms to the following syntax:

```
host_name[:port]
```

where *host_name* is the network name of the host on which the server is running and *port* is the port number to which to connect. If *port* is not specified, the default port number is used.

If *server* is not specified, the administrator requests are sent to the local server.

Standard input

The qmgr command reads standard input for directives until end of file is reached, or the exit or quit directive is read.

Standard output

If Standard Output is connected to a terminal, a command prompt will be written to standard output when qmgr is ready to read a directive.

If the **-e** option is specified, qmgr will echo the directives read from standard input to standard output.

Standard error

If the **-z** option is not specified, the qmgr command will write a diagnostic message to standard error for each error occurrence.

Directive syntax

A qmgr directive is one of the following forms:

```
command server [names] [attr OP value[,attr OP value,...]]
command queue [names] [attr OP value[,attr OP value,...]]
command node [names] [attr OP value[,attr OP value,...]]
```

where *command* is the command to perform on an object.

Commands are:

Command	Description
active	Sets the active objects. If the active objects are specified, and the name is not given in a qmgr cmd the active object names will be used.
create	Is to create a new object, applies to queues and nodes.
delete	Is to destroy an existing object, applies to queues and nodes.
set	Is to define or alter attribute values of the object.
unset	Is to clear the value of attributes of the object. This form does not accept an OP and value, only the attribute name.
list	Is to list the current attributes and associated values of the object.
print	Is to print all the queue and server attributes in a format that will be usable as input to the qmgr command.
names	Is a list of one or more names of specific objects The name list is in the form: `[name][@server][,queue_name[@server]...]` with no intervening white space. The name of an object is declared when the object is first created. If the name is @server, then all the objects of specified type at the server will be affected.
attr	Specifies the name of an attribute of the object which is to be set or modified. If the attribute is one which consist of a set of resources, then the attribute is specified in the form: `attribute_name.resource_name`
OP	Operation to be performed with the attribute and its value: • "=" – set the value of the attribute. If the attribute has an existing value, the current value is replaced with the new value. • "+=" – increase the current value of the attribute by the amount in the new value. • "-=" – decrease the current value of the attribute by the amount in the new value.
value	The value to assign to an attribute. If the value includes white space, commas or other special characters, such as the "#" character, the value string must be enclosed in quote marks (").

The following are examples of qmgr directives:

```
create queue fast priority=10,queue_type=e,enabled = true,max_running=0
set queue fast max_running +=2
create queue little
set queue little resources_max.mem=8mw,resources_max.cput=10
unset queue fast max_running
set node state = "down,offline"
active server s1,s2,s3
list queue @server1
set queue max_running = 10       - uses active queues
```

Exit status

Upon successful processing of all the operands presented to the qmgr command, the exit status will be a value of zero.

If the qmgr command fails to process any operand, the command exits with a value greater than zero.

Related topics

- pbs_server(8B)

Non-Adaptive Computing topics

- pbs_queue_attributes (7B)
- pbs_server_attributes (7B)
- qstart (8B), qstop (8B)
- qenable (8B), qdisable (8)
- PBS External Reference Specification

qmove

Move PBS batch jobs.

Synopsis

```
qmove destination jobId [jobId ...]
```

Description

To move a job is to remove the job from the queue in which it resides and instantiate the job in another queue. The qmove command issues a Move Job batch request to the batch server that currently owns each job specified by *jobId*.

A job in the **Running**, **Transiting**, or **Exiting** state cannot be moved.

Operands

The first operand, the new *destination*, is one of the following:

`queue`

`@server`

```
queue@server
```

If the *destination* operand describes only a queue, then qmove will move jobs into the queue of the specified name at the job's current server. If the *destination* operand describes only a batch server, then qmove will move jobs into the default queue at that batch server. If the *destination* operand describes both a queue and a batch server, then qmove will move the jobs into the specified queue at the specified server.

All following operands are *jobId*s which specify the jobs to be moved to the new *destination*. The qmove command accepts one or more *jobId* operands of the form: `sequenceNumber[.serverName][@server]`

Standard error

The qmove command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qmove command, the exit status will be a value of zero.

If the qmove command fails to process any operand, the command exits with a value greater than zero.

Related topics

- **qsub** on page 2395

Related topics(non-Adaptive Computing topics)

- pbs_movejob(3B)

qorder

Exchange order of two PBS batch jobs in any queue.

Synopsis

```
qorder job1_identifier job2_identifier
```

Description

To order two jobs is to exchange the jobs' positions in the queue(s) in which the jobs reside. The two jobs must be located on the same server. No attribute of the job, such as priority, is changed. The impact of changing the order in the queue(s) is dependent on local job schedule policy. For information about your local job schedule policy, contact your systems administrator.

> A job in the **running** state cannot be reordered.

Operands

Both operands are `job_identifiers` that specify the jobs to be exchanged. The qorder command accepts two `job_identifier` operands of the following form:
`sequence_number[.server_name][@server]`

The two jobs must be in the same location, so the server specification for the two jobs must agree.

Standard error

The qorder command will write diagnostic messages to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qorder command, the exit status will be a value of zero.

If the qorder command fails to process any operand, the command exits with a value greater than zero.

Related topics

- **qsub** on page 2395
- **qmove** on page 2380

Related topics(non-Adaptive Computing topics)

- pbs_orderjob(3B)
- pbs_movejob(3B)

qrerun

(Rerun a batch job)

Synopsis

```
qrerun [{-f}] <JOBID>[ <JOBID>] ...
```

Description

The qrerun command directs that the specified jobs are to be rerun if possible. To rerun a job is to terminate the session leader of the job and return the job to the queued state in the execution queue in which the job currently resides.

If a job is marked as not rerunable then the rerun request will fail for that job. If the mini-server running the job is down, or it rejects the request, the Rerun Job batch request will return a failure unless **-f** is used.

Using **-f** violates IEEE Batch Processing Services Standard and should be handled with great care. It should only be used under exceptional circumstances. The best practice is to fix the problem mini-server host and let qrerun run normally. The nodes may need manual cleaning (see the `-r` option on the **qsub** and **qalter** commands).

Options

Option	Description
-f	Force a rerun on a job

```
qrerun -f 15406
```

The qrerun all command is meant to be run if all of the compute nodes go down. If the machines have actually crashed, then we know that all of the jobs need to be restarted. The behavior if you don't run this would depend on how you bring up the pbs_mom daemons, but by default would be to cancel all of the jobs.

Running the command makes it so that all jobs are requeued without attempting to contact the moms on which they should be running.

Operands

The qrerun command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Standard error

The qrerun command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qrerun command, the exit status will be a value of zero.

If the qrerun command fails to process any operand, the command exits with a value greater than zero.

Examples

```
> qrerun 3233
```

(Job 3233 will be re-run.)

Related topics

- qsub(1B)
- qalter(1B)

Non-Adaptive Computing topics

- pbs_alterjob(3B)
- pbs_rerunjob(3B)

qrls

(Release hold on PBS batch jobs)

Synopsis

```
qrls [{-h <HOLD LIST>|-t <array_range>}] <JOBID>[ <JOBID>] ...
```

Description

The qrls command removes or releases holds which exist on batch jobs.

A job may have one or more types of holds which make the job ineligible for execution. The types of holds are USER, OTHER, and SYSTEM. The different types of holds may require that the user issuing the qrls command have special privileges. A user may always remove a USER hold on their own jobs, but only privileged users can remove OTHER or SYSTEM holds. An attempt to release a hold for which the user does not have the correct privilege is an error and no holds will be released for that job.

If no **-h** option is specified, the USER hold will be released.

If the job has no execution_time pending, the job will change to the queued state. If an execution_time is still pending, the job will change to the waiting state.

Options

Command	Name	Description
-h	hold_ list	Defines the types of hold to be released from the jobs. The hold_list option argument is a string consisting of one or more of the letters "u", "o", and "s" in any combination. The hold type associated with each letter is: • *u* – USER • *o* – OTHER • *s* – SYSTEM
-t	array_ range	The array_range argument is an integer id or a range of integers. Multiple ids or id ranges can be combined in a comma delimited list. Examples: -t 1-100 or -t 1,10,50-100 If an array range isn't specified, the command tries to operate on the entire array. The command acts on the array (or specified range of the array) just as it would on an individual job.

Operands

The qrls command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Examples

```
> qrls -h u 3233 release user hold on job 3233
```

Standard error

The qrls command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qrls command, the exit status will be a value of zero.

If the qrls command fails to process any operand, the command exits with a value greater than zero.

Related topics

Related topics

- qsub(1B)
- qalter(1B)
- qhold(1B)

Non-Adaptive Computing topics)

- pbs_alterjob(3B)
- pbs_holdjob(3B)
- pbs_rlsjob(3B)

qrun

(Run a batch job)

Synopsis

```
qrun [{-H <HOST>|-a}] <JOBID>[ <JOBID>] ...
```

Overview

The qrun command runs a job.

Format

-H	
Format	<STRING> Host Identifier
Default	---
Description	Specifies the host within the cluster on which the job(s) are to be run. The host argument is the name of a host that is a member of the cluster of hosts managed by the server. If the option is not specified, the server will select the "worst possible" host on which to execute the job.
Example	`qrun -H hostname 15406`

-a	
Format	---
Default	---
Description	Run the job(s) asynchronously.
Example	`qrun -a 15406`

Command details

The qrun command is used to force a batch server to initiate the execution of a batch job. The job is run regardless of scheduling position or resource requirements.

In order to execute qrun, the user must have PBS Operation or Manager privileges.

Examples

```
> qrun 3233
```

(Run job 3233.)

qsig

(Signal a job)

Synopsis

```
qsig [{-s <SIGNAL>}] <JOBID>[ <JOBID>] ...
 [-a]
```

Description

The qsig command requests that a signal be sent to executing batch jobs. The signal is sent to the session leader of the job. If the -s option is not specified, SIGTERM is sent. The request to signal a batch job will be rejected if:

- The user is not authorized to signal the job.
- The job is not in the running state.
- The requested signal is not supported by the system upon which the job is executing.

The qsig command sends a Signal Job batch request to the server which owns the job.

Options

Option	Name	Description
-s	signal	Declares which signal is sent to the job. The signal argument is either a signal name, e.g. SIGKILL, the signal name without the SIG prefix, e.g. KILL, or an unsigned signal number, e.g. 9. The signal name SIGNULL is allowed; the server will send the signal 0 to the job which will have no effect on the job, but will cause an obituary to be sent if the job is no longer executing. Not all signal names will be recognized by qsig. If it doesn't recognize the signal name, try issuing the signal number instead. Two special signal names, "suspend" and "resume", are used to suspend and resume jobs. Cray systems use the Cray-specific suspend()/resume() calls. On non-Cray system, suspend causes a SIGTSTP to be sent to all processes in the job's top task, wait 5 seconds, and then send a SIGSTOP to all processes in all tasks on all nodes in the job. This differs from TORQUE 2.0.0 which did not have the ability to propagate signals to sister nodes. Resume sends a SIGCONT to all processes in all tasks on all nodes. When suspended, a job continues to occupy system resources but is not executing and is not charged for walltime. The job will be listed in the "S" state. Manager or operator privilege is required to suspend or resume a job. Interactive jobs may not resume properly because the top-level shell will background the suspended child process.
-a	asynchronously	Makes the command run asynchronously.

Operands

The qsig command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Examples

```
> qsig -s SIGKILL 3233    send a SIGKILL to job 3233
> qsig -s KILL 3233       send a SIGKILL to job 3233
> qsig -s 9 3233          send a SIGKILL to job 3233
```

Standard error

The qsig command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qsig command, the exit status will be a value of zero.

If the qsig command fails to process any operand, the command exits with a value greater than zero.

Related topics

- qsub(1B)

Non-Adaptive Computing topics

- pbs_sigjob(3B)
- pbs_resources_*(7B) where * is system type
- PBS ERS

qstat

Show status of PBS batch jobs.

Synopsis

qstat [-c on page 2389] [-f [-1]][-W site_specific] [job_identifier... | destination...] [time]
qstat [-a|-i|-r|-e] [-c on page 2389] [-n [-1]] [-s] [-G|-M] [-R] [-u user_list]
[job_identifier... | destination...]
qstat -Q [-f [-1]] [-c on page 2389] [-W site_specific] [destination...]
qstat -q [-c on page 2389] [-G|-M] [destination...]
qstat -B [-c on page 2389] [-f [-1]][-W site_specific] [server_name...]
qstat -t [-c on page 2389]

Description

The qstat command is used to request the status of jobs, queues, or a batch server. The requested status is written to standard out.

When requesting job status, synopsis format 1 or 2, qstat will output information about each job_identifier or all jobs at each destination. Jobs for which the user does not have status privilege are not displayed.

When requesting queue or server status, synopsis format 3 through 5, qstat will output information about each destination.

> You can configure TORQUE with `CFLAGS='DTXT'` to change the alignment of text in qstat output. This noticeably improves `qstat -r` output.

Options

Option	Description
-c	Completed jobs are not displayed in the output. If desired, you can set the `PBS_QSTAT_NO_COMPLETE` environment variable to cause all qstat requests to not show completed jobs by default.
-f	Specifies that a full status display be written to standard out. The [time] value is the amount of walltime, in seconds, remaining for the job. [time] does not account for walltime multipliers.
-a	All jobs are displayed in the alternative format (see **Standard output** on page 2391). If the operand is a destination id, all jobs at that destination are displayed. If the operand is a job id, information about that job is displayed.
-e	If the operand is a job id or not specified, only jobs in executable queues are displayed. Setting the PBS_QSTAT_EXECONLY environment variable will also enable this option.
-i	Job status is displayed in the alternative format. For a destination id operand, statuses for jobs at that destination which are not running are displayed. This includes jobs which are queued, held or waiting. If an operand is a job id, status for that job is displayed regardless of its state.
-r	If an operand is a job id, status for that job is displayed. For a destination id operand, statuses for jobs at that destination which are running are displayed; this includes jobs which are suspended. Note that if there is no walltime given for a job, then elapsed time does not display.
-n	In addition to the basic information, nodes allocated to a job are listed.
-1	In combination with **-n**, the -1 option puts all of the nodes on the same line as the job ID. In combination with **-f**, attributes are not folded to fit in a terminal window. This is intended to ease the parsing of the qstat output.
-s	In addition to the basic information, any comment provided by the batch administrator or scheduler is shown.

Option	Description
-G	Show size information in giga-bytes.
-M	Show size information, disk or memory in mega-words. A word is considered to be 8 bytes.
-R	In addition to other information, disk reservation information is shown. Not applicable to all systems.
-t	Normal qstat output displays a summary of the array instead of the entire array, job for job. `qstat -t` expands the output to display the entire array. Note that arrays are now named with brackets following the array name; for example: `dbeer@napali:~/dev/torque/array_changes$ echo sleep 20 | qsub -t 0-299 189[].napali` Individual jobs in the array are now also noted using square brackets instead of dashes; for example, here is part of the output of `qstat -t` for the preceding array: `189[299].napali STDIN[299] dbeer 0 Q batch`
-u	Job status is displayed in the alternative format. If an operand is a job id, status for that job is displayed. For a destination id operand, statuses for jobs at that destination which are owned by the user(s) listed in user_list are displayed. The syntax of the user_list is: `user_name[@host][,user_name[@host],...]` Host names may be wild carded on the left end, e.g. "*.nasa.gov". User_name without a "@host" is equivalent to "user_name@*", that is at any host.
-Q	Specifies that the request is for queue status and that the operands are destination identifiers.
-q	Specifies that the request is for queue status which should be shown in the alternative format.
-B	Specifies that the request is for batch server status and that the operands are the names of servers.

Operands

If neither the **-Q** nor the **-B** option is given, the operands on the qstat command must be either job identifiers or destinations identifiers.

If the operand is a job identifier, it must be in the following form:

```
sequence_number[.server_name][@server]
```

where *sequence_number.server_name* is the job identifier assigned at submittal time (see **qsub**). If the *.server_name* is omitted, the name of the default server will be used. If *@server* is supplied, the request will be for the job identifier currently at that Server.

If the operand is a destination identifier, it is one of the following three forms:

- queue
- @server
- queue@server

If queue is specified, the request is for status of all jobs in that queue at the default server. If the @server form is given, the request is for status of all jobs at that server. If a full destination identifier, queue@server, is given, the request is for status of all jobs in the named queue at the named server.

If the **-Q** option is given, the operands are destination identifiers as specified above. If queue is specified, the status of that queue at the default server will be given. If queue@server is specified, the status of the named queue at the named server will be given. If @server is specified, the status of all queues at the named server will be given. If no destination is specified, the status of all queues at the default server will be given.

If the **-B** option is given, the operand is the name of a server.

Standard output

Displaying job status

If job status is being displayed in the default format and the **-f** option is not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the job identifier assigned by PBS.
- the job name given by the submitter.
- the job owner.
- the CPU time used.
- the job state:

Item	Description
C	Job is completed after having run.
E	Job is exiting after having run.
H	Job is held.
Q	Job is queued, eligible to run or routed.
R	Job is running.
T	Job is being moved to new location.

Item	Description
W	Job is waiting for its execution time (-a option) to be reached.
S	(Unicos only) Job is suspended.

- the queue in which the job resides.

If job status is being displayed and the -f option is specified, the output will depend on whether qstat was compiled to use a Tcl interpreter. See **Configuration** on page 2394 for details. If Tcl is not being used, full display for each job consists of the header line:

```
Job Id: job identifier
```

Followed by one line per job attribute of the form:

```
attribute_name = value
```

If any of the options **-a**, **-i**, **-r**, **-u**, **-n**, **-s**, **-G**, or **-M** are provided, the alternative display format for jobs is used. The following items are displayed on a single line, in the specified order, separated by white space:

- the job identifier assigned by PBS
- the job owner
- the queue in which the job currently resides
- the job name given by the submitter
- the session id (if the job is running)
- the number of nodes requested by the job
- the number of cpus or tasks requested by the job
- the amount of memory requested by the job
- either the cpu time, if specified, or wall time requested by the job, (hh:mm)
- the jobs current state
- the amount of cpu time or wall time used by the job (hh:mm)

If the **-r** option is provided, the line contains:

- the job identifier assigned by PBS
- the job owner
- the queue in which the job currently resides
- the number of nodes requested by the job
- the number of cpus or tasks requested by the job
- the amount of memory requested by the job
- either the cpu time or wall time requested by the job

- the jobs current state
- the amount of cpu time or wall time used by the job
- the amount of SRFS space requested on the big file system
- the amount of SRFS space requested on the fast file system
- the amount of space requested on the parallel I/O file system

The last three fields may not contain useful information at all sites or on all systems

Displaying queue status

If queue status is being displayed and the **-f** option was not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the queue name
- the maximum number of jobs that may be run in the queue concurrently
- the total number of jobs in the queue
- the enable or disabled status of the queue
- the started or stopped status of the queue
- for each job state, the name of the state and the number of jobs in the queue in that state
- the type of queue, execution or routing

If queue status is being displayed and the **-f** option is specified, the output will depend on whether qstat was compiled to use a Tcl interpreter. See **Configuration** on page 2394 for details. If Tcl is not being used, the full display for each queue consists of the header line:

```
Queue: queue_name
```

Followed by one line per queue attribute of the form:

```
attribute_name = value
```

If the **-Q** option is specified, queue information is displayed in the alternative format: The following information is displayed on a single line:

- the queue name
- the maximum amount of memory a job in the queue may request
- the maximum amount of cpu time a job in the queue may request
- the maximum amount of wall time a job in the queue may request
- the maximum amount of nodes a job in the queue may request
- the number of jobs in the queue in the running state
- the number of jobs in the queue in the queued state
- the maximum number (limit) of jobs that may be run in the queue concurrently

- the state of the queue given by a pair of letters:
  - either the letter *E* if the queue is Enabled or *D* if Disabled

    and
  - either the letter *R* if the queue is Running (started) or *S* if Stopped.

