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 system 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.

Legal Notices

Table of Contents

- **1.0 Philosophy and Goals of the Moab Workload Manager**
  - 1.1 Value of a Batch System
  - 1.2 Philosophy and Goals
  - 1.3 Workload
- **2.0 Installation and Initial Configuration**
  - 2.1 Hardware and Software Requirements
  - 2.2 Building and Installing Moab
  - 2.3 Upgrading Moab
  - 2.4 Initial Moab Configuration
  - 2.5 Initial Moab Testing (Monitor, Interactive, Simulation and Normal Modes)
- **3.0 Scheduler Basics**
  - 3.1 Layout of Scheduler Components
  - 3.2 Scheduling Environment and Objects
  - 3.2.2 Scheduling Dictionary
  - 3.3 Scheduling Iterations and Job Flow
  - 3.4 Configuring the Scheduler
  - 3.5 Credential Overview
    - Job Attributes/Flags Overview
- **4.0 Scheduler Commands**
  - 4.1 Client Overview
  - 4.2 Monitoring System Status
  - 4.3 Managing Jobs
  - 4.4 Managing Reservations
  - 4.5 Configuring Policies
  - 4.6 End-user Commands
- **5.0 Prioritizing Jobs and Allocating Resources**
  - 5.1 Job Prioritization
Priority Overview
  - Job Priority Factors
    - Fairshare Job Priority Example
  - Common Priority Usage
  - Prioritization Strategies
  - Manual Job Priority Adjustment
  - 5.2 Node Allocation
  - 5.3 Node Access
  - 5.4 Node Availability
  - 5.5 Task Distribution
  - 5.6 Scheduling Jobs When VMs Exist

- **6.0 Managing Fairness - Throttling Policies, Fairshare, and Allocation Management**
  - 6.1 Fairness Overview
  - 6.2 Usage Limits/Throttling Policies
  - 6.3 Fairshare
  - Sample FairShare Data File
  - 6.4 Charging and Allocation Management
  - 6.5 Internal Charging Facilities

- **7.0 Controlling Resource Access - Reservations, Partitions, and QoS Facilities**
  - 7.1 Advance Reservations
    - 7.1.1 Reservation Overview
    - 7.1.2 Administrative Reservations
    - 7.1.3 Standing Reservations
    - 7.1.4 Reservation Policies
    - 7.1.5 Configuring and Managing Reservations
    - 7.1.6 Enabling Reservations for End Users
  - 7.2 Partitions
  - 7.3 QoS Facilities

- **8.0 Optimizing Scheduling Behavior - Backfill, Node Sets, and Preemption**
  - 8.1 Optimization Overview
  - 8.2 Backfill
  - 8.3 Node Sets
  - 8.4 Preemption

- **9.0 Statistics, Accounting, and Profiling**
  - 9.1 Scheduler Performance Evaluation Overview
  - 9.2 Accounting - Job and System Statistics
  - 9.3 Testing New Versions and Configurations
  - 9.4 Answering What If? Questions with the Simulator

- **10.0 Managing Shared Resources - Usage Based Limits, Policies, and SMP Issues**
  - 10.1 Consumable Resource Handling
  - 10.2 Load Balancing Features
  - 10.3 Resource Usage Limits
  - 10.4 General SMP Issues

- **11.0 General Job Administration**
  - 11.1 Deferred Jobs and Job Holds
  - 11.2 Job Priority Management
  - 11.3 Suspend/Resume Handling
  - 11.4 Checkpoint/Restart
  - 11.5 Job Dependencies
  - 11.6 Setting Job Defaults and Per Job Limits
  - 11.7 General Job Policies
  - 11.8 Using a Local Queue
  - 11.9 Job Deadline Support
  - 11.10 Job Templates

- **12.0 General Node Administration**
  - 12.1 Node Location (Partitions, Racks, Queues, etc.)
  - 12.2 Node Attributes (Node Features, Speed, etc.)
  - 12.3 Node Specific Policies (MaxJobPerNode, etc.)
12.4 Managing Shared Cluster Resources
12.5 Node State Management
12.6 Managing Consumable Generic Resources
12.7 Enabling Generic Metrics
12.8 Enabling Generic Events

12.9 Resource Managers and Interfaces
12.10 Adding Resource Manager Interfaces
12.11 Managing Resources Directly with the Native Interface
12.12 Utilizing Multiple Resource Managers
12.13 License Management
12.14 Provisioning Managers
12.15 Network Management
12.16 Integrating with Hardware Managers
12.17 Enabling Resource Manager Translation

13.0 Resource Managers and Interfaces
13.1 Resource Manager Overview
13.2 Resource Manager Configuration
13.3 Resource Manager Extensions
13.4 Adding Resource Manager Interfaces
13.5 Managing Resources Directly with the Native Interface
13.6 Utilizing Multiple Resource Managers
13.7 License Management
13.8 Provisioning Managers
13.9 Network Management
13.10 Integrating with Hardware Managers
13.11 Enabling Resource Manager Translation

13.12 Troubleshooting and System Maintenance
13.13 Internal Diagnostics
13.14 Logging Facilities
13.15 Object Messages
13.16 Notifying Administrators of Failures and Critical Events
13.17 Issues with Client Commands
13.18 Tracking System Failures
13.19 Problems with Individual Jobs
13.20 Diagnostic Scripts