Displaying server status

If batch server status is being displayed and the **-f** option is not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the server name
- the maximum number of jobs that the server may run concurrently
- the total number of jobs currently managed by the server
- the status of the server
- for each job state, the name of the state and the number of jobs in the server in that state

If server status is being displayed and the **-f** option is specified, the output will depend on whether qstat was compiled to use a Tcl interpreter. See **Configuration** on page 2394 for details. If Tcl is not being used, the full display for the server consists of the header line:

```
Server: server name
```

Followed by one line per server attribute of the form:

```
attribute_name = value
```

Standard error

The qstat command will write a diagnostic message to standard error for each error occurrence.

Configuration

If qstat is compiled with an option to include a Tcl interpreter, using the **-f** flag to get a full display causes a check to be made for a script file to use to output the requested information. The first location checked is `$HOME/.qstatrc`. If this does not exist, the next location checked is administrator configured. If one of these is found, a Tcl interpreter is started and the script file is passed to it along with three global variables. The command line arguments are split into two variable named flags and operands . The status information is passed in a variable named objects . All of these variables are Tcl lists. The flags list contains the name of the command (usually "qstat") as its first element. Any other elements are command line option flags with any options they use, presented in the order given on the command line. They are broken up individually so that if two flags are given together on the command line, they are separated in the list. For example, if the user typed:

```
qstat -QfWbigdisplay
```

the flags list would contain

```
qstat -Q -f -W bigdisplay
```

The operands list contains all other command line arguments following the flags. There will always be at least one element in operands because if no operands are typed by the user, the default destination or server name is used. The objects list contains all the information retrieved from the server(s) so the Tcl interpreter can run once to format the entire output. This list has the same number of elements as the operands list. Each element is another list with two elements.

The first element is a string giving the type of objects to be found in the second. The string can take the values "server", "queue", "job" or "error".

The second element will be a list in which each element is a single batch status object of the type given by the string discussed above. In the case of "error", the list will be empty. Each object is again a list. The first element is the name of the object. The second is a list of attributes.

The third element will be the object text.

All three of these object elements correspond with fields in the structure batch_status which is described in detail for each type of object by the man pages for pbs_statjob(3), pbs_statque(3), and pbs_statserver (3). Each attribute in the second element list whose elements correspond with the attrl structure. Each will be a list with two elements. The first will be the attribute name and the second will be the attribute value.

Exit status

Upon successful processing of all the operands presented to the qstat command, the exit status will be a value of zero.

If the qstat command fails to process any operand, the command exits with a value greater than zero.

Related topics

- qalter(1B)
- qsub(1B)

Non-Adaptive Computing topics

- pbs_alterjob(3B)
- pbs_statjob(3B)
- pbs_statque(3B)
- pbs_statserver(3B)
- pbs_submit(3B)
- pbs_job_attributes(7B)
- pbs_queue_attributes(7B)
- pbs_server_attributes(7B)
- qmgr query_other_jobs parameter (allow non-admin users to see other users' jobs
- pbs_resources_*(7B) where * is system type
- PBS ERS

qsub

Submit PBS job.

Synopsis

```
qsub [-a date_time] [-A account_string] [-b secs] [-c checkpoint_options]
 [-C directive_prefix] [-d path] [-D path] [-e path] [-f] [-F] [-h]
 [-I ] [-j join ] [-k keep ] [-l resource_list ]
 [-m mail_options] [-M user_list] [-n] [-N name] [-o path]
 [-p priority] [-P user[:group]] [-q destination] [-r c] [-S path_to_shell(s)]
 [-t array_request] [-u user_list]
 [-v variable_list] [-V] [-W additional_attributes] [-x] [-X] [-z] [script]
```

Description

To create a job is to submit an executable script to a batch server. The batch server will be the default server unless the **-q** option is specified. The command parses a script prior to the actual script execution; it does not execute a script itself. All script-writing rules remain in effect, including the "#!" at the head of the file (see discussion of PBS_DEFAULT under **Environment variables** on page 2411). Typically, the script is a shell script which will be executed by a command shell such as sh or csh.

Options on the qsub command allow the specification of attributes which affect the behavior of the job.

The qsub command will pass certain environment variables in the Variable_List attribute of the job. These variables will be available to the job. The value for the following variables will be taken from the environment of the qsub command: HOME, LANG, LOGNAME, PATH, MAIL, SHELL, and TZ. These values will be assigned to a new name which is the current name prefixed with the string "PBS_O_". For example, the job will have access to an environment variable named PBS_O_HOME which have the value of the variable HOME in the qsub command environment.

In addition to the above, the following environment variables will be available to the batch job:

Variable	Description
PBS_O_HOST	The name of the host upon which the qsub command is running.
PBS_SERVER	The hostname of the pbs_server which qsub submits the job to.
PBS_O_QUEUE	The name of the original queue to which the job was submitted.
PBS_O_WORKDIR	The absolute path of the current working directory of the qsub command.
PBS_ARRAYID	Each member of a job array is assigned a unique identifier (see **-t** option).
PBS_ENVIRONMENT	Set to PBS_BATCH to indicate the job is a batch job, or to PBS_INTERACTIVE to indicate the job is a PBS interactive job (see **-I** option).
PBS_GPUFILE	The name of the file containing the list of assigned GPUs. For more information about how to set up TORQUE with GPUS, see **Accelerators** on page 794.

Variable	Description
PBS_JOBID	The job identifier assigned to the job by the batch system. It can be used in the stdout and stderr paths. TORQUE replaces `$PBS_JOBID` with the job's jobid (for example, `#PBS -o /tmp/$PBS_JOBID.output`).
PBS_JOBNAME	The job name supplied by the user.
PBS_NODEFILE	The name of the file contains the list of nodes assigned to the job (for parallel and cluster systems).
PBS_QUEUE	The name of the queue from which the job is executed.

Options

Option	Name	Description
-a	date_time	Declares the time after which the job is eligible for execution. The date_time argument is in the form: `[[[[CC]YY]MM]DD]hhmm[.SS]` where *CC* is the first two digits of the year (the century), *YY* is the second two digits of the year, *MM* is the two digits for the month, *DD* is the day of the month, *hh* is the hour, *mm* is the minute, and the optional *SS* is the seconds. If the month (MM) is not specified, it will default to the current month if the specified day (DD) is in the future. Otherwise, the month will be set to next month. Likewise, if the day (DD) is not specified, it will default to today if the time (hhmm) is in the future. Otherwise, the day will be set to tomorrow. For example, if you submit a job at 11:15 am with a time of `-a 1110`, the job will be eligible to run at 11:10 am tomorrow.
-A	account_ string	Defines the account string associated with the job. The account_string is an undefined string of characters and is interpreted by the server which executes the job. See section 2.7.1 of the PBS ERS.
-b	seconds	Defines the maximum number of seconds qsub will block attempting to contact pbs_server. If pbs_server is down, or for a variety of communication failures, qsub will continually retry connecting to pbs_server for job submission. This value overrides the CLIENTRETRY parameter in `torque.cfg`. This is a non-portable TORQUE extension. Portability-minded users can use the PBS_CLIENTRETRY environmental variable. A negative value is interpreted as infinity. The default is 0.

<table>
<tr><th>Option</th><th>Name</th><th>Description</th></tr>
<tr><td>-c</td><td>checkpoint_
options</td><td>Defines the options that will apply to the job. If the job executes upon a host which does not support checkpoint, these options will be ignored.
Valid checkpoint options are:
none – No checkpointing is to be performed.
enabled – Specify that checkpointing is allowed but must be explicitly invoked by either the qhold or qchkpt commands.
shutdown – Specify that checkpointing is to be done on a job at pbs_mom shutdown.
periodic – Specify that periodic checkpointing is enabled. The default interval is 10 minutes and can be changed by the $checkpoint_interval option in the MOM config file or by specifying an interval when the job is submitted
interval=minutes – Checkpointing is to be performed at an interval of minutes, which is the integer number of minutes of wall time used by the job. This value must be greater than zero.
depth=number – Specify a number (depth) of checkpoint images to be kept in the checkpoint directory.
dir=path – Specify a checkpoint directory (default is <code>/var/spool/torque/checkpoint</code>).</td></tr>
<tr><td>-C</td><td>directive_
prefix</td><td>Defines the prefix that declares a directive to the qsub command within the script file. (See the paragraph on script directives under Extended description on page 2412.)
If the <code>-C</code> option is presented with a directive_prefix argument that is the null string, qsub will not scan the script file for directives.</td></tr>
<tr><td>-d</td><td>path</td><td>Defines the working directory path to be used for the job. If the <code>-d</code> option is not specified, the default working directory is the home directory. This option sets the environment variable PBS_O_INITDIR.</td></tr>
<tr><td>-D</td><td>path</td><td>Defines the root directory to be used for the job. This option sets the environment variable PBS_O_ROOTDIR.</td></tr>
</table>

<table>
<tr><th>Option</th><th>Name</th><th>Description</th></tr>
<tr><td>-e</td><td>path</td><td>Defines the path to be used for the standard error stream of the batch job. The path argument is of the form:
<code>[hostname:]path_name</code>
where hostname is the name of a host to which the file will be returned, and path_name is the path name on that host in the syntax recognized by POSIX.

When specifying a directory for the location you need to include a trailing slash.

The argument will be interpreted as follows:
path_name – where path_name is not an absolute path name, then the qsub command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component.
hostname:path_name – where path_name is not an absolute path name, then the qsub command will not expand the path name relative to the current working directory of the command. On delivery of the standard error, the path name will be expanded relative to the user's home directory on the hostname system.
path_name – where path_name specifies an absolute path name, then the qsub will supply the name of the host on which it is executing for the hostname.
hostname:path_name – where path_name specifies an absolute path name, the path will be used as specified.
If the <code>-e</code> option is not specified, the default file name for the standard error stream will be used. The default name has the following form:
job_name.esequence_number – where job_name is the name of the job (see the -n name option) and sequence_number is the job number assigned when the job is submitted.</td></tr>
<tr><td>-f</td><td>---</td><td>Job is made fault tolerant. Jobs running on multiple nodes are periodically polled by mother superior. If one of the nodes fails to report, the job is canceled by mother superior and a failure is reported. If a job is fault tolerant, it will not be canceled based on failed polling (no matter how many nodes fail to report). This may be desirable if transient network failures are causing large jobs not to complete, where ignoring one failed polling attempt can be corrected at the next polling attempt.

If TORQUE is compiled with PBS_NO_POSIX_VIOLATION (there is no config option for this), you have to use <code>-W fault_tolerant=true</code> to mark the job as fault tolerant.</td></tr>
</table>

Option	Name	Description
-F	---	Specifies the arguments that will be passed to the job script when the script is launched. The accepted syntax is: `qsub -F "myarg1 myarg2 myarg3=myarg3value" myscript2.sh` Quotation marks are required. qsub will fail with an error message if the argument following `-F` is not a quoted value. The pbs_mom server will pass the quoted value as arguments to the job script when it launches the script.
-h	---	Specifies that a user hold be applied to the job at submission time.
-I	---	Declares that the job is to be run "interactively". The job will be queued and scheduled as any PBS batch job, but when executed, the standard input, output, and error streams of the job are connected through qsub to the terminal session in which qsub is running. Interactive jobs are forced to not rerunable. See **Extended description** on page 2412 for additional information of interactive jobs.
-j	join	Declares if the standard error stream of the job will be merged with the standard output stream of the job. An option argument value of *oe* directs that the two streams will be merged, intermixed, as standard output. An option argument value of *eo* directs that the two streams will be merged, intermixed, as standard error. If the join argument is *n* or the option is not specified, the two streams will be two separate files. If using either the -e or the -o option and the `-j eo|oe` option, the `-j` option takes precedence and all standard error and output messages go to the chosen output file.

<table>
<tr><th>Option</th><th>Name</th><th>Description</th></tr>
<tr><td>-k</td><td>keep</td><td>Defines which (if either) of standard output or standard error will be retained on the execution host. If set for a stream, this option overrides the path name for that stream. If not set, neither stream is retained on the execution host.

The argument is either the single letter "e" or "o", or the letters "e" and "o" combined in either order. Or the argument is the letter "n".
e – The standard error stream is to be retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by:
<code>job_name.esequence</code>
where job_name is the name specified for the job, and sequence is the sequence number component of the job identifier.
o – The standard output stream is to be retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by:
<code>job_name.osequence</code>
where job_name is the name specified for the job, and sequence is the sequence number component of the job identifier.
eo – Both the standard output and standard error streams will be retained.
oe – Both the standard output and standard error streams will be retained.
n – Neither stream is retained.</td></tr>
<tr><td>-l</td><td>resource_
list</td><td>Defines the resources that are required by the job and establishes a limit to the amount of resource that can be consumed. If not set for a generally available resource, such as CPU time, the limit is infinite. The resource_list argument is of the form:
<code>resource_name[=[value]][,resource_name[=[value]],...]</code>

In this situation, you should request the more inclusive resource first. For example, a request for procs should come before a gres request.

In TORQUE 3.0.2 or later, qsub supports the mapping of <code>-l gpus=X</code> to <code>-l gres=gpus:X</code>. This allows users who are using NUMA systems to make requests such as <code>-l ncpus=20:gpus=5</code> indicating they are not concerned with the GPUs in relation to the NUMA nodes they request, they only want a total of 20 cores and 5 GPUs.

For more information, see Requesting Resources on page 2237.

For information on specifying multiple types of resources for allocation, see Multi-Req Support on page 532.</td></tr>
</table>

Option	Name	Description
-m	mail_ options	Defines the set of conditions under which the execution server will send a mail message about the job. The mail_options argument is a string which consists of either the single character "n", or one or more of the characters "a", "b", and "e". If the character "n" is specified, no normal mail is sent. Mail for job cancels and other events outside of normal job processing are still sent. For the letters "a", "b", and "e": • *a* – Mail is sent when the job is aborted by the batch system. • *b* – Mail is sent when the job begins execution. • *e* – Mail is sent when the job terminates. If the `-m` option is not specified, mail will be sent if the job is aborted.
-M	user_list	Declares the list of users to whom mail is sent by the execution server when it sends mail about the job. The user_list argument is of the form: `user[@host][,user[@host],...]` If unset, the list defaults to the submitting user at the qsub host, i.e. the job owner.
-n	node-exclusive	Allows a user to specify an exclusive-node access/allocation request for the job. This affects only cpusets and compatible schedulers (see **Linux cpuset Support** on page 2274).
-N	name	Declares a name for the job. The name specified may be an unlimited number of characters in length. It must consist of printable, nonwhite space characters with the first character alphabetic. If the `-N` option is not specified, the job name will be the base name of the job script file specified on the command line. If no script file name was specified and the script was read from the standard input, then the job name will be set to STDIN.

<table>
<tr><th>Option</th><th>Name</th><th>Description</th></tr>
<tr><td>-o</td><td>path</td><td>Defines the path to be used for the standard output stream of the batch job. The path argument is of the form:
<code>[hostname:]path_name</code>
where hostname is the name of a host to which the file will be returned, and path_name is the path name on that host in the syntax recognized by POSIX.
When specifying a directory for the location you need to include a trailing slash.
The argument will be interpreted as follows:
path_name – where path_name is not an absolute path name, then the qsub command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component.
hostname:path_name – where path_name is not an absolute path name, then the qsub command will not expand the path name relative to the current working directory of the command. On delivery of the standard output, the path name will be expanded relative to the user's home directory on the hostname system.
path_name – where path_name specifies an absolute path name, then the qsub will supply the name of the host on which it is executing for the hostname.
hostname:path_namewhere path_name specifies an absolute path name, the path will be used as specified.
If the <code>-o</code> option is not specified, the default file name for the standard output stream will be used. The default name has the following form:
job_name.osequence_number – where job_name is the name of the job (see the -n name option) and sequence_number is the job number assigned when the job is submitted.</td></tr>
<tr><td>-p</td><td>priority</td><td>Defines the priority of the job. The priority argument must be a integer between -1024 and +1023 inclusive. The default is no priority which is equivalent to a priority of zero.</td></tr>
<tr><td>-P</td><td>user [:group]</td><td>Allows a root user or manager to submit a job as another user. TORQUE treats proxy jobs as though the jobs were submitted by the supplied username. This feature is available in TORQUE 2.4.7 and later, however, TORQUE 2.4.7 does not have the ability to supply the <code>[:group]</code> option; it is available in TORQUE 2.4.8 and later.</td></tr>
</table>

Option	Name	Description
-q	destination	Defines the destination of the job. The destination names a queue, a server, or a queue at a server. The qsub command will submit the script to the server defined by the destination argument. If the destination is a routing queue, the job may be routed by the server to a new destination. If the -q option is not specified, the qsub command will submit the script to the default server. (See **Environment variables** on page 2411 and the PBS ERS section 2.7.4, "Default Server".) If the -q option is specified, it is in one of the following three forms: • queue • @server • queue@server If the destination argument names a queue and does not name a server, the job will be submitted to the named queue at the default server. If the destination argument names a server and does not name a queue, the job will be submitted to the default queue at the named server. If the destination argument names both a queue and a server, the job will be submitted to the named queue at the named server.
-r	y/n	Declares whether the job is rerunable (see the **qrerun** command). The option argument is a single character, either y or n. If the argument is "y", the job is rerunable. If the argument is "n", the job is not rerunable. The default value is y, rerunable.
-S	path_list	Declares the path to the desires shell for this job. `qsub script.sh -S /bin/tcsh` If the shell path is different on different compute nodes, use the following syntax: `path[@host][,path[@host],...]` `qsub script.sh -S /bin/tcsh@node1,/usr/bin/tcsh@node2` Only one path may be specified for any host named. Only one path may be specified without the corresponding host name. The path selected will be the one with the host name that matched the name of the execution host. If no matching host is found, then the path specified without a host will be selected, if present. If the `-S` option is not specified, the option argument is the null string, or no entry from the path_list is selected, the execution will use the user's login shell on the execution host.

Option	Name	Description
-t	array_ request	Specifies the task ids of a job array. Single task arrays are allowed. The array_request argument is an integer id or a range of integers. Multiple ids or id ranges can be combined in a comma delimited list. Examples: `-t 1-100 or -t 1,10,50-100` An optional *slot limit* can be specified to limit the amount of jobs that can run concurrently in the job array. The default value is unlimited. The slot limit must be the last thing specified in the array_request and is delimited from the array by a percent sign (%). `qsub script.sh -t 0-299%5` This sets the slot limit to 5. Only 5 jobs from this array can run at the same time. You can use **qalter** to modify slot limits on an array. The server parameter **max_slot_ limit** can be used to set a global slot limit policy.
-u	user_list	Defines the user name under which the job is to run on the execution system. The user_list argument is of the form: `user[@host][,user[@host],...]` Only one user name may be given per specified host. Only one of the user specifications may be supplied without the corresponding host specification. That user name will used for execution on any host not named in the argument list. If unset, the user list defaults to the user who is running qsub.
-v	variable_ list	Expands the list of environment variables that are exported to the job. In addition to the variables described in the "Description" section above, variable_list names environment variables from the qsub command environment which are made available to the job when it executes. The variable_list is a comma separated list of strings of the form `variable` or `variable=value`. These variables and their values are passed to the job. Note that `-v` has a higher precedence than `-V`, so identically named variables specified via `-v` will provide the final value for an environment variable in the job.
-V	---	Declares that all environment variables in the qsub commands environment are to be exported to the batch job.

Option	Name	Description
-W	additional_ attributes	The `-W` option allows for the specification of additional job attributes. The general syntax of `-W` is in the form: `-W attr_name=attr_value`. You can use multiple `-W` options with this syntax: `-W attr_name1=attr_value1 -W attr_name2=attr_value2`. If white space occurs anywhere within the option argument string or the equal sign, "=", occurs within an attribute_value string, then the string must be enclosed with either single or double quote marks. PBS currently supports the following attributes within the `-W` option: • *depend=dependency_list* – Defines the dependency between this and other jobs. The dependency_list is in the form: `type[:argument[:argument...][,type:argument...]` The argument is either a numeric count or a PBS job id according to type. If argument is a count, it must be greater than 0. If it is a job id and not fully specified in the form `seq_number.server.name`, it will be expanded according to the default server rules which apply to job IDs on most commands. If argument is null (the preceding colon need not be specified), the dependency of the corresponding type is cleared (unset). For more information, see **depend=dependency_list valid dependencies** on page 2407. • *group_list=g_list* – Defines the group name under which the job is to run on the execution system. The g_list argument is of the form: `group[@host][,group[@host],...]` Only one group name may be given per specified host. Only one of the group specifications may be supplied without the corresponding host specification. That group name will used for execution on any host not named in the argument list. If not set, the group_list defaults to the primary group of the user under which the job will be run. • *interactive=true* – If the interactive attribute is specified, the job is an interactive job. The **-I** option is an alternative method of specifying this attribute. • *job_radix=<int>* – To be used with parallel jobs. It directs the Mother Superior of the job to create a distribution radix of size *<int>* between sisters. See **Managing Multi-node Jobs** on page 2236. • *stagein=file_list* • *stageout=file_list* – Specifies which files are staged (copied) in before job start or staged out after the job completes execution. On completion of the job, all staged-in and staged-out files are removed from the execution system. The file_list is in the form: `local_file@hostname:remote_file[,...]` regardless of the direction of the copy. The name local_file is the name of the file on the system where the job executed. It may be an absolute path or relative to the home directory of the user. The name remote_file is the destination name on the host specified by hostname. The name may be

Option	Name	Description
		absolute or relative to the user's home directory on the destination host. The use of wildcards in the file name is not recommended. The file names map to a remote copy program (rcp) call on the execution system in the follow manner: ◦ For stagein: `rcp hostname:remote_file local_file` ◦ For stageout: `rcp local_file hostname:remote_file` Data staging examples: `-W stagein=/tmp/input.txt@headnode:/home/user/input.txt` `-W stageout=/tmp/output.txt@headnode:/home/user/output.txt` If TORQUE has been compiled with wordexp support, then variables can be used in the specified paths. Currently only `$PBS_JOBID`, `$HOME`, and `$TMPDIR` are supported for stagein. • *umask=XXX* – Sets umask used to create stdout and stderr spool files in pbs_mom spool directory. Values starting with 0 are treated as octal values, otherwise the value is treated as a decimal umask value.
-x	---	By default, if you submit an interactive job with a script, the script will be parsed for PBS directives but the rest of the script will be ignored since it's an interactive job. The `-x` option allows the script to be executed in the interactive job and then the job completes. For example: script.sh #!/bin/bash ls ---end script--- qsub -I script.sh qsub: waiting for job 5.napali to start dbeer@napali:# <displays the contents of the directory, because of the ls command> qsub: job 5.napali completed
-X	---	Enables X11 forwarding. The DISPLAY environment variable must be set.
-z	---	Directs that the qsub command is not to write the job identifier assigned to the job to the commands standard output.

depend=dependency_list valid dependencies

For job dependencies to work correctly, you must set the **keep_completed** on page 2424 server parameter.

Dependency	Description
`synccount:count`	This job is the first in a set of jobs to be executed at the same time. Count is the number of additional jobs in the set.
`syncwith:jobid`	This job is an additional member of a set of jobs to be executed at the same time. In the above and following dependency types, jobid is the job identifier of the first job in the set.
`after:jobid[:jobid...]`	This job may be scheduled for execution at any point after jobs jobid have started execution.
`afterok:jobid[:jobid...]`	This job may be scheduled for execution only after jobs jobid have terminated with no errors. See the csh warning under **Extended description** on page 2412.
`afternotok:jobid[:jobid...]`	This job may be scheduled for execution only after jobs jobid have terminated with errors. See the csh warning under **Extended description** on page 2412.
`afterany:jobid[:jobid...]`	This job may be scheduled for execution after jobs jobid have terminated, with or without errors.
`on:count`	This job may be scheduled for execution after count dependencies on other jobs have been satisfied. This form is used in conjunction with one of the "before" forms (see below).
`before:jobid[:jobid...]`	When this job has begun execution, then jobs jobid... may begin.
`beforeok:jobid[:jobid...]`	If this job terminates execution without errors, then jobs jobid... may begin. See the csh warning under **Extended description** on page 2412.
`beforenotok:jobid[:jobid...]`	If this job terminates execution with errors, then jobs jobid... may begin. See the csh warning under **Extended description** on page 2412.

Dependency	Description
`beforeany:jobid[:jobid...]`	When this job terminates execution, jobs jobid... may begin. If any of the before forms are used, the jobs referenced by jobid must have been submitted with a dependency type of on. If any of the before forms are used, the jobs referenced by jobid must have the same owner as the job being submitted. Otherwise, the dependency is ignored.
Array dependencies make a job depend on an array or part of an array. If no count is given, then the entire array is assumed. For examples, see **Dependency examples** on page 2410.	
`afterstartarray:arrayid[count]`	After this many jobs have started from arrayid, this job may start.
`afterokarray:arrayid[count]`	This job may be scheduled for execution only after jobs in arrayid have terminated with no errors.
`afternotokarray:arrayid[count]`	This job may be scheduled for execution only after jobs in arrayid have terminated with errors.
`afteranyarray:arrayid[count]`	This job may be scheduled for execution after jobs in arrayid have terminated, with or without errors.
`beforestartarray:arrayid[count]`	Before this many jobs have started from arrayid, this job may start.
`beforeokarray:arrayid[count]`	If this job terminates execution without errors, then jobs in arrayid may begin.
`beforenotokarray:arrayid[count]`	If this job terminates execution with errors, then jobs in arrayid may begin.
`beforeanyarray:arrayid[count]`	When this job terminates execution, jobs in arrayid may begin. If any of the before forms are used, the jobs referenced by arrayid must have been submitted with a dependency type of on. If any of the before forms are used, the jobs referenced by arrayid must have the same owner as the job being submitted. Otherwise, the dependency is ignored.

Dependency	Description
Error processing of the existence, state, or condition of the job on which the newly submitted job is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the new job will be deleted by the server. Mail will be sent to the job submitter stating the error.	

Dependency examples

```
qsub -W depend=afterok:123.big.iron.com /tmp/script
```

```
qsub -W depend=before:234.hunk1.com:235.hunk1.com
```

```
/tmp/script
```

```
qsub script.sh -W depend=afterokarray:427[]
```

(This assumes every job in array 427 has to finish successfully for the dependency to be satisfied.)

```
qsub script.sh -W depend=afterokarray:427[][5]
```

(This means that 5 of the jobs in array 427 have to successfully finish in order for the dependency to be satisfied.)

Operands

The qsub command accepts a script operand that is the path to the script of the job. If the path is relative, it will be expanded relative to the working directory of the qsub command.

If the script operand is not provided or the operand is the single character "-", the qsub command reads the script from standard input. When the script is being read from Standard Input, qsub will copy the file to a temporary file. This temporary file is passed to the library interface routine pbs_submit. The temporary file is removed by qsub after pbs_submit returns or upon the receipt of a signal which would cause qsub to terminate.

Standard input

The qsub command reads the script for the job from standard input if the script operand is missing or is the single character "-".

Input files

The script file is read by the qsub command. qsub acts upon any directives found in the script.

When the job is created, a copy of the script file is made and that copy cannot be modified.

Standard output

Unless the -z option is set, the job identifier assigned to the job will be written to standard output if the job is successfully created.

Standard error

The qsub command will write a diagnostic message to standard error for each error occurrence.

Environment variables

The values of some or all of the variables in the qsub commands environment are exported with the job (see the **-v** and **-v** options).

The environment variable PBS_DEFAULT defines the name of the default server. Typically, it corresponds to the system name of the host on which the server is running. If PBS_DEFAULT is not set, the default is defined by an administrator established file.

The environment variable PBS_DPREFIX determines the prefix string which identifies directives in the script.

The environment variable PBS_CLIENTRETRY defines the maximum number of seconds qsub will block (see the **-b** option). Despite the name, currently qsub is the only client that supports this option.

torque.cfg

The `torque.cfg` file, located in PBS_SERVER_HOME (`/var/spool/torque` by default) controls the behavior of the qsub command. This file contains a list of parameters and values separated by whitespace.

- *QSUBSLEEP* – takes an integer operand which specifies time to sleep when running qsub command. Used to prevent users from overwhelming the scheduler.
- *SUBMITFILTER* – specifies the path to the submit filter used to pre-process job submission. The default path is libexecdir/qsub_filter, which falls back to /usr/local/sbin/torque_submitfilter for backwards compatibility. This `torque.cfg` parameter overrides this default.
- *SERVERHOST*
- *QSUBHOST*
- *QSUBSENDUID*
- *XAUTHPATH*
- *CLIENTRETRY*
- *VALIDATEGROUP*
- *DEFAULTCKPT*
- *VALIDATEPATH*
- *RERUNNABLEBYDEFAULT*

For example:

```
QSUBSLEEP 2
```

```
RERUNNABLEBYDEFAULT false
```

Extended description

Script Processing:

A job script may consist of PBS directives, comments and executable statements. A PBS directive provides a way of specifying job attributes in addition to the command line options. For example:

```
:
#PBS -N Job_name
#PBS -l walltime=10:30,mem=320kb
#PBS -m be
#
step1 arg1 arg2
step2 arg3 arg4
```

The qsub command scans the lines of the script file for directives. An initial line in the script that begins with the characters "#!" or the character ":" will be ignored and scanning will start with the next line. Scanning will continue until the first executable line, that is a line that is not blank, not a directive line, nor a line whose first nonwhite space character is "#". If directives occur on subsequent lines, they will be ignored.

A line in the script file will be processed as a directive to qsub if and only if the string of characters starting with the first nonwhite space character on the line and of the same length as the directive prefix matches the directive prefix.

The remainder of the directive line consists of the options to qsub in the same syntax as they appear on the command line. The option character is to be preceded with the "-" character.

If an option is present in both a directive and on the command line, that option and its argument, if any, will be ignored in the directive. The command line takes precedence.

If an option is present in a directive and not on the command line, that option and its argument, if any, will be processed as if it had occurred on the command line.

The directive prefix string will be determined in order of preference from:

- The value of the **-c** option argument if the option is specified on the command line.
- The value of the environment variable PBS_DPREFIX if it is defined.
- The four character string #PBS.

If the **-c** option is found in a directive in the script file, it will be ignored.

User Authorization:

When the user submits a job from a system other than the one on which the PBS Server is running, the name under which the job is to be executed is selected according to the rules listed under the **-u** option. The user submitting the job must be authorized to run the job under the execution user name. This authorization is provided if:

- The host on which qsub is run is trusted by the execution host (see `/etc/hosts.equiv`).
- The execution user has an `.rhosts` file naming the submitting user on the submitting host.

C-Shell .logout File:

The following warning applies for users of the c-shell, csh. If the job is executed under the csh and a `.logout` file exists in the home directory in which the job executes, the exit status of the job is that of the `.logout` script, not the job script. This may impact any inter-job dependencies. To preserve the job exit status, either remove the `.logout` file or place the following line as the first line in the `.logout` file:

```
set EXITVAL = $status
```

and the following line as the last executable line in `.logout`:

```
exit $EXITVAL
```

Interactive Jobs:

If the **-I** option is specified on the command line or in a script directive, or if the "interactive" job attribute declared true via the **-W** option, `-W interactive=true`, either on the command line or in a script directive, the job is an interactive job. The script will be processed for directives, but will not be included with the job. When the job begins execution, all input to the job is from the terminal session in which qsub is running.

When an interactive job is submitted, the qsub command will not terminate when the job is submitted. qsub will remain running until the job terminates, is aborted, or the user interrupts qsub with an SIGINT (the control-C key). If qsub is interrupted prior to job start, it will query if the user wishes to exit. If the user response "yes", qsub exits and the job is aborted.

One the interactive job has started execution, input to and output from the job pass through qsub. Keyboard generated interrupts are passed to the job. Lines entered that begin with the tilde (~) character and contain special sequences are escaped by qsub. The recognized escape sequences are:

Sequence	Description
~.	qsub terminates execution. The batch job is also terminated.
~susp	Suspend the qsub program if running under the C shell. "susp" is the suspend character (usually CNTL-Z).
~asusp	Suspend the input half of qsub (terminal to job), but allow output to continue to be displayed. Only works under the C shell. "asusp" is the auxiliary suspend character, usually CNTL-Y.

Exit status

Upon successful processing, the qsub exit status will be a value of zero.

If the qsub command fails, the command exits with a value greater than zero.

Related topics

- qalter(1B)
- qdel(1B)

- qhold(1B)
- qrls(1B)
- qsig(1B)
- qstat(1B)
- pbs_server(8B)

Non-Adaptive Computing topics

- pbs_connect(3B)
- pbs_job_attributes(7B)
- pbs_queue_attributes(7B)
- pbs_resources_irix5(7B)
- pbs_resources_sp2(7B)
- pbs_resources_sunos4(7B)
- pbs_resources_unicos8(7B)
- pbs_server_attributes(7B)
- qselect(1B)
- qmove(1B)
- qmsg(1B)
- qrerun(1B)

qterm

Terminate processing by a PBS batch server.

Synopsis

```
qterm [-t type] [server...]
```

Description

The qterm command terminates a batch server. When a server receives a terminate command, the server will go into the "Terminating" state. No new jobs will be allowed to be started into execution or enqueued into the server. The impact on jobs currently being run by the server depends

In order to execute qterm, the user must have PBS Operation or Manager privileges.

Options

Option	Name	Description
-t	type	Specifies the type of shut down. The types are: • *immediate* – If checkpointing is supported, all running jobs are to immediately stop execution. If checkpointing is supported, running jobs that can be checkpointed are checkpointed, terminated, and requeued. If checkpoint is not supported or the job cannot be checkpointed, running jobs are requeued if the rerunable attribute is true. Otherwise, jobs are killed. • *delay* – If checkpointing is supported, running jobs that can be checkpointed are checkpointed, terminated, and requeued. If a job cannot be checkpointed, but can be rerun, the job is terminated and requeued. Otherwise, running jobs are allowed to continue to run. Note, the operator or administrator may use the **qrerun** and **qdel** commands to remove running jobs. • *quick* – This is the default action if the `-t` option is not specified. This option is used when you wish that running jobs be left running when the server shuts down. The server will cleanly shutdown and can be restarted when desired. Upon restart of the server, jobs that continue to run are shown as running; jobs that terminated during the server's absence will be placed into the exiting state.

Operands

The server operand specifies which servers are to shut down. If no servers are given, then the default server will be terminated.

Standard error

The qterm command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qterm command, the exit status will be a value of zero.

If the qterm command fails to process any operand, the command exits with a value greater than zero.

Related topics(non-Adaptive Computing topics)

- pbs_server(8B)
- qmgr(8B)
- pbs_resources_aix4(7B)
- pbs_resources_irix5(7B)
- pbs_resources_sp2(7B)

- pbs_resources_sunos4(7B)
- pbs_resources_unicos8(7B)

trqauthd

(TORQUE authorization daemon)

Synopsis

```
trqauthd -D
trqauthd -d
```

Description

The trqauthd daemon, introduced in TORQUE 4.0.0, replaced the pbs_iff authentication process. When users connect to pbs_server by calling one of the TORQUE utilities or by using the TORQUE APIs, the new user connection must be authorized by a trusted entity which runs as root. The advantage of trqauthd's doing this rather than pbs_iff is that trqauthd is resident, meaning you do not need to be loaded every time a connection is made; multi-threaded; scalable; and more easily adapted to new functionality than pbs_iff.

Beginning in TORQUE 4.2.6, trqauthd can remember the currently active pbs_server host, enhancing high availability functionality. Previously, trqauthd tried to connect to each host in the `$TORQUE_HOME/`*`<server_name>`* file until it could successfully connect. Because it now remembers the active server, it tries to connect to that server first. If it fails to connect, it will go through the *<server_name>* file and try to connect to a host where an active pbs_server is running.