13.21 Improving User Effectiveness
13.22 User Feedback Loops
13.23 User Level Statistics
13.24 Job Start Time Estimates
13.25 Collecting Performance Information on Individual Jobs

13.26 Cluster Analysis, Testing, and Simulation
13.27 Evaluating New Releases and Policies
13.28 Testing New Middleware
13.29 Simulation Overview
13.30 Simulation Overview
13.31 Resource Traces
13.32 Workload Accounting Records
13.33 Simulation Specific Configuration

13.34 Moab Workload Manager for Grids
13.35 Grid Basics
13.36 Grid Configuration
13.37 Centralized Grid Management
13.38 Source-Destination Grid Management
13.39 Localized Grid Management
13.40 Resource Control and Access
13.41 Workload Submission and Control
13.42 Reservations in the Grid
13.43 Grid Usage Policies
13.44 Grid Scheduling Policies
13.45 Grid Credential Management
13.46 Grid Data Management
13.47 Accounting and Allocation Management
13.48 Grid Security
13.49 Grid Diagnostics and Validation

13.50 Green Computing
13.51 Establishing Script Interaction between Moab and a Power Management Tool
- 18.2 Enabling Green Power Management
- 18.3 Allocating and Adjusting Green Pool Size
- 18.4 Miscellaneous Power Management Options

- **19.0 Object Triggers**
  - 19.1 Trigger Creation
  - 19.2 Trigger Management
  - 19.3 Trigger Components
  - 19.4 Trigger Types
  - 19.5 Trigger Variables
  - 19.6 Trigger Examples

- **20.0 Virtual Private Clusters**
  - 20.1 Configuring VPC Profiles
  - 20.2 VPC Commands

- **21.0 Miscellaneous**
  - 21.1 User Feedback
  - 21.2 Enabling High Availability Features
  - 21.3 Identity Managers
  - 21.4 Information Services for the Enterprise and Grid
  - 21.5 Malleable Jobs

- **22.0 Database Configuration**
  - 22.1 SQLite3
  - 22.2 Connecting to a MySQL Database with an ODBC Driver

- **Appendices**
  - Appendix D: Adjusting Default Limits
  - Appendix E: Security Configuration
  - Appendix F: Parameters Overview
  - Appendix G: Commands Overview
    - Commands - checkjob
    - Commands - checknode
    - Commands - mcredctl
    - Commands - mdiag
      - Commands - mdiag -a (accounts)
      - Commands - mdiag -b (queues)
      - Commands - mdiag -c (class)
      - Commands - mdiag -f (fairshare)
      - Commands - mdiag -g (group)
      - Commands - mdiag -j (job)
      - Commands - mdiag -n (nodes)
      - Commands - mdiag -p (priority)
      - Commands - mdiag -q (QoS)
      - Commands - mdiag -r (reservation)
      - Commands - mdiag -R (Resource Manager)
      - Commands - mdiag -S
      - Commands - mdiag -t (Partition)
      - Commands - mdiag -T (Triggers)
      - Commands - mdiag -u (user)
    - Commands - mjobctl
    - timespec.shtml
    - Commands - mnodectl
    - Commands - moab
    - Commands - mrmctl
    - Commands - mrsvctl
    - Commands - mschedctl
    - Commands - mshow
      - Commands - mshow -a (Available Resources)
      - Commands - mshow (Usage in a Hosting Environment)
    - Commands - msub
- Applying the msub Submit Filter
- Submitting Jobs via msub in XML
- Commands - mvmct!
- Commands - resetstats
- Commands - showbf
- Commands - showq
- Commands - showres
- Commands - showstart
- Commands - showstate
- Commands - showstats
- Commands - showstats -f
- Commands Providing Maui Compatibility
  - Commands - canceljob
  - Commands - changeparam
  - Commands - diagnose
  - Commands - releasehold
  - Commands - releaseres
  - Commands - runjob
  - Commands - sethold
  - Commands - setqos
  - Commands - setres
  - Commands - setspri
  - Commands - showconfig
- Appendix H: Interfacing with Moab (APIs)
  - Moab Java API Quick Start Guide
- Appendix I: Considerations for Large Clusters
- Appendix J: Adding Moab as a Service
- Appendix K: Migrating from Maui 3.2
- Appendix O: Resource Manager Integration Overview
  - Compute Resource Managers
    - Moab-Loadleveler Integration Guide
    - Moab-TORQUE/PBS Integration Guide
      - PBS Integration Guide - RM Access Control
      - pbsdefault.shtml
    - Moab-SGE Integration Notes
    - Moab-SLURM Integration Guide
    - Wiki Interface Overview
      - Wiki Interface Specification
      - Wiki Socket Protocol Description
      - Wiki Configuration
    - Moab-LSF Integration Guide
      - LSF Integration via the Native Interface
    - Cray XT/TORQUE Integration Guide
  - Provisioning Resource Managers
    - Validating an xCAT Installation for Use with Moab
    - Integrating an xCAT Physical Provisioning Resource Manager with Moab
    - Enabling Moab Provisioning with SystemImager
  - Hardware Integration
    - NUMA - Integration Guide
- Appendix Q: Moab in the Data Center
- Appendix W: Wiki Interface Overview
- SCHEDCFG Flags
Legal Notices

Copyright

© 2011 Adaptive Computing Enterprises, Inc. All rights reserved. Distribution of this document for commercial purposes in either hard or soft copy form is strictly prohibited without prior written consent from Adaptive Computing Enterprises, Inc.