Options

-D — Debug	
Format	---
Default	---
Description	Run trqauthd in debug mode.
Example	`trqauthd -D`

-d — Terminate	
Format	---

-d — Terminate	
Default	---
Description	Terminate trqauthd.
Example	`trqauthd -d`

Appendix B: Server Parameters

TORQUE server parameters are specified using the **qmgr** command. The `set` subcommand is used to modify the **server** object. For example:

```
> qmgr -c 'set server default_queue=batch'
```

Parameters

acl_hosts	
Format	<HOST>[,<HOST>]... *or* <HOST>[range] *or* <HOST*> where the asterisk (*) can appear anywhere in the host name
Default	(Only the host running pbs_server may submit jobs.)
Description	Specifies a list of hosts from which jobs may be submitted. Hosts in the server nodes file located at `$TORQUE/server_priv/nodes` cannot be added to the list using the acl_hosts parameter (see **Server Node File Configuration** on page 2222). To submit batch or interactive jobs (see **Server Configuration** on page 2215) through hosts that are specified in the server nodes file, use the **submit_hosts** parameter. `Qmgr: set queue batch acl_hosts = "hostA,hostB"` `Qmgr: set queue batch acl_hosts += "hostE,hostF,hostG"` In version 2.5 and later, the wildcard (*) character can appear anywhere in the host name, and ranges are supported; these specifications also work for managers and operators. `Qmgr: set server acl_hosts = "galaxy*.tom.org"` `Qmgr: set server acl_hosts += "galaxy[0-50].tom.org"` `Qmgr: set server managers+=tom@galaxy[0-50].tom.org`

acl_host_enable

Format	<BOOLEAN>
Default	**FALSE**
Description	When set to `TRUE`, specifies that the **acl_hosts** value is enabled.

acl_logic_or

Format	<BOOLEAN>
Default	**FALSE**
Description	When set to `TRUE`,the user and group queue ACL's are logically OR'd. When set to `FALSE`, they are AND'd.

allow_node_submit

Format	<BOOLEAN>
Default	**FALSE**
Description	When set to `TRUE`, specifies that users can submit jobs directly from any trusted compute host directly or from within batch jobs (see **Configuring job submission hosts** on page 2216). When you enable **allow_node_submit**, you must also enable the **allow_proxy_user** on page 2418 parameter to allow user proxying when submitting and running jobs.

allow_proxy_user

Format	<BOOLEAN>
Default	**FALSE**
Description	When set to `TRUE`, specifies that users can proxy from one user to another. Proxy requests will be either validated by `ruserok()` or by the scheduler (see **Job Submission** on page 2234).

auto_node_np	
Format	<BOOLEAN>
Default	**DISABLED**
Description	When set to `TRUE`, automatically configures a node's np (number of processors) value based on the ncpus value from the status update. Requires full manager privilege to set or alter.

automatic_requeue_exit_code	
Format	<LONG>
Default	---
Description	This is an exit code, defined by the admin, that tells pbs_server to requeue the job instead of considering it as completed. This allows the user to add some additional checks that the job can run meaningfully, and if not, then the job script exits with the specified code to be requeued.

checkpoint_defaults	
Format	<STRING>
Default	---
Description	Specifies for a queue the default checkpoint values for a job that does not have checkpointing specified. The checkpoint_defaults parameter only takes effect on execution queues. `set queue batch checkpoint_defaults="enabled, periodic, interval=5"`

clone_batch_delay	
Format	<INTEGER>
Default	1
Description	Specifies the delay (in seconds) between clone batches (see **clone_batch_size**).

clone_batch_size

Format	<INTEGER>
Default	256
Description	Job arrays are created in batches of size *X*. *X* jobs are created, and after the **clone_batch_delay**, *X* more are created. This repeats until all are created.

copy_on_rerun

Format	<BOOLEAN>
Default	FALSE
Description	When set to `TRUE`, TORQUE will copy the output and error files over to the user-specified directory when the grerun command is executed (i.e. a job preemption). Output and error files are only created when a job is in running state before the preemption occurs. pbs_server and pbs_mom need to be on the same version. When you change the value, you must perform a pbs_server restart for the change to effect.

cray_enabled

Format	<BOOLEAN>
Default	**FALSE**
Description	When set to `TRUE`, specifies that this instance of pbs_server has Cray hardware that reports to it. See **Installation Notes for Moab and TORQUE for Cray** on page 1214.

default_queue

Format	<STRING>
Default	---
Description	Indicates the queue to assign to a job if no queue is explicitly specified by the submitter.

disable_server_id_check

Format	<BOOLEAN>
Default	**FALSE**
Description	When set to `TRUE`, makes it so the user for the job doesn't have to exist on the server. The user must still exist on all the compute nodes or the job will fail when it tries to execute. If you have **disable_server_id_check** set to `TRUE`, a user could request a group to which they do not belong. Setting VALIDATEGROUP to `TRUE` in the `torque.cfg` file prevents such a scenario (see **Appendix K: "torque.cfg" Configuration File** on page 2480).

display_job_server_suffix

Format	<BOOLEAN>
Default	**TRUE**
Description	When set to `TRUE`, TORQUE will display both the job ID and the host name. When set to `FALSE`, only the job ID will be displayed. If set to *FALSE*, the environment variable *NO_SERVER_SUFFIX* must be set to *TRUE* for pbs_track to work as expected.

interactive_jobs_can_roam

Format	<BOOLEAN>
Default	**FALSE**
Description	By default, interactive jobs run from the login node that they submitted from. When `TRUE`, interactive jobs may run on login nodes other than the one where the jobs were submitted to. See **Installation Notes for Moab and TORQUE for Cray** on page 1214.

job_exclusive_on_use

Format	<BOOLEAN>
Default	FALSE

job_exclusive_on_use

Description	When job_exclusive_on_use is set to *TRUE*, pbsnodes will show job-exclusive on a node when there's at least one of its processors running a job. This differs with the default behavior which is to show job-exclusive on a node when all of its processors are running a job.
Example	`set server job_exclusive_on_use=TRUE`

job_force_cancel_time

Format	<INTEGER>
Default	Disabled
Description	If a job has been deleted and is still in the system after *x* seconds, the job will be purged from the system. This is mostly useful when a job is running on a large number of nodes and one node goes down. The job cannot be deleted because the MOM cannot be contacted. The qdel fails and none of the other nodes can be reused. This parameter can used to remedy such situations.

job_log_file_max_size

Format	<INTEGER>
Default	---
Description	This specifies a soft limit (in kilobytes) for the job log's maximum size. The file size is checked every five minutes and if the *current day* file size is greater than or equal to this value, it is rolled from `<filename>` to `<filename.1>` and a new empty log is opened. If the current day file size exceeds the maximum size a second time, the <filename.1> log file is rolled to `<filename.2>`, the current log is rolled to `<filename.1>`, and a new empty log is opened. Each new log causes all other logs to roll to an extension that is one greater than its current number. Any value less than 0 is ignored by pbs_server (meaning the log will not be rolled).

job_log_file_roll_depth

Format	<INTEGER>
Default	---

job_log_file_roll_depth	
Description	This sets the maximum number of new log files that are kept in a day if the **job_log_file_max_size** parameter is set. For example, if the roll depth is set to 3, no file can roll higher than `<filename.3>`. If a file is already at the specified depth, such as `<filename.3>`, the file is deleted so it can be replaced by the incoming file roll, `<filename.2>`.

job_log_keep_days	
Format	<INTEGER>
Default	---
Description	This maintains logs for the number of days designated. If set to 4, any log file older than 4 days old is deleted.

job_nanny	
Format	<BOOLEAN>
Default	**FALSE**
Description	When set to `TRUE`, enables the experimental "job deletion nanny" feature. All job cancels will create a repeating task that will resend KILL signals if the initial job cancel failed. Further job cancels will be rejected with the message "job cancel in progress." This is useful for temporary failures with a job's execution node during a job delete request.

job_stat_rate	
Format	<INTEGER>
Default	300 (30 in TORQUE 1.2.0p5 and earlier)
Description	If the mother superior has not sent an update by the specified time, at the specified time pbs_server requests an update on job status from the mother superior.

job_start_timeout	
Format	<INTEGER>

job_start_timeout	
Default	---
Description	Specifies the pbs_server to pbs_mom TCP socket timeout in seconds that is used when the pbs_server sends a job start to the pbs_mom. It is useful when the MOM has extra overhead involved in starting jobs. If not specified, then the **tcp_timeout** parameter is used.

job_sync_timeout	
Format	<INTEGER>
Default	60
Description	When a stray job is reported on multiple nodes, the server sends a kill signal to one node at a time. This timeout determines how long the server waits between kills if the job is still being reported on any nodes.

keep_completed	
Format	<INTEGER>
Default	--- If you ran `torque.setup` on TORQUE installation, the default is 300.
Description	The amount of time a job will be kept in the queue after it has entered the completed state. keep_completed *must* be set for job dependencies to work. For more information, see **Keeping Completed Jobs** on page 2251.

lock_file	
Format	<STRING>
Default	`torque/server_priv/server.lock`
Description	Specifies the name and location of the lock file used to determine which high availability server should be active. If a full path is specified, it is used verbatim by TORQUE. If a relative path is specified, TORQUE will prefix it with `torque/server_priv`.

lock_file_update_time	
Format	<INTEGER>
Default	3
Description	Specifies how often (in seconds) the thread will update the lock file.

lock_file_check_time	
Format	<INTEGER>
Default	9
Description	Specifies how often (in seconds) a high availability server will check to see if it should become active.

log_events	
Format	Bitmap
Default	---
Description	By default, all events are logged. However, you can customize things so that only certain events show up in the log file. These are the bitmaps for the different kinds of logs: `#define PBSEVENT_ERROR 0x0001 /* internal errors */` `#define PBSEVENT_SYSTEM 0x0002 /* system (server) events */` `#define PBSEVENT_ADMIN 0x0004 /* admin events */` `#define PBSEVENT_JOB 0x0008 /* job related events */` `#define PBSEVENT_JOB_USAGE 0x0010 /* End of Job accounting */` `#define PBSEVENT_SECURITY 0x0020 /* security violation events */` `#define PBSEVENT_SCHED 0x0040 /* scheduler events */` `#define PBSEVENT_DEBUG 0x0080 /* common debug messages */` `#define PBSEVENT_DEBUG2 0x0100 /* less needed debug messages */` `#define PBSEVENT_FORCE 0x8000 /* set to force a message */` If you want to log only error, system, and job information, use qmgr to set log_events to 11: `set server log_events = 11`

log_file_max_size	
Format	<INTEGER>
Default	0
Description	Specifies a soft limit, in kilobytes, for the server's log file. The file size is checked every 5 minutes, and if the *current day* file size is greater than or equal to this value then it will be rolled from *X* to *X.1* and a new empty log will be opened. Any value less than or equal to 0 will be ignored by pbs_server (the log will not be rolled).

log_file_roll_depth	
Format	<INTEGER>
Default	1
Description	Controls how deep the current day log files will be rolled, if log_file_max_size is set, before they are deleted.

log_keep_days	
Format	<INTEGER>
Default	0
Description	Specifies how long (in days) a server or MOM log should be kept.

log_level	
Format	<INTEGER>
Default	0
Description	Specifies the pbs_server logging verbosity. Maximum value is 7.

mail_body_fmt	
Format	A printf-like format string

mail_body_fmt	
Default	PBS Job Id: %i Job Name: %j Exec host: %h %m %d
Description	Override the default format for the body of outgoing mail messages. A number of printf-like format specifiers and escape sequences can be used: \n new line \t tab \\ backslash \' single quote \" double quote %d details concerning the message %h PBS host name %i PBS job identifier %j PBS job name %m long reason for message %r short reason for message %% a single %

mail_domain	
Format	<STRING>
Default	---
Description	Override the default domain for outgoing mail messages. If set, emails will be addressed to <user->@<hostdomain>. If unset, the job's Job_Owner attribute will be used. If set to `never`, TORQUE will never send emails.

mail_from	
Format	<STRING>
Default	adm
Description	Specify the name of the sender whenTORQUEsends emails.

mail_subject_fmt	
Format	A printf-like format string

mail_subject_fmt	
Default	PBS JOB %i
Description	Override the default format for the subject of outgoing mail messages. A number of printf-like format specifiers and escape sequences can be used: \n new line \t tab \\ backslash \' single quote \" double quote %d details concerning the message %h PBS host name %i PBS job identifier %j PBS job name %m long reason for message %r short reason for message %% a single %

managers	
Format	`<user>@<host.sub.domain>[,<user>@<host.sub.domain>...]`
Default	root@localhost
Description	List of users granted batch administrator privileges. The host, sub-domain, or domain name may be wildcarded by the use of an asterisk character (*). Requires full manager privilege to set or alter.

max_job_array_size	
Format	<INTEGER>
Default	Unlimited
Description	Sets the maximum number of jobs that can be in a single job array.

max_slot_limit	
Format	<INTEGER>

max_slot_limit

Default	Unlimited
Description	This is the maximum number of jobs that can run concurrently in any job array. Slot limits can be applied at submission time with **qsub**, or it can be modified with **qalter**. `qmgr -c 'set server max_slot_limit=10'` No array can request a slot limit greater than 10. Any array that does not request a slot limit receives a slot limit of 10. Using the example above, slot requests greater than 10 are rejected with the message: "Requested slot limit is too large, limit is 10."

max_threads

Format	<INTEGER>
Default	The value of min_threads ((2 * the number of procs listed in `/proc/cpuinfo`) + 1) * 20
Description	This is the maximum number of threads that should exist in the thread pool at any time. See **Setting min_threads and max_threads** on page 2303 for more information.

max_user_queuable

Format	<INTEGER>
Default	Unlimited
Description	When set, `max_user_queuable` places a system-wide limit on the amount of jobs that an individual user can queue. `qmgr -c 'set server max_user_queuable=500'`

min_threads

Format	<INTEGER>
Default	(2 * the number of procs listed in `/proc/cpuinfo`) + 1. If TORQUE is unable to read `/proc/cpuinfo`, the default is 10.
Description	This is the minimum number of threads that should exist in the thread pool at any time. See **Setting min_threads and max_threads** on page 2303 for more information.

moab_array_compatible	
Format	<BOOLEAN>
Default	TRUE
Description	This parameter places a hold on jobs that exceed the slot limit in a job array. When one of the active jobs is completed or deleted, one of the held jobs goes to a queued state.

mom_job_sync	
Format	<BOOLEAN>
Default	**TRUE**
Description	When set to `TRUE`, specifies that the pbs_server will synchronize its view of the job queue and resource allocation with compute nodes as they come online. If a job exists on a compute node, it will be automatically cleaned up and purged. (Enabled by default in TORQUE 2.2.0 and higher.) Jobs that are no longer reported by the mother superior are automatically purged by pbs_server. Jobs that pbs_server instructs the MOM to cancel have their processes killed in addition to being deleted (instead of leaving them running as in versions of TORQUE prior to 4.1.1).

next_job_number	
Format	<INTEGER>
Default	---
Description	Specifies the ID number of the next job. If you set your job number too low and TORQUE repeats a job number that it has already used, the job will fail. Before setting next_job_number to a number lower than any number that TORQUE has already used, you must clear out your `.e` and `.o` files. If you use Moab Workload Manager and have configured it to synchronize job IDs with TORQUE), then Moab will generate the job ID and next_job_number will have no effect on the job ID. See **Resource Manager Configuration** on page 588 for more information.

node_check_rate	
Format	<INTEGER>
Default	600

node_check_rate	
Description	Specifies the minimum duration (in seconds) that a node can fail to send a status update before being marked down by the pbs_server daemon.

node_pack	
Format	<BOOLEAN>
Default	---
Description	Controls how multiple processor nodes are allocated to jobs. If this attribute is set to `TRUE`, jobs will be assigned to the multiple processor nodes with the fewest free processors. This packs jobs into the fewest possible nodes leaving multiple processor nodes free for jobs which need many processors on a node. If set to false, jobs will be scattered across nodes reducing conflicts over memory between jobs. If unset, the jobs are packed on nodes in the order that the nodes are declared to the server (in the nodes file). Default value: unset - assigned to nodes as nodes in order that were declared.

node_ping_rate	
Format	<INTEGER>
Default	300
Description	Specifies the maximum interval (in seconds) between successive "pings" sent from the pbs_server daemon to the pbs_mom daemon to determine node/daemon health.

no_mail_force	
Format	<BOOLEAN>
Default	**FALSE**
Description	When set to `TRUE`, eliminates all e-mails when `mail_options` (see **qsub** on page 2395) is set to "n". The job owner won't receive e-mails when a job is deleted by a different user or a job failure occurs. If no_mail_force is unset or is `FALSE`, then the job owner receives e-mails when a job is deleted by a different user or a job failure occurs.

np_default	
Format	<INTEGER>
Default	---
Description	Allows the administrator to unify the number of processors (np) on all nodes. The value can be dynamically changed. A value of 0 tells pbs_server to use the value of np found in the nodes file. The maximum value is 32767.

operators	
Format	`<user>@<host.sub.domain>[,<user>@<host.sub.domain>...]`
Default	root@localhost
Description	List of users granted batch operator privileges. Requires full manager privilege to set or alter.

pass_cpuclock	
Format	<BOOLEAN>
Default	**TRUE**
Description	If set to *TRUE*, the pbs_server daemon passes the option and its value to the pbs_mom daemons for direct implementation by the daemons, making the CPU frequency adjustable as part of a resource request by a job submission. If set to *FALSE*, the pbs_server daemon creates and passes a `PBS_CPUCLOCK` job environment variable to the pbs_mom daemons that contains the value of the **cpuclock** attribute used as part of a resource request by a job submission. The CPU frequencies on the MOMs are not adjusted. The environment variable is for use by prologue and epilogue scripts, enabling administrators to log and research when users are making **cpuclock** requests, as well as researchers and developers to perform CPU clock frequency changes using a method outside of that employed by the TORQUE pbs_mom daemons.

poll_jobs	
Format	<BOOLEAN>
Default	**TRUE** (FALSE in TORQUE 1.2.0p5 and earlier)

poll_jobs	
Description	If set to `TRUE`, pbs_server will poll job info from MOMs over time and will not block on handling requests which require this job information. If set to `FALSE`, no polling will occur and if requested job information is stale, pbs_server may block while it attempts to update this information. For large systems, this value should be set to `TRUE`.

query_other_jobs	
Format	<BOOLEAN>
Default	**FALSE**
Description	When set to `TRUE`, specifies whether or not non-admin users may view jobs they do not own.

record_job_info	
Format	<BOOLEAN>
Default	**FALSE**
Description	This must be set to `TRUE` in order for job logging to be enabled.

record_job_script	
Format	<BOOLEAN>
Default	**FALSE**
Description	If set to `TRUE`, this adds the contents of the script executed by a job to the log. For record_job_script to take effect, **record_job_info** on page 2433 must be set to `TRUE`.

resources_available	
Format	<STRING>
Default	---

resources_available	
Description	Allows overriding of detected resource quantity limits (see **Assigning queue resource limits** on page 2287). pbs_server must be restarted for changes to take effect. Also, resources_available is constrained by the smallest of queue.resources_available and the server.resources_available.

scheduling	
Format	<BOOLEAN>
Default	---
Description	Allows pbs_server to be scheduled. When `FALSE`, pbs_server is a resource manager that works on its own. When `TRUE`, TORQUE allows a scheduler, such as Moab or Maui, to dictate what pbs_server should do.

submit_hosts	
Format	"<HOSTNAME>[,<HOSTNAME>]..."
Default	---
Description	Indicates which hosts included in the server nodes file located at `$TORQUE/server_priv/nodes` (see **Server Node File Configuration** on page 2222) can submit batch or interactive jobs (see **Configuring job submission hosts** on page 2216). For more information on adding hosts that are not included in the first nodes file, see the **acl_hosts** parameter.

tcp_timeout	
Format	<INTEGER>
Default	300
Description	Specifies the timeout for idle TCP connections. If no communication is received by the server on the connection after the timeout, the server closes the connection. There is an exception for connections made to the server on port 15001 (default); timeout events are ignored on the server for such connections established by a client utility or scheduler. Responsibility rests with the client to close the connection first (See **Appendix F: Large Cluster Considerations** on page 2463 for additional information.). If you use Moab Workload Manager, prevent communication errors by giving **tcp_timeout** at least twice the value of the Moab RMPOLLINTERVAL.

thread_idle_seconds	
Format	<INTEGER>
Default	300
Description	This is the number of seconds a thread can be idle in the thread pool before it is deleted. If threads should not be deleted, set to `-1`. TORQUE will always maintain at least **min_threads** number of threads, even if all are idle.

Appendix C: Node Manager (MOM) Configuration

Under TORQUE, MOM configuration is accomplished using the `mom_priv/config` file located in the PBS directory on each execution server. You must create this file and insert any desired lines in a text editor (blank lines are allowed). When you modify the `mom_priv/config` file, you must restart pbs_mom.

The following examples demonstrate two methods of modifying the `mom_priv/config` file:

```
> echo "\$loglevel 3" > /var/spool/torque/mom_priv/config
```

```
> vim /var/spool/torque/mom_priv/config
...
$loglevel 3
```

For details, see these topics:

- **Parameters** on page 2435
- **Node Features and Generic Consumable Resource Specification** on page 2453
- **Command-line Arguments** on page 2453

Related topics

- **Appendix A: Commands Overview** on page 2337
- **Appendix G: Prologue and Epilogue Scripts** on page 2469

Parameters

These parameters go in the `mom_priv/config` file. They control various behaviors for the MOMs.

arch	
Format	<STRING>
Description	Specifies the architecture of the local machine. This information is used by the scheduler only.

arch	
Example	`arch ia64`

$attempt_to_make_dir	
Format	<BOOLEAN>
Description	When set to *TRUE*, specifies that you want TORQUE to attempt to create the output directories for jobs if they do not already exist. Default is FALSE. TORQUE uses this parameter to make the directory as the *user* and not as *root*. TORQUE will create the directory (or directories) ONLY if the user has permissions to do so.
Example	`$attempt_to_make_dir true`

$clienthost	
Format	<STRING>
Description	Specifies the machine running pbs_server. This parameter is deprecated. Use **$pbsserver**.
Example	`$clienthost node01.teracluster.org`

$check_poll_time	
Format	<STRING>
Description	Amount of time between checking running jobs, polling jobs, and trying to resend obituaries for jobs that haven't sent successfully. Default is 45 seconds.
Example	`$check_poll_time 90`

$configversion	
Format	<STRING>
Description	Specifies the version of the config file data.
Example	`$configversion 113`

$cputmult	
Format	<FLOAT>
Description	CPU time multiplier. If set to 0.0, MOM level cputime enforcement is disabled.
Example	`$cputmult 2.2`

$down_on_error	
Format	<BOOLEAN>
Description	Causes the MOM to report itself as state "down" to pbs_server in the event of a failed health check. This feature is experimental. For more information, see **Parameters** on page 2435.
Example	`$down_on_error true`

$enablemomrestart	
Format	<BOOLEAN>
Description	Enables automatic restarts of the MOM. If enabled, the MOM will check if its binary has been updated and restart itself at a safe point when no jobs are running; thus making upgrades easier. The check is made by comparing the mtime of the pbs_mom executable. Command-line args, the process name, and the PATH env variable are preserved across restarts. It is recommended that this not be enabled in the config file, but enabled when desired with momctl (see **Parameters** on page 2435 for more information.)
Example	`$enablemomrestart true`

$exec_with_exec	
Format	<BOOLEAN>
Description	pbs_mom uses the exec command to start the job script rather than the TORQUE default method, which is to pass the script's contents as the input to the shell. This means that if you trap signals in the job script, they will be trapped for the job. Using the default method, you would need to configure the shell to also trap the signals. Default is FALSE.
Example	`$exec_with_exec true`

$ext_pwd_retry	
Format	<INTEGER>
Description	(*Available in TORQUE 2.5.10, 3.0.4, and later.*) Specifies the number of times to retry checking the password. Useful in cases where external password validation is used, such as with LDAP. The default value is 3 retries.
Example	`$ext_pwd_retry = 5`

$ideal_load	
Format	<FLOAT>
Description	Ideal processor load.
Example	`$ideal_load 4.0`

$igncput	
Format	<BOOLEAN>
Description	Ignores limit violation pertaining to CPU time. Default is FALSE.
Example	`$igncput true`

$ignmem	
Format	<BOOLEAN>
Description	Ignores limit violations pertaining to physical memory. Default is FALSE.
Example	`$ignmem true`

$ignvmem	
Format	<BOOLEAN>
Description	Ignores limit violations pertaining to virtual memory. Default is FALSE.
Example	`$ignvmem true`

$ignwalltime	
Format	<BOOLEAN>
Description	Ignore walltime (do not enable MOM based walltime limit enforcement).
Example	`$ignwalltime true`

$jobdirectory_sticky	
Format	<BOOLEAN>
Description	When this option is set (`true`), the job directory on the MOM can have a sticky bit set. The default is `false`.
Example	`$jobdirectory_sticky true`

$job_exit_wait_time	
Format	<INTEGER>

$job_exit_wait_time	
Description	This is the timeout to clean up parallel jobs after one of the sister nodes for the parallel job goes down or is otherwise unresponsive. The MOM sends out all of its kill job requests to sisters and marks the time. Additionally, the job is placed in the substate `JOB_SUBSTATE_EXIT_WAIT`. The MOM then periodically checks jobs in this state and if they are in this state for more than the specified time, death is assumed and the job gets cleaned up. Default is 10 minutes.
Example	`$job_exit_wait_time 300`

$job_output_file_unmask	
Format	<STRING>
Description	Uses the specified umask when creating job output and error files. Values can be specified in base 8, 10, or 16; leading 0 implies octal and leading 0x or 0X hexadecimal. A value of "userdefault" will use the user's default umask. This parameter is in version 2.3.0 and later.
Example	`$job_output_file_umask 027`

$job_starter	
Format	<STRING>
Description	Specifies the fully qualified pathname of the job starter. If this parameter is specified, instead of executing the job command and job arguments directly, the MOM will execute the job starter, passing the job command and job arguments to it as its arguments. The job starter can be used to launch jobs within a desired environment.
Example	`$job_starter /var/torque/mom_priv/job_starter.sh` `> cat /var/torque/mom_priv/job_starter.sh` `#!/bin/bash` `export FOOHOME=/home/foo` `ulimit -n 314` `$*`

$log_directory	
Format	<STRING>

$log_directory

Description	Changes the log directory. Default is `TORQUE_HOME/mom_logs/`. TORQUE_HOME default is `/var/spool/torque/` but can be changed in the `./configure` script. The value is a string and should be the full path to the desired MOM log directory.
Example	`$log_directory /opt/torque/mom_logs/`

$log_file_suffix

Format	<STRING>
Description	Optional suffix to append to log file names. If %h is the suffix, pbs_mom appends the hostname for where the log files are stored if it knows it, otherwise it will append the hostname where the MOM is running.
Example	`$log_file_suffix %h = 20100223.mybox` `$log_file_suffix foo = 20100223.foo`

$logevent

Format	<STRING>
Description	Specifies a bitmap for event types to log.
Example	`$logevent 255`

$loglevel

Format	<INTEGER>
Description	Specifies the verbosity of logging with higher numbers specifying more verbose logging. Values may range between 0 and 7.
Example	`$loglevel 4`

$log_file_max_size

Format	<INTEGER>

$log_file_max_size	
Description	Soft limit for log file size in kilobytes. Checked every 5 minutes. If the log file is found to be greater than or equal to log_file_max_size the current log file will be moved from X to X.1 and a new empty file will be opened.
Example	`$log_file_max_size = 100`

$log_file_roll_depth	
Format	<INTEGER>
Description	Specifies how many times a log fill will be rolled before it is deleted.
Example	`$log_file_roll_depth = 7`

$log_keep_days	
Format	<INTEGER>
Description	Specifies how many days to keep log files. pbs_mom deletes log files older than the specified number of days. If not specified, pbs_mom won't delete log files based on their age.
Example	`$log_keep_days 10`

$max_conn_timeout_micro_sec	
Format	<INTEGER>
Description	Specifies how long pbs_mom should wait for a connection to be made. Default value is 10000 (.1 sec).
Example	`$max_conn_timeout_micro_sec 30000` This sets the connection timeout on the MOM to .3 seconds..

$max_join_job_wait_time	
Format	<INTEGER>

$max_join_job_wait_time	
Description	The interval to wait for jobs stuck in a prerun state before deleting them from the MOMs and requeueing them on the server. Default is 10 minutes.
Example	`$max_join_job_wait_time 300`

$max_load	
Format	<FLOAT>
Description	Maximum processor load.
Example	`$max_load 4.0`

$memory_pressure_duration	
Format	<INTEGER>
Description	(*Applicable in version 3.0 and later.*) Memory pressure duration sets a limit to the number of times the value of memory_pressure_threshold can be exceeded before a process is terminated. This can only be used with **$memory_pressure_threshold**.
Example	`$memory_pressure_duration 5`

$memory_pressure_threshold	
Format	<INTEGER>
Description	(*Applicable in version 3.0 and later.*) The memory_pressure of a cpuset provides a simple per-cpuset running average of the rate that the processes in a cpuset are attempting to free up in-use memory on the nodes of the cpuset to satisfy additional memory requests. The memory_pressure_threshold is an integer number used to compare against the reclaim rate provided by the memory_pressure file. If the threshold is exceeded and memory_pressure_duration is set, then the process terminates after exceeding the threshold by the number of times set in memory_pressure_duration. If memory_pressure duration is not set, then a warning is logged and the process continues. Memory_pressure_threshold is only valid with memory_pressure enabled in the root cpuset. To enable, log in as the super user and execute the command `echo 1 >> /dev/cpuset/memory_pressure_enabled`. See the cpuset man page for more information concerning memory pressure.

$memory_pressure_threshold	
Example	`$memory_pressure_threshold 1000`

$mom_hierarchy_retry_time	
Format	<SECONDS>
Description	Specifies the amount of time that a MOM waits to retry a node in the hierarchy path after a failed connection to that node. The default is 90 seconds.
Example	`$mom_hierarchy_retry_time 30`

$mom_host	
Format	<STRING>
Description	Sets the local hostname as used by pbs_mom.
Example	`$mom_host node42`

$node_check_script	
Format	<STRING>
Description	Specifies the fully qualified pathname of the health check script to run (see **Compute Node Health Check** on page 2328 for more information).
Example	`$node_check_script /opt/batch_tools/nodecheck.pl`

$node_check_interval	
Format	<STRING>

$node_check_interval	
Description	Specifies the number of MOM intervals between subsequent executions of the specified health check. This value default to 1 indicating the check is run every MOM interval (see **Compute Node Health Check** on page 2328 for more information). $node_check_interval has two special strings that can be set: • *jobstart* – makes the node health script run when a job is started. • *jobend* – makes the node health script run after each job has completed on a node.
Example	`$node_check_interval 5`

$nodefile_suffix	
Format	<STRING>
Description	Specifies the suffix to append to a host names to denote the data channel network adapter in a multi-homed compute node.
Example	`$nodefile_suffix i` with the suffix of "i" and the control channel adapter with the name *node01*, the data channel would have a hostname of *node01i*.

$nospool_dir_list	
Format	<STRING>
Description	If this is configured, the job's output is spooled in the working directory of the job or the specified output directory. Specify the list in full paths, delimited by commas. If the job's working directory (or specified output directory) is in one of the paths in the list (or a subdirectory of one of the paths in the list), the job is spooled directly to the output location. $nospool_dir_list * is accepted. The user that submits the job must have write permission on the folder where the job is written, and read permission on the folder where the file is spooled. Alternatively, you can use the $spool_as_final_name parameter to force the job to spool directly to the final output. This should generally be used only when the job can run on the same machine as where the output file goes, or if there is a shared filesystem. If not, this parameter can slow down the system or fail to create the output file.
Example	`$nospool_dir_list /home/mike/jobs/,/var/tmp/spool/`

opsys	
Format	<STRING>
Description	Specifies the operating system of the local machine. This information is used by the scheduler only.
Example	`opsys RHEL3`

$pbsclient	
Format	<STRING>
Description	Specifies machines which the MOM daemon will trust to run resource manager commands via **momctl**. This may include machines where monitors, schedulers, or admins require the use of this command.
Example	`$pbsclient node01.teracluster.org`

$pbsserver	
Format	<STRING>
Description	Specifies the machine running pbs_server. This parameter replaces the deprecated parameter **$clienthost**.
Example	`$pbsserver node01.teracluster.org`

$prologalarm	
Format	<INTEGER>
Description	Specifies maximum duration (in seconds) which the MOM will wait for the job prologue or job epilogue to complete. The default value is 300 seconds (5 minutes). The maximum value is 300 and when set to anything higher than that, it is treated as 300.
Example	`$prologalarm 60`

$rcpcmd	
Format	<STRING>
Description	Specifies the full path and optional additional command line args to use to perform remote copies.
Example	`mom_priv/config:` `$rcpcmd /usr/local/bin/scp -i /etc/sshauth.dat`

$remote_reconfig	
Format	<STRING>
Description	Enables the ability to remotely reconfigure pbs_mom with a new config file. Default is disabled. This parameter accepts various forms of true, yes, and 1. For more information on how to reconfigure MOMs, see **momctl** `-r`.
Example	`$remote_reconfig true`

$remote_checkpoint_dirs	
Format	<STRING>
Description	Specifies which server checkpoint directories are remotely mounted. It tells the MOM which directories are shared with the server. Using remote checkpoint directories eliminates the need to copy the checkpoint files back and forth between the MOM and the server. All entries must be on the same line, separated by a space.
Example	`$remote_checkpoint_dirs /checkpointFiles /bigStorage /fast` *This informs the MOM that the* `/checkpointFiles`*,* `/bigStorage`*, and* `/fast` *directories are remotely mounted checkpoint directories.*

$reduce_prolog_checks	
Format	<STRING>
Description	If enabled, TORQUE will only check if the file is a regular file and is executable, instead of the normal checks listed on the prologue and epilogue page. Default is FALSE.

$reduce_prolog_checks	
Example	`$reduce_prolog_checks true`

$reject_job_submission	
Format	<BOOLEAN>
Description	If set to **TRUE**, jobs will be rejected and the user will receive the message, "Jobs cannot be run on mom %s." Default is FALSE.
Example	`$reject_job_submission job01`

$resend_join_job_wait_time	
Format	<INTEGER>
Description	This is the timeout for the Mother Superior to re-send the join job request if it didn't get a reply from all the sister MOMs. The resend happens only once. Default is 5 minutes.
Example	`$resend_join_job_wait_time 120`

$restricted	
Format	<STRING>
Description	Specifies hosts which can be trusted to access MOM services as non-root. By default, no hosts are trusted to access MOM services as non-root.
Example	`$restricted *.teracluster.org`

$rpp_throttle	
Format	<INTEGER>
Description	This integer is in microseconds and causes a sleep after every RPP packet is sent. It is for systems that experience job failures because of incomplete data.
Example	`$rpp_throttle 100` (will cause a 100 microsecond sleep)

size[fs=<FS>]	
Format	N/A
Description	Specifies that the available and configured disk space in the <FS> filesystem is to be reported to the pbs_server and scheduler. To request disk space on a per job basis, specify the file resource as in `qsub -l nodes=1,file=1000kb`. Unlike most MOM config options, the *size* parameter is not preceded by a "$" character.
Example	`size[fs=/localscratch]` The available and configured disk space in the `/localscratch` filesystem will be reported.

$source_login_batch	
Format	<STRING>
Description	Specifies whether or not MOM will source the `/etc/profile`, etc. type files for *batch* jobs. Parameter accepts various forms of true, false, yes, no, 1 and 0. Default is TRUE. This parameter is in version 2.3.1 and later.
Example	`$source_login_batch False` MOM will bypass the sourcing of `/etc/profile`, etc. type files.

$source_login_interactive	
Format	<STRING>
Description	Specifies whether or not MOM will source the `/etc/profile`, etc. type files for *interactive* jobs. Parameter accepts various forms of true, false, yes, no, 1 and 0. Default is TRUE. This parameter is in version 2.3.1 and later.
Example	`$source_login_interactive False` MOM will bypass the sourcing of `/etc/profile`, etc. type files.

$spool_as_final_name	
Format	<BOOLEAN>

$spool_as_final_name	
Description	This makes the job write directly to its output destination instead of a spool directory. This allows users easier access to the file if they want to watch the jobs output as it runs.
Example	`$spool_as_final_name true`

$status_update_time	
Format	<INTEGER>
Description	Specifies the number of seconds between subsequent MOM-to-server update reports. Default is 45 seconds.
Example	status_update_time: `$status_update_time 120` MOM will send server update reports every 120 seconds.

$thread_unlink_calls	
Format	<BOOLEAN>
Description	Threads calls to unlink when deleting a job. Default is false. If it is set to TRUE, pbs_mom will use a thread to delete the job's files.
Example	thread_unlink_calls: `$thread_unlink_calls true`

$timeout	
Format	<INTEGER>
Description	Specifies the number of seconds before a TCP connection on the MOM will timeout. Default is 300 seconds. In version 3.x and earlier, this specifies the number of seconds before MOM-to-MOM messages will timeout if RPP is disabled. Default is 60 seconds.
Example	`$timeout 120` A TCP connection will wait up to 120 seconds before timing out. For 3.x and earlier, MOM-to-MOM communication will allow up to 120 seconds before timing out.

$tmpdir	
Format	<STRING>
Description	Specifies a directory to create job-specific scratch space (see Creating Per-Job Temporary Directories).
Example	`$tmpdir /localscratch`

$usecp	
Format	<HOST>:<SRCDIR> <DSTDIR>
Description	Specifies which directories should be staged (see **NFS and Other Networked Filesystems** on page 2307)
Example	`$usecp *.fte.com:/data /usr/local/data`

$use_smt	
Format	<BOOLEAN>
Description	Indicates that the user would like to use SMT. If set, each logical core inside of a physical core will be used as a normal core for cpusets. This parameter is on by default. If SMT is used, you will need to set the *np* attribute so that each logical processor is counted.
Example	`$use_smt false`

$varattr	
Format	<INTEGER> <STRING>

$varattr	
Description	Provides a way to keep track of dynamic attributes on nodes. <INTEGER> is how many seconds should go by between calls to the script to update the dynamic values. If set to -1, the script is read only one time. <STRING> is the script path. This script should check for whatever dynamic attributes are desired, and then output lines in this format: `name=value` Include any arguments after the script's full path. These features are visible in the output of **pbsnodes** `-a` varattr=Matlab=7.1;Octave=1.0. For information about using $varattr to request dynamic features in Moab, see **Resource Manager Extensions** on page 618.
Example	`$varattr 25 /usr/local/scripts/nodeProperties.pl arg1 arg2 arg3`

$wallmult	
Format	<FLOAT>
Description	Sets a factor to adjust walltime usage by multiplying a default job time to a common reference system. It modifies real walltime on a per-MOM basis (MOM configuration parameters). The factor is used for walltime calculations and limits in the same way that cputmult is used for cpu time. If set to 0.0, MOM level walltime enforcement is disabled.
Example	`$wallmult 2.2`

$xauthpath	
Format	<STRING>
Description	Specifies the path to the xauth binary to enable X11 forwarding.
Example	`$xauthpath /opt/bin/xauth/`

Related topics

- **Appendix C: Node Manager (MOM) Configuration** on page 2435

Node Features and Generic Consumable Resource Specification

Node features (a.k.a. "node properties") are opaque labels which can be applied to a node. They are not consumable and cannot be associated with a value. (Use generic resources described below for these purposes). Node features are configured within the nodes file on the **pbs_server** head node. This file can be used to specify an arbitrary number of node features.

Additionally, per node consumable generic resources may be specified using the format "<ATTR> <VAL>" with no leading dollar ("$") character. When specified, this information is routed to the scheduler and can be used in scheduling decisions. For example, to indicate that a given host has two tape drives and one node-locked matlab license available for batch jobs, the following could be specified:

`mom_priv/config:`

```
$clienthost 241.13.153.7
tape 2
matlab 1
```

Dynamic consumable resource information can be routed in by specifying a path preceded by an exclamation point. (!) as in the example below. If the resource value is configured in this manner, the specified file will be periodically executed to load the effective resource value.

`mom_priv/config:`

```
$clienthost 241.13.153.7
tape !/opt/rm/gettapecount.pl
matlab !/opt/tools/getlicensecount.pl
```

Related topics

- **Appendix C: Node Manager (MOM) Configuration** on page 2435

Command-line Arguments

Below is a table of pbs_mom command-line startup flags.

Flag	Description
a <integer>	Alarm time in seconds.
c <file>	Config file path.
C <dir-ectory>	Checkpoint path.
d <dir-ectory>	Home directory.
L <file>	Log file.

Flag	Description
M <integer>	MOM port to listen on.
p	Perform 'poll' based job recovery on restart (jobs persist until associated processes terminate).
P	On restart, deletes all jobs that were running on MOM (Available in 2.4.X and later).
q	On restart, requeues all jobs that were running on MOM (Available in 2.4.X and later).
r	On restart, kills all processes associated with jobs that were running on MOM, and then requeues the jobs.
R <integer>	MOM 'RM' port to listen on.
S <integer>	pbs_server port to connect to.
v	Display version information and exit.
x	Disable use of privileged port.
?	Show usage information and exit.

For more details on these command-line options, see **pbs_mom** on page 2344.