Trademarks

Adaptive Computing, Cluster Resources, Moab, Moab Workload Manager, Moab Viewpoint, Moab Cluster Manager, Moab Cluster Suite, Moab Grid Scheduler, Moab Grid Suite, Moab Access Portal, and other Adaptive Computing products are either registered trademarks or trademarks of Adaptive Computing Enterprises, Inc. The Adaptive Computing logo and the Cluster Resources logo are trademarks of Adaptive Computing Enterprises, Inc. All other company and product names may be trademarks of their respective companies.
1.0 Philosophy

A 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.

- 1.1 Value of a Batch System
- 1.2 Philosophy and Goals of the Moab Scheduler
- 1.3 Workload
1.1 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:

- 1.1.1 Traffic Control
- 1.1.2 Mission Policies
- 1.1.3 Optimizations

1.1.1 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.

1.1.2 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.

1.1.3 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.
1.2 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.

1.2.1 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

1.2.2 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

1.2.3 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
1.3 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:

- 1.3.1 Batch Workload
- 1.3.2 Interactive Workload
- 1.3.3 Calendar Workload
- 1.3.4 Service Workload

1.3.1 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.

1.3.2 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.

1.3.3 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 may 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.

1.3.4 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.

**See Also**

- QOS/SLA Enforcement
2.0 Installation and Initial Configuration

- 2.1 Prerequisites
- 2.2 Building and Installing Moab
- 2.3 Upgrading Moab
- 2.4 Initial Configuration
- 2.5 Initial Testing
2.1 Hardware and Software Requirements

- 2.1.1 Hardware Requirements
- 2.1.2 Supported Platforms

2.1.1 Hardware Requirements

Adaptive Computing recommends a quad-core system with 8 GB of RAM and at least 100 GB of disk space; such a configuration is sufficient for most operating environments. If you have questions about unique configuration requirements, contact your account representative.

2.1.2 Supported Platforms

Moab works with a variety of platforms. Many commonly used resource managers, operating systems, and architectures are supported.

2.1.1.1 Resource Managers that Integrate with Moab

The following resource managers integrate with Moab:

- BProc
- clubMASK
- LoadLeveler
- LSF
- OpenPBS
- PBSPro
- S3
- SLURM
- TORQUE
- WIKI
- xCAT

2.1.1.2 Supported Operating Systems

Moab supports variants of Linux, including:

- Debian
- Fedora
- FreeBSD
- RedHat
- SuSE

Moab supports variants of Unix, including:

- AIX
- IRIX
- HP-UX
- OS/X
- OSF/Tru-64
- Solaris
- SunOS

2.1.1.3 Supported Architectures

Supported hardware architectures:

- AMD x86
- AMD Opteron
- HP
- Intel x86
- Intel IA-32
- Intel IA-64
- IBM i-Series
- IBM p-Series
- IBM x-Series
- IBM SP
- Mac G4, G5
- SGI Altix
2.2 Building and Installing Moab

- 2.2.1 Moab Server Installation
- 2.2.2 Moab Client Installation

After reading this section you will be able to:

- install the Moab server.
- install end-user commands on remote systems.

This section assumes a working knowledge of Linux or Unix based operating systems, including use of commands such as:

- `tar`
- `make`
- `vi`

Some operating systems use different commands (such as `gmake` and `gtar` instead of `make` and `tar`).

### 2.2.1 Moab Server Installation

Before installing Moab, view the Prerequisites to verify your platform is supported.

By default, the Moab home directory is configured as `/opt/moab`, the Moab server daemon is installed to `/opt/moab/sbin/`, and the client commands are installed to `/opt/moab/bin/`. `$MOABHOMEDIR` is the location of the `etc/`, `log/`, `spool/`, and `stat/` directories. `$MOABHOMEDIR` is the default location for the `moab.cfg` and `moab-private.cfg` files. Moab searches for server files in this order:

```
/opt/moab/
/opt/moab/etc/
/etc/
$MOABHOMEDIR
```

$MOABHOMEDIR is required whenever the Moab binary is started or when client commands are used. Adaptive Computing recommends putting $MOABHOMEDIR in a global location, such as `/etc/profile`, `/etc/bashrc`, or `/etc/environment`.

Moab contains a number of architectural parameter settings that you can adjust for non-standard installations. See Appendix D - Adjusting Default Limits and make any needed changes prior to using `make install`.

### To install Moab

1. Untar the distribution file.

```
> tar -xzvf moab-6.0.0.tar.gz
```

2. Navigate to the unpacked directory.

```
> cd moab-6.0.0
```

3. Configure the installation package.

You can customize the location of the Moab home directory, the server daemon, the client commands, and configure Moab to use a resource manager when using the `.configure` command. For a complete list of options, use `.configure --help`.

An example of some commonly used options for `.configure` is provided below.

```
> /configure --prefix=/usr/local --with-homedir=/var/moab --with-torque=/var/spool/torque/
```

In the above example:
The install directory (--prefix option) is configured as /usr/local and the home directory (--with-homedir option) is configured as /var/moab.

The Moab server daemon installs to /usr/local/sbin/.

The Moab client commands install to /usr/local/bin/.