Related topics

- **Appendix C: Node Manager (MOM) Configuration** on page 2435

Appendix D: Diagnostics and Error Codes

TORQUE has a diagnostic script to assist you in giving TORQUE Support the files they need to support issues. It should be run by a user that has access to run all TORQUE commands and access to all TORQUE directories (this is usually root).

The script (`contrib/diag/tdiag.sh`) is available in TORQUE 2.3.8, TORQUE 2.4.3, and later. The script grabs the node file, server and MOM log files, and captures the output of `qmgr -c 'p s'`. These are put in a tar file.

The script also has the following options (this can be shown in the command line by entering `./tdiag.sh -h`):

```
USAGE: ./torque_diag [-d DATE] [-h] [-o OUTPUT_FILE] [-t TORQUE_HOME]
```

- *DATE* should be in the format YYYYmmdd. For example, " 20091130" would be the date for November 30th, 2009. If no date is specified, today's date is used.
- *OUTPUT_FILE* is the optional name of the output file. The default output file is `torque_diag<today's_date>.tar.gz`. TORQUE_HOME should be the path to your TORQUE directory. If no directory is specified, `/var/spool/torque` is the default.

Table 4-5: TORQUE error codes

Error code name	Number	Description
PBSE_FLOOR	15000	No error
PBSE_UNKJOBID	15001	Unknown job ID error
PBSE_NOATTR	15002	Undefined attribute
PBSE_ATTRRO	15003	Cannot set attribute, read only or insufficient permission
PBSE_IVALREQ	15004	Invalid request
PBSE_UNKREQ	15005	Unknown request
PBSE_TOOMANY	15006	Too many submit retries
PBSE_PERM	15007	Unauthorized Request
PBSE_IFF_NOT_FOUND	15008	trqauthd unable to authenticate
PBSE_MUNGE_NOT_FOUND	15009	Munge executable not found, unable to authenticate
PBSE_BADHOST	15010	Access from host not allowed, or unknown host
PBSE_JOBEXIST	15011	Job with requested ID already exists
PBSE_SYSTEM	15012	System error
PBSE_INTERNAL	15013	PBS server internal error
PBSE_REGROUTE	15014	Dependent parent job currently in routing queue
PBSE_UNKSIG	15015	Unknown/illegal signal name

Error code name	Number	Description
PBSE_BADATVAL	15016	Illegal attribute or resource value for
PBSE_MODATRRUN	15017	Cannot modify attribute while job running
PBSE_BADSTATE	15018	Request invalid for state of job
PBSE_UNKQUE	15020	Unknown queue
PBSE_BADCRED	15021	Invalid credential
PBSE_EXPIRED	15022	Expired credential
PBSE_QUNOENB	15023	Queue is not enabled
PBSE_QACESS	15024	Access to queue is denied
PBSE_BADUSER	15025	Bad UID for job execution
PBSE_HOPCOUNT	15026	Job routing over too many hops
PBSE_QUEEXIST	15027	Queue already exists
PBSE_ATTRTYPE	15028	Incompatible type
PBSE_QUEBUSY	15029	Cannot delete busy queue
PBSE_QUENBIG	15030	Queue name too long
PBSE_NOSUP	15031	No support for requested service
PBSE_QUENOEN	15032	Cannot enable queue, incomplete definition
PBSE_PROTOCOL	15033	Batch protocol error
PBSE_BADATLST	15034	Bad attribute list structure
PBSE_NOCONNECTS	15035	No free connections
PBSE_NOSERVER	15036	No server specified

Error code name	Number	Description
PBSE_UNKRESC	15037	Unknown resource type
PBSE_EXCQRESC	15038	Job exceeds queue resource limits
PBSE_QUENODFLT	15039	No default queue specified
PBSE_NORERUN	15040	Job is not rerunnable
PBSE_ROUTEREJ	15041	Job rejected by all possible destinations (check syntax, queue resources, ...)
PBSE_ROUTEEXPD	15042	Time in Route Queue Expired
PBSE_MOMREJECT	15043	Execution server rejected request
PBSE_BADSCRIPT	15044	(qsub) cannot access script file
PBSE_STAGEIN	15045	Stage-in of files failed
PBSE_RESCUNAV	15046	Resource temporarily unavailable
PBSE_BADGRP	15047	Bad GID for job execution
PBSE_MAXQUED	15048	Maximum number of jobs already in queue
PBSE_CKPBSY	15049	Checkpoint busy, may retry
PBSE_EXLIMIT	15050	Resource limit exceeds allowable
PBSE_BADACCT	15051	Invalid Account
PBSE_ALRDYEXIT	15052	Job already in exit state
PBSE_NOCOPYFILE	15053	Job files not copied
PBSE_CLEANEDOUT	15054	Unknown job id after clean init
PBSE_NOSYNCMSTR	15055	No master found for sync job set

Error code name	Number	Description
PBSE_BADDEPEND	15056	Invalid Job Dependency
PBSE_DUPLIST	15057	Duplicate entry in list
PBSE_DISPROTO	15058	Bad DIS based Request Protocol
PBSE_EXECTHERE	15059	Cannot execute at specified host because of checkpoint or stagein files
PBSE_SISREJECT	15060	Sister rejected
PBSE_SISCOMM	15061	Sister could not communicate
PBSE_SVRDOWN	15062	Request not allowed: Server shutting down
PBSE_CKPSHORT	15063	Not all tasks could checkpoint
PBSE_UNKNODE	15064	Unknown node
PBSE_UNKNODEATR	15065	Unknown node-attribute
PBSE_NONODES	15066	Server has no node list
PBSE_NODENBIG	15067	Node name is too big
PBSE_NODEEXIST	15068	Node name already exists
PBSE_BADNDATVAL	15069	Illegal value for
PBSE_MUTUALEX	15070	Mutually exclusive values for
PBSE_GMODERR	15071	Modification failed for
PBSE_NORELYMOM	15072	Server could not connect to MOM
PBSE_NOTSNODE	15073	No time-share node available
PBSE_JOBTYPE	15074	Wrong job type

Error code name	Number	Description
PBSE_BADACLHOST	15075	Bad ACL entry in host list
PBSE_MAXUSERQUED	15076	Maximum number of jobs already in queue for user
PBSE_BADDISALLOWTYPE	15077	Bad type in disallowed_types list
PBSE_NOINTERACTIVE	15078	Queue does not allow interactive jobs
PBSE_NOBATCH	15079	Queue does not allow batch jobs
PBSE_NORERUNABLE	15080	Queue does not allow rerunable jobs
PBSE_NONONRERUNABLE	15081	Queue does not allow nonrerunable jobs
PBSE_UNKARRAYID	15082	Unknown Array ID
PBSE_BAD_ARRAY_REQ	15083	Bad Job Array Request
PBSE_BAD_ARRAY_DATA	15084	Bad data reading job array from file
PBSE_TIMEOUT	15085	Time out
PBSE_JOBNOTFOUND	15086	Job not found
PBSE_NOFAULTTOLERANT	15087	Queue does not allow fault tolerant jobs
PBSE_NOFAULTINTOLERANT	15088	Queue does not allow fault intolerant jobs
PBSE_NOJOBARRAYS	15089	Queue does not allow job arrays
PBSE_RELAYED_TO_MOM	15090	Request was relayed to a MOM
PBSE_MEM_MALLOC	15091	Error allocating memory - out of memory
PBSE_MUTEX	15092	Error allocating controling mutex (lock/unlock)
PBSE_THREADATTR	15093	Error setting thread attributes
PBSE_THREAD	15094	Error creating thread

Error code name	Number	Description
PBSE_SELECT	15095	Error in socket select
PBSE_SOCKET_FAULT	15096	Unable to get connection to socket
PBSE_SOCKET_WRITE	15097	Error writing data to socket
PBSE_SOCKET_READ	15098	Error reading data from socket
PBSE_SOCKET_CLOSE	15099	Socket close detected
PBSE_SOCKET_LISTEN	15100	Error listening on socket
PBSE_AUTH_INVALID	15101	Invalid auth type in request
PBSE_NOT_IMPLEMENTED	15102	This functionality is not yet implemented
PBSE_QUENOTAVAILABLE	15103	Queue is currently not available
PBSE_TMPDIFFOWNER	15104	tmpdir owned by another user
PBSE_TMPNOTDIR	15105	tmpdir exists but is not a directory
PBSE_TMPNONAME	15106	tmpdir cannot be named for job
PBSE_CANTOPENSOCKET	15107	Cannot open demux sockets
PBSE_CANTCONTACTSISTERS	15108	Cannot send join job to all sisters
PBSE_CANTCREATETMPDIR	15109	Cannot create tmpdir for job
PBSE_BADMOMSTATE	15110	Mom is down, cannot run job
PBSE_SOCKET_INFORMATION	15111	Socket information is not accessible
PBSE_SOCKET_DATA	15112	Data on socket does not process correctly
PBSE_CLIENT_INVALID	15113	Client is not allowed/trusted
PBSE_PREMATURE_EOF	15114	Premature End of File

Error code name	Number	Description
PBSE_CAN_NOT_SAVE_FILE	15115	Error saving file
PBSE_CAN_NOT_OPEN_FILE	15116	Error opening file
PBSE_CAN_NOT_WRITE_FILE	15117	Error writing file
PBSE_JOB_FILE_CORRUPT	15118	Job file corrupt
PBSE_JOB_RERUN	15119	Job can not be rerun
PBSE_CONNECT	15120	Can not establish connection
PBSE_JOBWORKDELAY	15121	Job function must be temporarily delayed
PBSE_BAD_PARAMETER	15122	Parameter of function was invalid
PBSE_CONTINUE	15123	Continue processing on job. (Not an error)
PBSE_JOBSUBSTATE	15124	Current sub state does not allow trasaction.
PBSE_CAN_NOT_MOVE_FILE	15125	Error moving file
PBSE_JOB_RECYCLED	15126	Job is being recycled
PBSE_JOB_ALREADY_IN_QUEUE	15127	Job is already in destination queue.
PBSE_INVALID_MUTEX	15128	Mutex is NULL or otherwise invalid
PBSE_MUTEX_ALREADY_LOCKED	15129	The mutex is already locked by this object
PBSE_MUTEX_ALREADY_UNLOCKED	15130	The mutex has already been unlocked by this object
PBSE_INVALID_SYNTAX	15131	Command syntax invalid
PBSE_NODE_DOWN	15132	A node is down. Check the MOM and host

Error code name	Number	Description
PBSE_SERVER_NOT_FOUND	15133	Could not connect to batch server
PBSE_SERVER_BUSY	15134	Server busy. Currently no available threads

Appendix E: Considerations before Upgrading

TORQUE is flexible in regards to how it can be upgraded. In most cases, a TORQUE "shutdown" followed by a *configure*, *make*, *make install* procedure as documented in this guide is all that is required. See **Installing TORQUE** on page 2193 for more information. This process will preserve existing configuration and in most cases, existing workload.

A few considerations are included below:

- If upgrading from OpenPBS, PBSPro, or TORQUE 1.0.3 or earlier, queued jobs whether active or idle will be lost. In such situations, job queues should be completely drained of all jobs.
- If not using the **pbs_mom** `-r` or `-p` flag (see **Command-line Arguments** on page 2453), running jobs may be lost. In such cases, running jobs should be allowed to be completed or should be requeued before upgrading TORQUE.
- pbs_mom and pbs_server daemons of differing versions may be run together. However, not all combinations have been tested and unexpected failures may occur.
- When upgrading from early versions of TORQUE (pre-4.0) and Moab, you may encounter a problem where Moab core files are regularly created in `/opt/moab`. This can be caused by old TORQUE library files used by Moab that try to authorize with the old TORQUE pbs_iff authorization daemon. You can resolve the problem by removing the old version library files from `/usr/local/lib`.

To upgrade

1. Build new release (do not install).
2. Stop all TORQUE daemons (see **qterm** and **momctl** `-s`).
3. Install new TORQUE (use *make install*).
4. Start all TORQUE daemons.

Rolling upgrade

If you are upgrading to a new point release of your current version (for example, from 4.2.2 to 4.2.3) and not to a new major release from your current version (for example, from 4.1 to 4.2), you can use the following procedure to upgrade TORQUE without taking your nodes offline.

> Because TORQUE version 4.1.4 changed the way that pbs_server communicates with the MOMs, it is not recommended that you perform a rolling upgrade of TORQUE from version 4.1.3 to 4.1.4.

To perform a rolling upgrade in TORQUE

1. Enable the **pbs_mom** on page 2344 flag on the MOMs you want to upgrade. The `enablemomrestart` option causes a MOM to check if its binary has been updated and restart itself at a safe point when no jobs are running. You can enable this in the MOM configuration file, but it is recommended that you use momctl instead.

   ```
   > momctl -q enablemomrestart=1 -h :ALL
   ```

   The `enablemomrestart` flag is enabled on all nodes.

2. Replace the pbs_mom binary, located in `/usr/local/bin` by default. pbs_mom will continue to run uninterrupted because the pbs_mom binary has already been loaded in RAM.

   ```
   > torque-package-mom-linux-x86_64.sh --install
   ```

   The next time pbs_mom is in an idle state, it will check for changes in the binary. If pbs_mom detects that the binary on disk has changed, it will restart automatically, causing the new pbs_mom version to load.

   After the pbs_mom restarts on each node, the `enablemomrestart` parameter will be set back to false (`0`) for that node.

> If you have cluster with high utilization, you may find that the nodes never enter an idle state so pbs_mom never restarts. When this occurs, you must manually take the nodes offline and wait for the running jobs to complete before restarting pbs_mom. To set the node to an offline state, which will allow running jobs to complete but will not allow any new jobs to be scheduled on that node, use `pbsnodes -o` *<nodeName>*. After the new MOM has started, you must make the node active again by running `pbsnodes -c` *<nodeName>*.

Appendix F: Large Cluster Considerations

TORQUE has enhanced much of the communication found in the original OpenPBS project. This has resulted in a number of key advantages including support for:

- larger clusters.
- more jobs.
- larger jobs.
- larger messages.

In most cases, enhancements made apply to all systems and no tuning is required. However, some changes have been made configurable to allow site specific modification. The configurable communication parameters are: **node_check_rate**, **node_ping_rate**, and **tcp_timeout**.

For details, see these topics:

Scalability Guidelines

In very large clusters (in excess of 1,000 nodes), it may be advisable to tune a number of communication layer timeouts. By default, PBS MOM daemons timeout on inter-MOM messages after 60 seconds. In TORQUE 1.1.0p5 and higher, this can be adjusted by setting the timeout parameter in the `mom_priv/config` file (see **Appendix C: Node Manager (MOM) Configuration** on page 2435). If 15059 errors (cannot receive message from sisters) are seen in the MOM logs, it may be necessary to increase this value.

Client-to-server communication timeouts are specified via the **tcp_timeout** server option using the **qmgr** command.

> On some systems, *ulimit* values may prevent large jobs from running. In particular, the open file descriptor limit (i.e., `ulimit -n`) should be set to at least the maximum job size in procs + 20. Further, there may be value in setting the `fs.file-max` in `sysctl.conf` to a high value, such as:
>
> ```
> /etc/sysctl.conf:
> fs.file-max = 65536
> ```

Related topics

- **Appendix F: Large Cluster Considerations** on page 2463

End-user Command Caching

qstat

In a large system, users may tend to place excessive load on the system by manual or automated use of resource manager end user client commands. A simple way of reducing this load is through the use of client command wrappers which cache data. The example script below will cache the output of the command '**qstat** `-f`' for 60 seconds and report this info to end users.

```
#!/bin/sh

# USAGE: qstat $@

CMDPATH=/usr/local/bin/qstat
CACHETIME=60
TMPFILE=/tmp/qstat.f.tmp

if [ "$1" != "-f" ] ; then
  #echo "direct check (arg1=$1) "
  $CMDPATH $1 $2 $3 $4
  exit $?
fi

if [ -n "$2" ] ; then
  #echo "direct check (arg2=$2)"
  $CMDPATH $1 $2 $3 $4
  exit $?
fi

if [ -f $TMPFILE ] ; then
  TMPFILEMTIME=`stat -c %Z $TMPFILE`
else
  TMPFILEMTIME=0
fi

NOW=`date +%s`

AGE=$(($NOW - $TMPFILEMTIME))

#echo AGE=$AGE

for i in 1 2 3;do
  if [ "$AGE" -gt $CACHETIME ] ; then
    #echo "cache is stale "

    if [ -f $TMPFILE.1 ] ; then
      #echo someone else is updating cache

      sleep 5

      NOW=`date +%s`

      TMPFILEMTIME=`stat -c %Z $TMPFILE`

AGE=$(($NOW - $TMPFILEMTIME))
    else
      break;
    fi
  fi
done

if [ -f $TMPFILE.1 ] ; then
  #echo someone else is hung

  rm $TMPFILE.1
fi

if [ "$AGE" -gt $CACHETIME ] ; then
  #echo updating cache

  $CMDPATH -f > $TMPFILE.1

mv $TMPFILE.1 $TMPFILE

fi

#echo "using cache"
```

```
cat $TMPFILE

exit 0
```

The above script can easily be modified to cache any command and any combination of arguments by changing one or more of the following attributes:

- script name
- value of $CMDPATH
- value of $CACHETIME
- value of $TMPFILE

For example, to cache the command **pbsnodes** `-a`, make the following changes:

- Move original pbsnodes command to `pbsnodes.orig`.
- Save the script as '`pbsnodes`'.
- Change $CMDPATH to `pbsnodes.orig`.
- Change $TMPFILE to `/tmp/pbsnodes.a.tmp`.

Related topics

- **Appendix F: Large Cluster Considerations** on page 2463

Moab and TORQUE Configuration for Large Clusters

There are a few basic configurations for Moab and TORQUE that can potentially improve performance on large clusters.

Moab configuration

In the `moab.cfg` file, add:

1. `RMPOLLINTERVAL 30,30` - This sets the minimum and maximum poll interval to 30 seconds.
2. `RMCFG[<name>] FLAGS=ASYNCSTART` - This tells Moab not to block until it receives a confirmation that the job starts.
3. `RMCFG[<name>] FLAGS=ASYNCDELETE` - This tells Moab not to block until it receives a confirmation that the job was deleted.

TORQUE configuration

1. Follow the Starting TORQUE in large environments recommendations.
2. Increase `job_start_timeout` on pbs_server. The default is *300* (5 minutes), but for large clusters the value should be changed to something like *600* (10 minutes). Sites running very large parallel jobs might want to set this value even higher.
3. Use a node health check script on all MOM nodes. This helps prevent jobs from being scheduled on bad nodes and is especially helpful for large parallel jobs.

4. Make sure that `ulimit -n` (maximum file descriptors) is set to *unlimited*, or a very large number, and not the default.
5. For clusters with a high job throughput it is recommended that the server parameter `max_threads` be increased from the default. The default is (2 * number of cores + 1) * 10.

Related topics

- **Appendix F: Large Cluster Considerations** on page 2463

Starting TORQUE in Large Environments

If running TORQUE in a large environment, use these tips to help TORQUE start up faster.

Fastest possible start up

1. Create a MOM hierarchy, even if your environment has a one-level MOM hierarchy (meaning all MOMs report directly to pbs_server), and copy the file to the `mom_priv` directory on the MOMs.
2. Start pbs_server with the -n option. This specifies that pbs_server won't send the hierarchy to the MOMs unless a MOM requests it.
3. Start the MOMs normally.

If no daemons are running

1. Start pbs_server with the -c option.
2. Start the MOMs without the `-w` option.

If MOMs are running and just restarting pbs_server

1. Start pbs_server without the `-c` option.

If restarting a MOM or all MOMs

1. Start pbs_server without the `-w` option. Starting it with `-w` causes the MOMs to appear to be down.

Related topics

- **Appendix F: Large Cluster Considerations** on page 2463

Other Considerations

job_stat_rate

In a large system, there may be many users, many jobs, and many requests for information. To speed up response time for users and for programs using the API the **job_stat_rate** can be used to tweak when the pbs_server daemon will query MOMs for job information. By increasing this number, a system will not be constantly querying job information and causing other commands to block.

poll_jobs

The **poll_jobs** parameter allows a site to configure how the pbs_server daemon will poll for job information. When set to TRUE, the pbs_server will poll job information in the background and not block on user requests. When set to FALSE, the pbs_server may block on user requests when it has stale job information data. Large clusters should set this parameter to TRUE.

Internal settings

On large, slow, and/or heavily loaded systems, it may be desirable to increase the `pbs_tcp_timeout` setting used by the pbs_mom daemon in MOM-to-MOM communication. This setting defaults to 20 seconds and requires rebuilding code to adjust. For client-server based communication, this attribute can be set using the **qmgr** command. For MOM-to-MOM communication, a source code modification is required. To make this change, edit the `$TORQUEBUILDDIR/src/lib/Libifl/tcp_dis.c` file and set `pbs_tcp_timeout` to the desired maximum number of seconds allowed for a MOM-to-MOM request to be serviced.

> A system may be heavily loaded if it reports multiple 'End of File from addr' or 'Premature end of message' failures in the pbs_mom or pbs_server logs.

Scheduler settings

If using Moab, there are a number of parameters which can be set on the scheduler which may improve TORQUE performance. In an environment containing a large number of short-running jobs, the JOBAGGREGATIONTIME parameter (see **Appendix A: Moab Parameters** on page 902) can be set to reduce the number of workload and resource queries performed by the scheduler when an event based interface is enabled. If the pbs_server daemon is heavily loaded and PBS API timeout errors (i.e. "Premature end of message") are reported within the scheduler, the "TIMEOUT" attribute of the RMCFG parameter may be set with a value of between 30 and 90 seconds.

File system

TORQUE can be configured to disable file system blocking until data is physically written to the disk by using the `--disable-filesync` argument with *configure*. While having filesync enabled is more reliable, it may lead to server delays for sites with either a larger number of nodes, or a large number of jobs. Filesync is enabled by default.

Network ARP cache

For networks with more than 512 nodes it is mandatory to increase the kernel's internal ARP cache size. For a network of ~1000 nodes, we use these values in `/etc/sysctl.conf` on all nodes and servers:

```
/etc/sysctl.conf

# Don't allow the arp table to become bigger than this
net.ipv4.neigh.default.gc_thresh3 = 4096
# Tell the gc when to become aggressive with arp table cleaning.
# Adjust this based on size of the LAN.
net.ipv4.neigh.default.gc_thresh2 = 2048
# Adjust where the gc will leave arp table alone
net.ipv4.neigh.default.gc_thresh1 = 1024
# Adjust to arp table gc to clean-up more often
net.ipv4.neigh.default.gc_interval = 3600
# ARP cache entry timeout
net.ipv4.neigh.default.gc_stale_time = 3600
```

Use `sysctl -p` to reload this file.

The ARP cache size on other Unixes can presumably be modified in a similar way.

An alternative approach is to have a static `/etc/ethers` file with all hostnames and MAC addresses and load this by `arp -f /etc/ethers`. However, maintaining this approach is quite cumbersome when nodes get new MAC addresses (due to repairs, for example).

Related topics

- **Appendix F: Large Cluster Considerations** on page 2463

Appendix G: Prologue and Epilogue Scripts

TORQUE provides administrators the ability to run scripts before and/or after each job executes. With such a script, a site can prepare systems, perform node health checks, prepend and append text to output and error log files, cleanup systems, and so forth.

The following table shows which MOM runs which script. All scripts must be in the `TORQUE_HOME/mom_priv/` directory and be available on every compute node. The "Mother Superior" is the pbs_mom on the first node allocated for a job. While it is technically a sister node, it is not a "Sister" for the purposes of the following table.

> The execution directory for each script is `TORQUE_HOME/mom_priv/`.

Script	Execution location	Execute as	File permissions
prologue	Mother Superior	root	Readable and executable by root and NOT writable by anyone but root (e.g., `-r-x-----`)
epilogue		root	
prologue.user		user	Readable and executable by root and other (e.g., `-r-x---r-x`)
epilogue.user		user	

Script	Execution location	Execute as	File permissions
prologue.parallel	Sister	root	Readable and executable by root and NOT writable by anyone but root (e.g., `-r-x-----`)
epilogue.parallel		root	
prologue.user.parallel		user	Readable and executable by root and other (e.g., `-r-x---r-x`)
epilogue.user.parallel		user	
epilogue.precancel	Mother Superior This script runs after a job cancel request is received from pbs_server and before a kill signal is sent to the job process.	user	Readable and executable by root and other (e.g., `-r-x---r-x`)

`epilogue.parallel` is available in version 2.1 and later.

This section contains these topics:

Script Order of Execution

When jobs start, the order of script execution is `prologue` followed by `prologue.user`. On job exit, the order of execution is `epilogue.user` followed by `epilogue` unless a job is canceled. In that case, `epilogue.precancel` is executed first. `epilogue.parallel` is executed only on the Sister nodes when the job is completed.

The `epilogue` and `prologue` scripts are controlled by the system administrator. However, beginning in TORQUE version 2.4 a user `epilogue` and `prologue` script can be used on a per job basis. (See **Per Job Prologue and Epilogue Scripts** on page 2472 for more information.)

Root squashing is now supported for `epilogue` and `prologue` scripts.

Related topics

- **Appendix G: Prologue and Epilogue Scripts** on page 2469

Script Environment

The `prologue` and `epilogue` scripts can be very simple. On most systems, the script must declare the execution shell using the `#!<SHELL>` syntax (for example, "`#!/bin/sh`"). In addition, the script may want to process context sensitive arguments passed by TORQUE to the script.

Prologue Environment

The following arguments are passed to the `prologue`, `prologue.user`, and `prologue.parallel` scripts:

Argument	Description
argv[1]	job id
argv[2]	job execution user name
argv[3]	job execution group name
argv[4]	job name (TORQUE 1.2.0p4 and higher only)
argv[5]	list of requested resource limits (TORQUE 1.2.0p4 and higher only)
argv[6]	job execution queue (TORQUE 1.2.0p4 and higher only)
argv[7]	job account (TORQUE 1.2.0p4 and higher only)

Epilogue Environment

TORQUE supplies the following arguments to the `epilogue`, `epilogue.user`, `epilogue.precancel`, and `epilogue.parallel` scripts:

Argument	Description
argv[1]	job id
argv[2]	job execution user name
argv[3]	job execution group name
argv[4]	job name

Argument	Description
argv[5]	session id
argv[6]	list of requested resource limits
argv[7]	list of resources used by job
argv[8]	job execution queue
argv[9]	job account
argv[10]	job exit code

The `epilogue.precancel` script is run after a job cancel request is received by the MOM and before any signals are sent to job processes. If this script exists, it is run whether the canceled job was active or idle.

> The cancel job command (**qdel**) will take as long to return as the `epilogue.precancel` script takes to run. For example, if the script runs for 5 minutes, it takes 5 minutes for qdel to return.

For all scripts, the environment passed to the script is empty. However, if you submit the job using msub rather than qsub, some Moab environment variables are available in the TORQUE prologue and epilogue script environment: MOAB_CLASS, MOAB_GROUP, MOAB_JOBARRAYINDEX, MOAB_JOBARRAYRANGE, MOAB_JOBID, MOAB_JOBNAME, MOAB_MACHINE, MOAB_NODECOUNT, MOAB_NODELIST, MOAB_PARTITION, MOAB_PROCCOUNT, MOAB_QOS, MOAB_TASKMAP, and MOAB_USER. See **msub** on page 290 for more information.

Also, standard input for both scripts is connected to a system dependent file. Currently, for all systems this is `/dev/null`. Except for epilogue scripts of an interactive job, `prologue.parallel`, `epilogue.precancel`, and `epilogue.parallel`, the standard output and error are connected to output and error files associated with the job. For an interactive job, since the pseudo terminal connection is released after the job completes, the standard input and error point to `/dev/null`. For `prologue.parallel` and `epilogue.parallel`, the user will need to redirect `stdout` and `stderr` manually.

Related topics

- **Appendix G: Prologue and Epilogue Scripts** on page 2469

Per Job Prologue and Epilogue Scripts

TORQUE supports per job prologue and epilogue scripts when using the **qsub** `-l` option. The syntax is:

```
qsub -l prologue=<prologue_script_path> epilogue=<epilogue_script_path>
<script>.
```

The path can be either relative (from the directory where the job is submitted) or absolute. The files must be owned by the user with at least execute and read privileges, and the permissions must not be writeable by group or other.

/home/usertom/dev/

```
-r-x------ 1 usertom usertom 24 2009-11-09 16:11 prologue_script.sh
-r-x------ 1 usertom usertom 24 2009-11-09 16:11 epilogue_script.sh
```

Example 4-29:

```
$ qsub -l prologue=/home/usertom/dev/prologue_
script.sh,epilogue=/home/usertom/dev/epilogue_script.sh job14.pl
```

This job submission executes the `prologue` script first. When the `prologue` script is complete, `job14.pl` runs. When `job14.pl` completes, the `epilogue` script is executed.

Related topics

- **Appendix G: Prologue and Epilogue Scripts** on page 2469

Prologue and Epilogue Scripts Time Out

TORQUE takes preventative measures against prologue and epilogue scripts by placing an alarm around the scripts execution. By default, TORQUE sets the alarm to go off after 5 minutes of execution. If the script exceeds this time, it will be terminated and the node will be marked down. This timeout can be adjusted by setting the **$prologalarm** parameter in the `mom_priv/config` file.

> While TORQUE is executing the `epilogue`, `epilogue.user`, or `epilogue.precancel` scripts, the job will be in the *E* (exiting) state.

If an `epilogue.parallel` script cannot open the `.OU` or `.ER` files, an error is logged but the script is continued.

Related topics

- **Appendix G: Prologue and Epilogue Scripts** on page 2469

Prologue Error Processing

If the `prologue` script executes successfully, it should exit with a zero status. Otherwise, the script should return the appropriate error code as defined in the table below. The pbs_mom will report the script's exit status to pbs_server which will in turn take the associated action. The following table describes each exit code for the prologue scripts and the action taken.

Error	Description	Action
-4	The script timed out	Job will be requeued

Error	Description	Action
-3	The wait(2) call returned an error	Job will be requeued
-2	Input file could not be opened	Job will be requeued
-1	Permission error (script is not owned by root, or is writable by others)	Job will be requeued
0	Successful completion	Job will run
1	Abort exit code	Job will be aborted
>1	other	Job will be requeued

Example 4-30:

Following are example prologue and epilogue scripts that write the arguments passed to them in the job's standard out file:

prologue	
Script	`#!/bin/sh` `echo "Prologue Args:"` `echo "Job ID: $1"` `echo "User ID: $2"` `echo "Group ID: $3"` `echo ""` `exit 0`
stdout	`Prologue Args:` `Job ID: 13724.node01` `User ID: user1` `Group ID: user1`

epilogue	
Script	`#!/bin/sh` `echo "Epilogue Args:"` `echo "Job ID: $1"` `echo "User ID: $2"` `echo "Group ID: $3"` `echo "Job Name: $4"` `echo "Session ID: $5"` `echo "Resource List: $6"` `echo "Resources Used: $7"` `echo "Queue Name: $8"` `echo "Account String: $9"` `echo ""` `exit 0`
stdout	`Epilogue Args:` `Job ID: 13724.node01` `User ID: user1` `Group ID: user1` `Job Name: script.sh` `Session ID: 28244` `Resource List: neednodes=node01,nodes=1,walltime=00:01:00` `Resources Used: cput=00:00:00,mem=0kb,vmem=0kb,walltime=00:00:07` `Queue Name: batch` `Account String:`

Example 4-31:

The Ohio Supercomputer Center contributed the following scripts:

"prologue creates a unique temporary directory on each node assigned to a job before the job begins to run, and epilogue deletes that directory after the job completes.

> Having a separate temporary directory on each node is probably not as good as having a good, high performance parallel filesystem.

```
prologue

#!/bin/sh
# Create TMPDIR on all the nodes
# Copyright 1999, 2000, 2001 Ohio Supercomputer Center
# prologue gets 3 arguments:
# 1 -- jobid
# 2 -- userid
# 3 -- grpid
#
jobid=$1
user=$2
group=$3
nodefile=/var/spool/pbs/aux/$jobid
if [ -r $nodefile ] ; then
    nodes=$(sort $nodefile | uniq)
else
    nodes=localhost
fi
tmp=/tmp/pbstmp.$jobid
for i in $nodes ; do
    ssh $i mkdir -m 700 $tmp \&\& chown $user.$group $tmp
done
exit 0
```

```
epilogue

#!/bin/sh
# Clear out TMPDIR
# Copyright 1999, 2000, 2001 Ohio Supercomputer Center
# epilogue gets 9 arguments:
# 1 -- jobid
# 2 -- userid
# 3 -- grpid
# 4 -- job name
# 5 -- sessionid
# 6 -- resource limits
# 7 -- resources used
# 8 -- queue
# 9 -- account
#
jobid=$1
nodefile=/var/spool/pbs/aux/$jobid
if [ -r $nodefile ] ; then
    nodes=$(sort $nodefile | uniq)
else
    nodes=localhost
fi
tmp=/tmp/pbstmp.$jobid
for i in $nodes ; do
    ssh $i rm -rf $tmp
done
exit 0
```

`prologue`, `prologue.user`, and `prologue.parallel` scripts can have dramatic effects on job scheduling if written improperly.

Related topics

- Appendix G: Prologue and Epilogue Scripts on page 2469

Appendix H: Running Multiple TORQUE Servers and MOMs on the Same Node

TORQUE can be configured to allow multiple servers and MOMs to run on the same node. This example will show how to configure, compile and install two different TORQUE servers and MOMs on the same node. For details, see these topics:

- **Configuring the first TORQUE** on page 2477
- **Configuring the second TORQUE** on page 2477
- **Bringing the first TORQUE server online** on page 2477
- **Bringing the second TORQUE server online** on page 2478

Configuring the first TORQUE

```
./configure --with-server-home=/usr/spool/PBS1 --bindir=/usr/spool/PBS1/bin --
sbindir=/usr/spool/PBS1/sbin
```

Then make and make install will place the first TORQUE into `/usr/spool/PBS1` with the executables in their corresponding directories.

Configuring the second TORQUE

```
./configure --with-server-home=/usr/spool/PBS2 --bindir=/usr/spool/PBS2/bin --
sbindir=/usr/spool/PBS2/sbin
```

Then make and make install will place the second TORQUE into `/usr/spool/PBS2` with the executables in their corresponding directories.

Bringing the first TORQUE server online

Each command, including pbs_server and pbs_mom, takes parameters indicating which servers and ports to connect to or listen on (when appropriate). Each of these is documented in their corresponding man pages (configure with `--enable-docs`).

In this example the first TORQUE server will accept batch requests on port 35000, communicate with the MOMs on port 35001, and communicate via RPP on port 35002. The first TORQUE MOM will try to connect to the server on port 35000, it will listen for requests from the server on port 35001 and will communicate via RPP on port 35002. (Each of these command arguments is discussed in further details on the corresponding man page. In particular, `-t create` is only used the first time a server is run.)

```
> pbs_server -p 35000 -M 35001 -R 35002 -t create
> pbs_mom -S 35000 -M 35001 -R 35002
```

Afterwards, when using a client command to make a batch request it is necessary to specify the server name and server port (35000):

```
> pbsnodes -a -s node01:35000
```

Submitting jobs can be accomplished using the `-q` option ([queue][@host[:port]]):

```
> qsub -q @node01:35000 /tmp/script.pbs
```

Bringing the second TORQUE server online

In this example the second TORQUE server will accept batch requests on port 36000, communicate with the MOMS on port 36002, and communicate via RPP on port 36002. The second TORQUE MOM will try to connect to the server on port 36000, it will listen for requests from the server on port 36001 and will communicate via RPP on port 36002.

```
> pbs_server -p 36000 -M 36001 -R 36002 -t create
> pbs_mom -S 36000 -M 36001 -R 36002
```

Afterward, when using a client command to make a batch request it is necessary to specify the server name and server port (36002):

```
> pbsnodes -a -s node01:36000
> qsub -q @node01:36000 /tmp/script.pbs
```

Appendix I: Security Overview

The authorization model for TORQUE changed in version 4.0.0 from `pbs_iff` to a daemon called `trqauthd`. The job of the `trqauthd` daemon is the same as `pbs_iff`. The difference is that `trqauthd` is a resident daemon whereas `pbs_iff` is invoked by each client command. `pbs_iff` is not scalable and is prone to failure under even small loads. `trqauthd` is very scalable and creates the possibility for better security measures in the future.

trqauthd and pbs_iff authorization theory

The key to security of both `trqauthd` and `pbs_iff` is the assumption that any host which has been added to the TORQUE cluster has been secured by the administrator. Neither `trqauthd` nor `pbs_iff` do authentication. They only do authorization of users. Given that the host system is secure the following is the procedure by which `trqauthd` and `pbs_iff` authorize users to `pbs_server`.

1. Client utility makes a connection to `pbs_server` on a dynamic port.
2. Client utility sends a request to `trqauthd` with the user name and port.
3. `trqauthd` verifies the user ID and then sends a request to `pbs_server` on a privileged port with the user ID and dynamic port to authorize the connection.
4. `trqauthd` reports results of the server to client utility.

Both `trqauthd` and `pbs_iff` use Unix domain sockets for communication from the client utility. Unix domain sockets have the ability to verify that a user is who they say they are by using security features that are part of the file system.

Appendix J: Job Submission Filter ("qsub wrapper")

When a "submit filter" exists, TORQUE will send the command file (or contents of STDIN if piped to qsub) to that script/executable and allow it to evaluate the submitted request based on specific site policies. The resulting file is then handed back to qsub and processing continues. Submit filters can check user jobs for correctness based on site policies. They can also modify user jobs as they are submitted. Some examples of what a submit filter might evaluate and check for are:

- Memory Request - Verify that the job requests memory and rejects if it does not.
- Job event notifications - Check if the job does one of the following and rejects it if it:
  - explicitly requests no notification.
  - requests notifications but does not provide an email address.
- Walltime specified - Verify that the walltime is specified.
- Global Walltime Limit - Verify that the walltime is below the global max walltime.
- Test Walltime Limit - If the job is a test job, this check rejects the job it if it requests a walltime longer than the testing maximum.

The script below reads the original submission request from STDIN and shows how you could insert parameters into a job submit request:

```
#!/bin/sh
# add default memory constraints to all requests
# that did not specify it in user's script or on command line
echo "#PBS -l mem=16MB"
while read i
do
echo $i
done
```

If you use a qsub script that includes #PBS directives to pass arguments instead of on the command line; for example,

```
#!/bin/sh
#
#This is an example script example.sh
#
#These commands set up the Grid Environment for your job:
#PBS -N ExampleJob
#PBS -l nodes=1,walltime=00:01:00
#PBS -q np_workq
#PBS -M YOURUNIQNAME@umich.edu
#PBS -m abe

#print the time and date
date

#wait 10 seconds
sleep 10

#print the time and date again
date
```

then your submit filter script must detect these directives and print them to stdout in order for qsub to see them.