The Moab tools files install to /usr/local/tools/.

Moab is configured to work with the TORQUE resource manager.

All Moab executables are placed in $MOABHOMEDIR/bin (such as /moab-6.0.0/bin/) until the installation is performed.

If you choose the default path (/opt/moab/), the administrator must update $PATH manually to include the new default folders.

You can install the Moab init script, allowing Moab to start automatically when the machine is booted, by using --with-init.

4. Install Moab.

Moab should be installed by root. If you cannot install Moab as root, please contact Customer Support.

> sudo make install

A default moab.cfg file will be created in the Moab home directory.

5. Copy the license file.

> cp moab.lic $MOABHOMEDIR/moab.lic

The license file should be placed in the same directory as moab.cfg (which is /opt/moab/ by default) before starting Moab. 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, administrator e-mail notification must be enabled and moab.cfg must contain e-mail addresses for level 1 administrators:

```
ADMINCFG[1] USERS=u1,u2,u3[,...]
USERCFG[u1] u1@company.com
USERCFG[u2] u2@company.com
USERCFG[u3] u3@company.com
MAILPROGRAM DEFAULT
```

Moab has an internal license that enables some functionality for a limited time for evaluation purposes. If you want to enable adaptive energy management, dynamic multi-OS provisioning, grid management, and other features, or if you want to evaluate Moab for a longer period, contact evaluation support. Use mdiag -S -v to see which features your license supports.


> moab

moabd is an alternate method of starting Moab that sets the MOABHOMEDIR and LD_LIBRARY_PATH environment variables before calling the Moab binary. It is safe and recommended if things are not installed in their default locations, but can be used in all cases.

2.2.2 Moab Client Installation

Moab has several client commands that are used at remote locations to check various statistics. You can choose to copy all, or only a subset, of the client commands to a remote location. Below is a suggested list of client commands to install on end-user accessible nodes.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>

- sudo make install
- cp moab.lic $MOABHOMEDIR/moab.lic
- moab
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkjob</td>
<td>display detailed job summary</td>
</tr>
<tr>
<td>msub</td>
<td>submit a job</td>
</tr>
<tr>
<td>showq</td>
<td>display job queue</td>
</tr>
<tr>
<td>showbf</td>
<td>display immediate resource availability</td>
</tr>
<tr>
<td>showstart</td>
<td>display estimated job start time</td>
</tr>
<tr>
<td>showstats</td>
<td>display usage statistics</td>
</tr>
<tr>
<td>setres</td>
<td>create personal reservation</td>
</tr>
<tr>
<td>releaseres</td>
<td>release personal reservation</td>
</tr>
</tbody>
</table>

For more information on all client commands, see the Commands Overview.

### 2.2.2.1 Command Installation when Server and Client Have Similar Architecture

Moab commands are enabled on remote systems by copying desired command executables to the client machine (or to a shared network file system). To enable client communication with the Moab server, use a nearly-blank `moab.cfg` file on the client system that has only one line that defines the `SCHEDCFG` parameter with a `SERVER` attribute.

```
SCHEDCFG[Moab] SERVER=moabserver:42559
```

Place the file in `/etc` on the remote submission hosts.

> The client commands and the Moab daemon must have the same version and revision number.

### 2.2.2.2 Command Installation when Server and Client Have Diverse Architecture

Moab clients need to be built for each client system that has different architecture from the server. If you are using secret key security (enabled by default), a common secret key must be specified for the client and server. Verify `moab-private.cfg` is configured properly.

**See Also**

- End User Commands
2.3 Upgrading Moab

Upgrading Moab may require changing the database. Please see the README.database file included in the Moab distribution for specific version information. You can 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.

The Moab 5.4 uninstaller does not remove the 5.4 man pages. These must be removed manually when upgrading from Moab 5.4. You can use this script to remove the man pages:

```
------Start of script------
#!/bin/bash
LIST="checkjob.1 mdiag.1 mdiag-n.1 mjobctl.1 mrmctl.1 mshow.1 releaseres.1 schedctl.1
showconfig.1 showstart.1 checknode.1 mdiag-f.1 mdiag-p.1 mjstat.1 mrsvtcl.1 mshowa.1
resetstats.1 setres.1 showq.1 showstate.1 mcredctl.1 mdiag-I.1 mdiag-S.1
mnodectl.1 mschedctl.1 mshowa.1 msub.1 runjob.1 showbf.1 showres.1 showstats.1"
MAN_DIR=/usr/local/share/man
for file in $LIST
do
   rm -f $MAN_DIR/man1/$file
done
------End of script------
```

To upgrade Moab:

1. Untar the distribution file.
   ```
   > tar -xzvf moab-6.0.0.tar.gz
   ```
2. Navigate to the unpacked directory.
   ```
   > cd moab-6.0.0
   ```
3. 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.
4. Stop Moab.
   The Moab server must be stopped before the new version is installed.
   ```
   > 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.
   - New jobs cannot be submitted to Moab.
5. Install Moab.
   ```
   > sudo make install
   ```
   Moab should be installed by root. If you cannot install Moab as root, please contact Customer Support.
6. Verify the version number is correct before starting the new server version.

```
> moab --about
```

```
Defaults:   server=:42559  cfgdir=/opt/moab  vardir=/opt/moab
Build dir: /home/admin01/dev/moab/
Build host: node01
Build date: Thu Mar  5 13:08:47 MST 2009
Build args: NA
Compiled as little endian.
Version: moab server 5.0.0 (revision 12867)
Copyright 2000-2008 Cluster Resources, Inc., All Rights Reserved
```

7. Start Moab.

```
> moab
```

```
> moabd is a safe and recommended method of starting Moab if things are not installed in their default locations.
```

2.3.1 Upgrading the Moab 6.0 Database

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. 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. Below are the SQL statements required to update the schema for Moab 6.0.

```
<  Description VARCHAR(1024),
  >  Description TEXT,
  >
ALTER TABLE Events ADD COLUMN Name VARCHAR(64);
```

```
CREATE TABLE Nodes (       ID VARCHAR(64),
  OperatingSystem VARCHAR(64),
  ConfiguredProcessors 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),
  OSLList 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,
```
CREATE TABLE ProvisionData (  
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,  
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,  
PSDedicated DOUBLE,  
PSUtilized DOUBLE,  
PRIMARY KEY (ID)) ;

CREATE TABLE Requests (  
JobID VARCHAR(64),  
PRIMARY KEY (ID)) ;
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 ("TNode",18);
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 ("Taction",43);
INSERT INTO ObjectType (Name,ID) VALUES ("VM",45);
INSERT INTO ObjectType (Name,ID) VALUES ("VPC",47);
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 ("NODEPOWERON",48);
INSERT INTO EventType (Name,ID) VALUES ("NODEPOWEROFF",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);
2.4 Initial Moab Configuration

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:

- **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
  ```

- **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
  ```

See Also

- Parameter Overview
- mdiag -C command (for diagnosing current Moab configuration)
2.5 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:

- **2.5.1 Minimal Configuration Required To Start**
  - 2.5.1.1 Normal Mode
  - 2.5.1.2 Monitor Mode
  - 2.5.1.3 Interactive Mode
  - 2.5.1.4 Simulation Mode

2.5.1 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 line **SCHEDCFG MODE=MONITOR** in the **moab.cfg** file.

2.5.1.1 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.

2.5.1.2 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.

2.5.1.2.1 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.

### 2.5.1.3 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.

### 2.5.1.4 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.

The initial test drive simulation can be configured using the following steps:

(Consider viewing the simulation configuration demo.)

```
> vi moab.cfg

(change the `MODE` attribute of the `SCHEDCFG` parameter from `NORMAL` to `SIMULATION`)
(add 'SIMRESOURCETRACEFILE traces/Resource.Trace1')
(add 'SIMWORKLOADTRACEFILE traces/Workload.Trace1')