The same command line arguments passed to qsub will be passed to the submit filter and in the same order. Exit status of -1 will cause qsub to reject the submission with a message stating that it failed due to administrative policies.

The "submit filter" must be executable, must be available on each of the nodes where users may submit jobs, and by default must be located at `${libexecdir}/qsub_filter` (for version 2.1 and older: `/usr/local/sbin/torque_submitfilter`). At run time, if the file does not exist at this new preferred path then qsub will fall back to the old hard-coded path. The submit filter location can be customized by setting the *SUBMITFILTER* parameter inside the file (see **Appendix K: "torque.cfg" Configuration File** on page 2480), as in the following example:

`torque.cfg:`

```
SUBMITFILTER /opt/torque/submit.pl
...
```

Initial development courtesy of Oak Ridge National Laboratories.

Appendix K: "torque.cfg" Configuration File

CLIENTRETRY	
Format	<INT>

CLIENTRETRY	
Default	0
Description	Seconds between retry attempts to talk to pbs_server.
Example	`CLIENTRETRY 10` *TORQUE waits 10 seconds after a failed attempt before it attempts to talk to pbs_server again.*

DEFAULTCKPT	
For mat	One of *None*, *Enabled*, *Shutdown*, *Periodic*, *Interval=minutes*, *depth=number*, or *dir=path*
Default	*None*
Description	Default value for job's checkpoint attribute. For a description of all possible values, see **qsub** on page 2395. This default setting can be overridden at job submission with the qsub -c option.
Example	`DEFAULTCKPT Shutdown` *By default, TORQUE checkpoints at pbs_mom shutdown.*

FAULT_TOLERANT_BY_DEFAULT	
Format	<BOOLEAN>
Default	FALSE
Description	Sets all jobs to fault tolerant by default. (See `qsub -f` for more information on fault tolerance.)
Example	`FAULT_TOLERANT_BY_DEFAULT TRUE` *Jobs are fault tolerant by default. They will not be canceled based on failed polling, no matter how many nodes fail to report.*

HOST_NAME_SUFFIX	
Format	<STRING>
Default	---
Description	Specifies a hostname suffix. When qsub submits a job, it also submits the username of the submitter and the name of the host from which the user submitted the job. TORQUE appends the value of HOST_NAME_SUFFIX to the hostname. This is useful for multi-homed systems that may have more than one name for a host.
Example	`HOST_NAME_SUFFIX -ib` *When a job is submitted, the -ib suffix is appended to the host name.*

QSUBHOST	
Format	<HOSTNAME>
Default	---
Description	The hostname given as the argument of this option will be used as the PBS_O_HOST variable for job submissions. By default, PBS_O_HOST is the hostname of the submission host. This option allows administrators to override the default hostname and substitute a new name.
Example	`QSUBHOST host1` *The default hostname associated with a job is host1.*

QSUBSENDUID	
Format	N/A
Default	---
Description	Integer for job's PBS_OUID variable. Specifying the parameter name anywhere in the config file enables the feature. Removing the parameter name disables the feature.
Example	`QSUBSENDUID` *TORQUE assigns a unique ID to a job when it is submitted by qsub.*

QSUBSLEEP	
Format	<INT>
Default	0
Description	Specifies time, in seconds, to sleep between a user's submitting and TORQUE's starting a qsub command. Used to prevent users from overwhelming the scheduler.
Example	`QSUBSLEEP 2` *When a job is submitted with qsub, it will sleep for 2 seconds.*

RERUNNABLEBYDEFAULT	
Format	<BOOLEAN>
Default	TRUE
Description	Specifies if a job is re-runnable by default. Setting this to false causes the re-runnable attribute value to be false unless the users specifies otherwise with the **qsub** `-r` option. (New in TORQUE 2.4.)
Example	`RERUNNABLEBYDEFAULT FALSE` *By default, qsub jobs cannot be rerun.*

SERVERHOST	
Format	<STRING>
Default	localhost
Description	If set, the **qsub** on page 2395 command will open a connection to the host specified by the SERVERHOST string.
Example	`SERVERHOST orion15` *The server will open socket connections and and communicate using serverhost orion15.*

SUBMITFILTER	
Format	<STRING>
Default	${libexecdir}/qsub_filter (for version 2.1 and older: /usr/local/sbin/torque_submitfilter)
Description	Specifies the location of the submit filter (see **Appendix J: Job Submission Filter ("qsub wrapper")** on page 2479 used to pre-process job submission.
Example	`SUBMITFILTER /usr/local/sbin/qsub_filter` *The location of the submit filter is specified as* `/usr/local/sbin/qsub_filter`*.*

TRQ_IFNAME	
Format	<STRING>
Default	null
Description	Allows you to specify a specific network interface to use for outbound TORQUE requests. The string is the name of a network interface, such as *eth0* or *eth1*, depending on which interface you want to use.
Example	`TRQ_IFNAME eth1` *Outbound TORQUE requests are handled by* eth1*.*

VALIDATEGROUP	
Format	<BOOLEAN>
Default	FALSE
Description	Validate submit user's group on **qsub** commands. For TORQUE builds released after 2/8/2011, *VALIDATEGROUP* also checks any groups requested in group_list at the submit host. Set *VALIDATEGROUP* to "TRUE" if you set **disable_server_id_check** to TRUE.
Example	`VALIDATEGROUP TRUE` qsub *verifies the submitter's group ID.*

VALIDATEPATH	
Format	<BOOLEAN>
Default	TRUE
Description	Validate local existence of '-d' working directory.
Example	`VALIDATEPATH FALSE` *qsub does not validate the path.*

Appendix L: TORQUE Quick Start Guide

Initial installation

TORQUE is now hosted at https://github.com under the adaptivecomputing organization. To download source, you will need to use the git utility. For example:

```
[root]# git clone https://github.com/adaptivecomputing.com/torque.git -b 5.0.1 5.0.1
```

To download a different version, replace each 5.0.1 with the desired version. After downloading a copy of the repository, you can list the current branches by typing `git branch -a` from within the directory of the branch you cloned.

> If you're checking source out from git, read the `README.building-40` file in the repository.

Extract and build the distribution on the machine that will act as the "TORQUE server" - the machine that will monitor and control all compute nodes by running the pbs_server daemon. See the example below:

```
> tar -xzvf torque.tar.gz
> cd torque
> ./configure
> make
> make install
```

> OSX 10.4 users need to change the *#define __TDARWIN* in `src/include/pbs_config.h` to *#define __TDARWIN_8*.

> After installation, verify you have PATH environment variables configured for `/usr/local/bin/` and `/usr/local/sbin/`. Client commands are installed to `/usr/local/bin` and server binaries are installed to `/usr/local/sbin`.

> In this document, TORQUE_HOME corresponds to where TORQUE stores its configuration files. The default is `/var/spool/torque`.

Initialize/Configure TORQUE on the server (pbs_server)

- Once installation on the TORQUE server is complete, configure the pbs_server daemon by executing the command `torque.setup <USER>` found packaged with the distribution source code, where *<USER>* is a username that will act as the TORQUE admin. This script will set up a basic batch queue to get you started. If you experience problems, make sure that the most recent TORQUE executables are being executed, or that the executables are in your current PATH.

> If you are upgrading from TORQUE 2.5.9, run pbs_server -u before running `torque.setup`.
>
> ```
> [root]# pbs_server -u
> ```

- If doing this step manually, be certain to run the command `pbs_server -t create` to create the new batch database. If this step is not taken, the pbs_server daemon will be unable to start.
- Proper server configuration can be verified by following the steps listed in Testing server configuration.

Install TORQUE on the compute nodes

To configure a compute node do the following on each machine (see page 19, Section 3.2.1 of *PBS Administrators Manual* for full details):

- Create the self-extracting, distributable packages with make packages (See the `INSTALL` file for additional options and features of the distributable packages) and use the parallel shell command from your cluster management suite to copy and execute the package on all nodes (i.e. xCAT users might do `prcp torque-package-linux-i686.sh main:/tmp/; psh main /tmp/torque-package-linux-i686.sh --install`). Optionally, distribute and install the clients package.

Configure TORQUE on the compute nodes

- For each compute host, the MOM daemon must be configured to trust the pbs_server daemon. In TORQUE 2.0.0p4 and earlier, this is done by creating the `TORQUE_HOME/mom_priv/config` file and setting the **$pbsserver** parameter. In TORQUE 2.0.0p5 and later, this can also be done by creating the `TORQUE_HOME/server_name` file and placing the server hostname inside.
- Additional config parameters may be added to `TORQUE_HOME/mom_priv/config` (see **Appendix C: Node Manager (MOM) Configuration** on page 2435 for details).

Configure data management on the compute nodes

Data management allows jobs' data to be staged in/out or to and from the server and compute nodes.

- For shared filesystems (i.e., NFS, DFS, AFS, etc.) use the **$usecp** parameter in the `mom_priv/config` files to specify how to map a user's home directory.

  (Example: `$usecp gridmaster.tmx.com:/home /home`)
- For local, non-shared filesystems, rcp or scp must be configured to allow direct copy without prompting for passwords (key authentication, etc.)

Update TORQUE server configuration

On the TORQUE server, append the list of newly configured compute nodes to the `TORQUE_HOME/server_priv/nodes` file:

```
server_priv/nodes

computenode001.cluster.org
computenode002.cluster.org
computenode003.cluster.org
```

Start the pbs_mom daemons on compute nodes

- Next start the pbs_mom daemon on each compute node by running the pbs_mom executable.

Run the trqauthd daemon to run client commands (see **Configuring trqauthd for Client Commands** on page 2205). This enables running client commands.

Verifying correct TORQUE installation

The pbs_server daemon was started on the TORQUE server when the `torque.setup` file was executed or when it was manually configured. It must now be restarted so it can reload the updated configuration changes.

```
# shutdown server
> qterm # shutdown server

# start server
> pbs_server

# verify all queues are properly configured
> qstat -q

# view additional server configuration
> qmgr -c 'p s'

# verify all nodes are correctly reporting
> pbsnodes -a

# submit a basic job
>echo "sleep 30" | qsub

# verify jobs display
> qstat
```

At this point, the job will not start because there is no scheduler running. The scheduler is enabled in the next step below.

Enabling the scheduler

Selecting the cluster scheduler is an important decision and significantly affects cluster utilization, responsiveness, availability, and intelligence. The default TORQUE scheduler, pbs_sched, is very basic and will provide poor utilization of your cluster's resources. Other options, such as Maui Scheduler or Moab Workload Manager are highly recommended. If using Maui/Moab, see **Moab-TORQUE Integration Guide** on page 1206. If using pbs_sched, start this daemon now.

> If you are installing ClusterSuite, TORQUE and Moab were configured at installation for interoperability and no further action is required.

Startup/Shutdown service script for TORQUE/Moab (OPTIONAL)

Optional startup/shutdown service scripts are provided as an example of how to run TORQUE as an OS service that starts at bootup. The scripts are located in the `contrib/init.d/` directory of the TORQUE tarball you downloaded. In order to use the script you must:

- Determine which init.d script suits your platform the best.
- Modify the script to point to TORQUE's install location. This should only be necessary if you used a non-default install location for TORQUE (by using the `--prefix` option of `./configure`).
- Place the script in the `/etc/init.d/` directory.
- Use a tool like `chkconfig` to activate the start-up scripts or make symbolic links (S99moab and K15moab, for example) in desired runtimes (`/etc/rc.d/rc3.d/` on Redhat, etc.).

Related topics

- **Advanced configuration** on page 2207

Appendix M: BLCR Acceptance Tests

This section contains a description of the testing done to verify the functionality of the BLCR implementation. For details, see these topics:

Test Environment

All these tests assume the following test program and shell script, `test.sh`.

```
#include
int main( int argc, char *argv[] )
{
int i;

    for (i=0; i<100; i++)
    {
        printf("i = %d\n", i);
        fflush(stdout);
        sleep(1);
    }
}
#!/bin/bash

/home/test/test
```

Related topics

- **Appendix M: BLCR Acceptance Tests** on page 2488

Test 1 - Basic Operation

Introduction

This test determines if the proper environment has been established.

Test steps

Submit a test job and the issue a hold on the job.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
```

Possible failures

Normally the result of qhold is nothing. If an error message is produced saying that qhold is not a supported feature then one of the following configuration errors might be present.

- The TORQUE images may have not be configured with `--enable-blcr`
- BLCR support may not be installed into the kernel with insmod.
- The config script in mom_priv may not exist with `$checkpoint_script defined`.
- The config script in mom_priv may not exist with `$restart_script defined`.
- The config script in mom_priv may not exist with `$checkpoint_run_exe defined`.
- The scripts referenced in the config file may not exist.
- The scripts referenced in the config file may not have the correct permissions.

Successful results

If no configuration was done to specify a specific directory location for the checkpoint file, the default location is off of the TORQUE directory, which in my case is `/var/spool/torque/checkpoint`.

Otherwise, go to the specified directory for the checkpoint image files. This was done by either specifying an option on job submission, i.e. `-c dir=/home/test` or by setting an attribute on the execution queue. This is done with the command `qmgr -c 'set queue batch checkpoint_dir=/home/test'`.

Doing a directory listing shows the following.

```
# find /var/spool/torque/checkpoint
/var/spool/torque/checkpoint
/var/spool/torque/checkpoint/999.xxx.yyy.CK
/var/spool/torque/checkpoint/999.xxx.yyy.CK/ckpt.999.xxx.yyy.1205266630
# find /var/spool/torque/checkpoint |xargs ls -l
-r-------- 1 root root 543779 2008-03-11 14:17
/var/spool/torque/checkpoint/999.xxx.yyy.CK/ckpt.999.xxx.yyy.1205266630

/var/spool/torque/checkpoint:
total 4
drwxr-xr-x 2 root root 4096 2008-03-11 14:17 999.xxx.yyy.CK

/var/spool/torque/checkpoint/999.xxx.yyy.CK:
total 536
-r-------- 1 root root 543779 2008-03-11 14:17 ckpt.999.xxx.yyy.1205266630
```

Doing a `qstat -f` command should show the job in a held state, *job_state = H*. Note that the attribute checkpoint_name is set to the name of the file seen above.

If a checkpoint directory has been specified, there will also be an attribute *checkpoint_dir* in the output of `qstat -f`.

```
$ qstat -f
Job Id: 999.xxx.yyy
    Job_Name = test.sh
    Job_Owner = test@xxx.yyy
    resources_used.cput = 00:00:00
    resources_used.mem = 0kb
    resources_used.vmem = 0kb
    resources_used.walltime = 00:00:06
    job_state = H
    queue = batch
    server = xxx.yyy
    Checkpoint = u
    ctime = Tue Mar 11 14:17:04 2008
    Error_Path = xxx.yyy:/home/test/test.sh.e999
    exec_host = test/0
    Hold_Types = u
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Tue Mar 11 14:17:10 2008
    Output_Path = xxx.yyy:/home/test/test.sh.o999
    Priority = 0
    qtime = Tue Mar 11 14:17:04 2008
    Rerunable = True
    Resource_List.neednodes = 1
    Resource_List.nodect = 1
    Resource_List.nodes = 1
    Resource_List.walltime = 01:00:00
    session_id = 9402 substate = 20
    Variable_List = PBS_O_HOME=/home/test,PBS_O_LANG=en_US.UTF-8,
        PBS_O_LOGNAME=test,
        PBS_O_PATH=/usr/local/perltests/bin:/home/test/bin:/usr/local/s
bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games,
        PBS_O_SHELL=/bin/bash,PBS_SERVER=xxx.yyy,
        PBS_O_HOST=xxx.yyy,PBS_O_WORKDIR=/home/test,
        PBS_O_QUEUE=batch
    euser = test
    egroup = test
    hashname = 999.xxx.yyy
    queue_rank = 3
    queue_type = E comment = Job started on Tue Mar 11 at 14:17
    exit_status = 271
    submit_args = test.sh
    start_time = Tue Mar 11 14:17:04 2008
    start_count = 1
    checkpoint_dir = /var/spool/torque/checkpoint/999.xxx.yyy.CK
    checkpoint_name = ckpt.999.xxx.yyy.1205266630
```

The value of `Resource_List.*` is the amount of resources requested.

Related topics

- **Appendix M: BLCR Acceptance Tests** on page 2488

Test 2 - Persistence of Checkpoint Images

Introduction

This test determines if the checkpoint files remain in the default directory after the job is removed from the TORQUE queue.

Note that this behavior was requested by a customer but in fact may not be the right thing to do as it leaves the checkpoint files on the execution node. These will gradually build up over time on the node being limited only by disk space. The right thing would seem to be that the checkpoint files are copied to the user's home directory after the job is purged from the execution node.

Test steps

Assuming the steps of Test 1 (see **Test 1 - Basic Operation** on page 2489), delete the job and then wait until the job leaves the queue after the completed job hold time. Then look at the contents of the default checkpoint directory to see if the files are still there.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
> qdel 999
> sleep 100
> qstat
>
> find /var/spool/torque/checkpoint
... files ...
```

Possible failures

The files are not there, did Test 1 actually pass?

Successful results

The files are there.

Related topics

- **Appendix M: BLCR Acceptance Tests** on page 2488

Test 3 - Restart after Checkpoint

Introduction

This test determines if the job can be restarted after a checkpoint hold.

Test steps

Assuming the steps of Test 1 (see **Test 1 - Basic Operation** on page 2489), issue a **qrls** command. Have another window open into the `/var/spool/torque/spool` directory and tail the job.

Successful results

After the qrls, the job's output should resume.

Related topics

- **Appendix M: BLCR Acceptance Tests** on page 2488

Test 4 - Multiple Checkpoint/Restart

Introduction

This test determines if the checkpoint/restart cycle can be repeated multiple times.

Test steps

Start a job and then while tailing the job output, do multiple **qhold**/**qrls** operations.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
> qrls 999
> qhold 999
> qrls 999
> qhold 999
> qrls 999
```

Successful results

After each qrls, the job's output should resume. Also tried "while true; do qrls 999; qhold 999; done" and this seemed to work as well.

Related topics

- **Appendix M: BLCR Acceptance Tests** on page 2488

Test 5 - Periodic Checkpoint

Introduction

This test determines if automatic periodic checkpoint will work.

Test steps

Start the job with the option `-c enabled,periodic,interval=1` and look in the checkpoint directory for checkpoint images to be generated about every minute.

```
> qsub -c enabled,periodic,interval=1 test.sh
999.xxx.yyy
```

Successful results

After each qrls, the job's output should resume. Also tried "while true; do qrls 999; qhold 999; done" and this seemed to work as well.

Related topics

- **Appendix M: BLCR Acceptance Tests** on page 2488

Test 6 - Restart from Previous Image

Introduction

This test determines if the job can be restarted from a previous checkpoint image.

Test steps

Start the job with the option `-c enabled,periodic,interval=1` and look in the checkpoint directory for checkpoint images to be generated about every minute. Do a **qhold** on the job to stop it. Change the attribute checkpoint_name with the **qalter** command. Then do a **qrls** to restart the job.

```
> qsub -c enabled,periodic,interval=1 test.sh
999.xxx.yyy
> qhold 999
> qalter -W checkpoint_name=ckpt.999.xxx.yyy.1234567
> qrls 999
```

Successful results

The job output file should be truncated back and the count should resume at an earlier number.

Related topics

- **Appendix M: BLCR Acceptance Tests** on page 2488