> moab &
```

In simulation mode, the scheduler does not background itself as it does in monitor and normal modes.

The sample workload and resource traces files allow the simulation to emulate a 192 node IBM SP. In this mode, all Moab commands can be run as if on a normal system. The `mschedctl` command can be used to advance the simulation through time. The Simulation section describes the use of the simulator in detail.

If you are familiar with Moab, you may want to use the simulator to tune scheduling policies for your own workload and system. The resource and workload traces are described further in the Collecting Traces section.
Generally, at least a week's worth of workload should be collected to make the results of the simulation statistically meaningful. Once the traces are collected, the simulation can be started with some initial policy settings. Typically, the scheduler is able to simulate between 10 and 100 minutes of wallclock time per second for medium to large systems. As the simulation proceeds, various statistics can be monitored if desired. At any point, the simulation can be ended and the statistics of interest recorded. One or more policies can be modified, the simulation re-run, and the results compared. Once you are satisfied with the scheduling results, you can run the scheduler live with the tuned policies.
3.0 Scheduler Basics

- 3.1 Layout of Scheduler Components
- 3.2 Scheduling Environment and Objects
- 3.3 Scheduling Iterations and Job Flow
- 3.4 Configuring the Scheduler
- 3.5 Credential Overview
3.1 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.

Moab checks for critical config files in directories in this order:

- /opt/moab/
- /opt/moab/etc/
- /etc/

* $(MOABHOMEDIR) (default is /opt/moab and can be modified via the --with-homedir parameter during ./configure)
  - .moab.ck - checkpoint file
  - .moab.pid - lock file
  - moab.lic - license file
  - contrib - directory containing contributed code and plug-ins
  - docs - directory for documentation
  - 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
  - spool - directory for temporary files
  - 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)
  - bin - directory for client commands
    - {showq, setres, ...} - client commands
  - 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. (Example: events.Sat_Oct_10_2009). Additionally, a checkpoint file, .moab.ck, and lock file, .moab.pid, are maintained in the Moab home directory.

3.1.1 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.
See Also

- Commands Overview
- Installation
3.2 Scheduling Environment

- 3.2.1 Scheduling Objects
  - 3.2.1.1 Jobs
    - 3.2.1.1.1 Job States
    - 3.2.1.1.2 Requirement (or Req)
  - 3.2.1.2 Nodes
  - 3.2.1.3 Advance Reservations
  - 3.2.1.4 Policies
  - 3.2.1.5 Resources
  - 3.2.1.6 Task
  - 3.2.1.7 PE
  - 3.2.1.8 Class (or Queue)
  - 3.2.1.9 Resource Manager (RM)
- 3.2.2 Scheduling Dictionary

3.2.1 Scheduling Objects

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.

3.2.1.1 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.

3.2.1.1.1 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>
<thead>
<tr>
<th>State</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>deferred</td>
<td>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 DEFCOUNT times before the job is placed in batch hold.</td>
</tr>
<tr>
<td>hold</td>
<td>Job is idle and is not eligible to run due to a user, (system) administrator, or batch system hold (also, batchhold, systemhold, userhold).</td>
</tr>
<tr>
<td>idle</td>
<td>Job is currently queued and eligible to run but is not executing (also, notqueued).</td>
</tr>
<tr>
<td>migrated</td>
<td>Job that has been migrated to a remote peer, but have not yet begun execution. Migrated jobs show can show up on any grid system</td>
</tr>
<tr>
<td>staged</td>
<td>Job has been migrated to another scheduler but has not yet started executing. Staged jobs show up only when using data staging.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Execution States</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>Definition</td>
</tr>
<tr>
<td>deferred</td>
<td>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 DEFCOUNT times before the job is placed in batch hold.</td>
</tr>
<tr>
<td>hold</td>
<td>Job is idle and is not eligible to run due to a user, (system) administrator, or batch system hold (also, batchhold, systemhold, userhold).</td>
</tr>
<tr>
<td>idle</td>
<td>Job is currently queued and eligible to run but is not executing (also, notqueued).</td>
</tr>
<tr>
<td>migrated</td>
<td>Job that has been migrated to a remote peer, but have not yet begun execution. Migrated jobs show can show up on any grid system</td>
</tr>
<tr>
<td>staged</td>
<td>Job has been migrated to another scheduler but has not yet started executing. Staged jobs show up only when using data staging.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution States</td>
<td></td>
</tr>
</tbody>
</table>
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.
canceling | Job has been canceled and is in process of cleaning up.

<table>
<thead>
<tr>
<th>State</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>completed</td>
<td>Job has completed running without failure.</td>
</tr>
<tr>
<td>removed</td>
<td>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.</td>
</tr>
<tr>
<td>vacated</td>
<td>Job canceled after partial execution due to a system failure.</td>
</tr>
</tbody>
</table>

### 3.2.1.2 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:

- **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.

### 3.2.1.3 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.

### 3.2.1.3 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.

### 3.2.1.4 Policies

A configuration file specifies policies controls how and when jobs start. Policies include job prioritization, fairness policies, fairshare configuration policies, and scheduling policies.

### 3.2.1.5 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:

- 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.

### 3.2.1.6 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.

### 3.2.1.7 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:

\[
PE = \text{MAX} \left( \frac{\text{ProcsRequestedByJob}}{\text{TotalConfiguredProcs}}, \frac{\text{MemoryRequestedByJob}}{\text{TotalConfiguredMemory}}, \frac{\text{DiskRequestedByJob}}{\text{TotalConfiguredDisk}}, \frac{\text{SwapRequestedByJob}}{\text{TotalConfiguredSwap}} \right) \times \text{TotalConfiguredProcs}
\]

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:

\[
PE = \text{MAX} \left( \frac{2}{(100 \times 4)}, \frac{768}{(100 \times 1024)} \right) \times (100 \times 4) = 3.
\]

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.

### 3.2.1.8 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Job</td>
<td>A queue may be associated with a default job duration, default size, or default resource requirements.</td>
</tr>
<tr>
<td>Attributes</td>
<td></td>
</tr>
<tr>
<td>Host Constraints</td>
<td>A queue may constrain job execution to a particular set of hosts.</td>
</tr>
<tr>
<td>Job Constraints</td>
<td>A queue may constrain the attributes of jobs that may submitted, including setting limits such as max wallclock time and minimum number of processors.</td>
</tr>
<tr>
<td>Access List</td>
<td>A queue may constrain who may submit jobs into it based on such things as user lists and group lists.</td>
</tr>
<tr>
<td>Special Access</td>
<td>A queue may associate special privileges with jobs including adjusted job priority.</td>
</tr>
</tbody>
</table>

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.

### 3.2.1.9 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.
3.2.2 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 QoS's. 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.)
<table>
<thead>
<tr>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition:</strong> A processing unit. A processor is a consumable resource. Nodes typically consist of one or more processors. (same as CPU)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quality of Service (QoS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition:</strong> An object that provides special services, resources, and so forth.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition:</strong> (see Class)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reservation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition:</strong> An object that reserves a specific collection or resources for a specific timeframe for use by jobs that meet specific conditions.</td>
</tr>
</tbody>
</table>

| 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. |

<table>
<thead>
<tr>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition:</strong> Hardware, generic resources such as software, and features available on a node, including memory, disk, swap, and processors.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resource, Available</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition:</strong> 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).</td>
</tr>
</tbody>
</table>

| 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. |

| Node cl003's resources available for user jsmith: |
| - 2 processors |
| - 392 MB memory |

| Node cl003's resources available to a user other than jsmith: |
| - 1 processor |
| - 142 MB memory |

<table>
<thead>
<tr>
<th>Resource, Configured</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition:</strong> The total amount of consumable resources that are available on a compute node for use by job tasks.</td>
</tr>
</tbody>
</table>
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.

Node cl003's configured resources:
- 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 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.

Node cl003's dedicated resources are:
- 1 processor
- 250 MB memory

Resource, Utilized

Definition: All consumable resources actually used by all job tasks running on the compute node.
**Example:** Node cl003 has 4 processors and 512 MB of memory. It is executing 2 tasks of job cliserver.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.

Node cl003’s utilized resources are:
- 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.
3.3 Scheduling Iterations and Job Flow

- 3.3.1 Scheduling Iterations
  - 3.3.1.1 Update State Information
  - 3.3.1.2 Handle User Requests
  - 3.3.1.3 Perform Next Scheduling Cycle
- 3.3.2 Detailed Job Flow
  - 3.3.2.1 Determine Basic Job Feasibility
  - 3.3.2.2 Prioritize Jobs
  - 3.3.2.3 Enforce Configured Throttling Policies
  - 3.3.2.4 Determine Resource Availability
  - 3.3.2.5 Allocate Resources to Job
  - 3.3.2.6 Distribute Jobs Tasks Across Allocated Resources
  - 3.3.2.7 Launch Job

3.3.1 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

3.3.1.1 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.

3.3.1.2 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.

3.3.1.3 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.

3.3.2 Detailed Job Flow

3.3.2.1 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.

3.3.2.2 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.

### 3.3.2.3 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.

### 3.3.2.4 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.

\[
R.\text{Available} = R.\text{Configured} - \text{MAX}(R.\text{Dedicated}, R.\text{Utilized}).
\]

The RESOURCEAVAILABILITYPOLICY parameter can be modified to adjust this behavior.

### 3.3.2.5 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.

### 3.3.2.6 Distribute Jobs Tasks Across Allocated Resources

With the resources selected, Moab then maps job tasks to the actual resources. This distribution of tasks is typically based on simple task distribution algorithms such as round-robin or max blocking, but can also incorporate parallel language (such as MPI and PVM) library-specific patterns used to minimize interprocess communication overhead.

### 3.3.2.7 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.
3.4 Configuring the Scheduler

- 3.4.1 Adjusting Server Behavior
  - 3.4.1.1 Logging
  - 3.4.1.2 Checkpointing
  - 3.4.1.3 Client Interface
  - 3.4.1.4 Scheduler Mode

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 3` will 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.

### 3.4.1 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.

#### 3.4.1.1 Logging

Moab provides extensive and highly configurable logging facilities controlled by parameters.

- **LOGDIR** - Indicates directory for log files.
- **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.

#### 3.4.1.2 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.

- **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.

#### 3.4.1.3 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.

3.4.1.4 Scheduler Mode

The scheduler mode of operation is controlled by setting the MODE attribute of the SCHEDCFG parameter. The following modes are allowed:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive</td>
<td>Moab interactively confirms each scheduling action before taking any steps. (See interactive mode overview for more information.)</td>
</tr>
<tr>
<td>Monitor</td>
<td>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.)</td>
</tr>
<tr>
<td>Normal</td>
<td>Moab actively schedules workload according to mission objectives and policies; it creates reservations; starts, cancels, preempts, and modifies jobs; and takes other scheduling actions.</td>
</tr>
<tr>
<td>Simulation</td>
<td>Moab obtains workload and resource information from specified simulation trace files and schedules the defined virtual environment.</td>
</tr>
<tr>
<td>Singlestep</td>
<td>Moab behaves as in NORMAL mode but will only schedule a single iteration and then exit.</td>
</tr>
<tr>
<td>Slave</td>
<td>Moab behaves as in NORMAL mode but will only start a job when explicitly requested by a trusted grid peer service or administrator.</td>
</tr>
<tr>
<td>Test</td>
<td>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.)</td>
</tr>
</tbody>
</table>

See Also

- Initial Configuration
- Adding #INCLUDE files to moab.cfg
3.5 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.

- 3.5.1 General Credential Attributes
- 3.5.2 User Credential
- 3.5.3 Group Credential
- 3.5.4 Account (or Project) Credential
- 3.5.5 Class (or Queue) Credential
- 3.5.6 QoS Credential

3.5.1 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.

3.5.1.1 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, QTWEIGHT, and XFWEIGHT.

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
```

3.5.1.2 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.

3.5.1.3 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.

3.5.1.4 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.

<table>
<thead>
<tr>
<th>Credential</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account</td>
<td>ALIST</td>
<td>(allows credential to access specified list of accounts)</td>
</tr>
<tr>
<td>Partition</td>
<td>PLIST</td>
<td>(allows credential to access specified list of partitions)</td>
</tr>
<tr>
<td>QoS</td>
<td>QLIST</td>
<td>(allows credential to access specified list of QoS's)</td>
</tr>
</tbody>
</table>

Example

```
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 the *DEF attribute (shown in the following table) to specify the default credential or partition for a particular credential.

<table>
<thead>
<tr>
<th>Credential</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account</td>
<td>ADEF</td>
<td>(specifies default account)</td>
</tr>
<tr>
<td>Class</td>
<td>CDEF</td>
<td>(specifies default class)</td>
</tr>
<tr>
<td>QoS</td>
<td>QDEF</td>
<td>(specifies default QoS)</td>
</tr>
</tbody>
</table>

Example

```
# 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
```
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:

<table>
<thead>
<tr>
<th>Credential</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td></td>
</tr>
<tr>
<td>Account, Group, QoS</td>
<td>MEMBERULIST</td>
</tr>
<tr>
<td>Class</td>
<td>EXCLUDEUSERLIST and REQUIREDUSERLIST</td>
</tr>
</tbody>
</table>

**Example**

```
# account omega3 can only be accessed by users johnh, stevek, jenp
ACCOUNTCFG[omega3] MEMBERULIST=johnh,stevek,jenp
```

**Example 1: 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 PDEF=pluto
```

**Example 2: 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: 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
```

### 3.5.1.5 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
```
### 3.5.1.6 Job Defaults, Credential State, and General Configuration

Credentials may apply defaults and force job configuration settings via the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COMMENT</strong></td>
<td>Associates a comment string with the target credential.</td>
<td>USERCFG[steve] COMMENT='works for boss, provides good service'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLASSCFG[i3] COMMENT='queue for I/O intensive workload'</td>
</tr>
<tr>
<td><strong>HOLD</strong></td>
<td>Specifies a hold should be placed on all jobs associated with the target credential.</td>
<td>GROUPCFG[bert] HOLD=yes</td>
</tr>
<tr>
<td><strong>JOBFLAGS</strong></td>
<td>Assigns the specified job flag to all jobs with the associated credential.</td>
<td>CLASSCFG[batch] JOBFLAGS=suspendable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QOSCFG[special] JOBFLAGS=restartable</td>
</tr>
<tr>
<td><strong>NOSUBMIT</strong></td>
<td>Specifies whether jobs belonging to this credential can submit jobs using msub.</td>
<td>ACCOUNTCFG[general] NOSUBMIT=TRUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLASSCFG[special] NOSUBMIT=TRUE</td>
</tr>
<tr>
<td><strong>OVERRUN</strong></td>
<td>Specifies the amount of time a job may exceed its wallclock limit before being terminated. (Only applies to user and class credentials.)</td>
<td>CLASSCFG[bigmem] OVERRUN=00:15:00</td>
</tr>
<tr>
<td><strong>VARIABLE</strong></td>
<td>Specifies attribute-value pairs associated with the specified credential. These variables may be used in triggers and other interfaces to modify system behavior.</td>
<td>GROUPCFG[staff] VARIABLE='nocharge=true'</td>
</tr>
</tbody>
</table>

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.
3.5.2 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

3.5.2.1 Role

Moab supports role-based authorization, mapping particular roles to collections of specific users. See the Security section for more information.

3.5.2.1 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.

```plaintext
USERCFG[sally] EMAILADDRESS=sally@acme.com
```

3.5.2.2 Disable Moab User Email

You can disable Moab email notifications for a specific user.

```plaintext
USERCFG[john] NOEMAIL=TRUE
```

3.5.3 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.

3.5.4 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: MANAGERS Attribute

<table>
<thead>
<tr>
<th>ACCOUNTCFG[general]</th>
<th>MANAGERS=ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNTCFG[special]</td>
<td>MANAGERS=stevep</td>
</tr>
</tbody>
</table>

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: 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 |

3.5.5 Class Credential

- 3.5.5.1 Class Job Defaults
- 3.5.5.2 Per Job Min/Max Limits
- 3.5.5.3 Resource Access
- 3.5.5.4 Class Membership Constraints
- 3.5.5.5 Attributes Enabling Class Access to Other Credentials
- 3.5.5.6 Special Class Attributes (such as Managers and Job Prologs)
- 3.5.5.7 Setting Default Classes
- 3.5.5.8 Creating a Remap Class
- 3.5.5.9 Class Attribute Overview
- 3.5.5.10 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.

3.5.5.1 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT.ATTR</td>
<td>Job Attribute</td>
</tr>
</tbody>
</table>
### 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

```
CLASSCFG[batch] DEFAULT.DISK=200MB DEFAULT.FEATURES=prod
DEFAULT.WCLIMIT=1:00:00
CLASSCFG[debug] DEFAULT.FEATURES=debug DEFAULT.WCLIMIT=00:05:00
```

### 3.5.5.2 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.

<table>
<thead>
<tr>
<th>Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX.CPUTIME</td>
<td>Max Allowed Utilized CPU Time</td>
</tr>
<tr>
<td>MAX.NODE</td>
<td>Max Allowed Node Count</td>
</tr>
<tr>
<td>MAX.PROC</td>
<td>Max Allowed Processor Count</td>
</tr>
<tr>
<td>MAX.PS</td>
<td>Max Requested Processor-Seconds</td>
</tr>
<tr>
<td>MIN.NODE</td>
<td>Min Allowed Node Count</td>
</tr>
<tr>
<td>MIN.PROC</td>
<td>Min Allowed Processor Count</td>
</tr>
<tr>
<td>MIN.PS</td>
<td>Min Requested Processor-Seconds</td>
</tr>
<tr>
<td>MIN.TPN</td>
<td>Min Tasks Per Node</td>
</tr>
<tr>
<td>MIN.WCLIMIT</td>
<td>Min Requested Wallclock Limit</td>
</tr>
<tr>
<td>MAX.WCLIMIT</td>
<td>Max Requested Wallclock Limit</td>
</tr>
</tbody>
</table>

The parameters listed in the preceding table are for classes only, 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.
3.5.5.3 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.

3.5.5.4 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.

3.5.5.5 Attributes Enabling Class Access to Other Credentials

Classes may be configured to allow jobs to access other credentials such as QoS’s and Accounts. This is accomplished using the QDEF, QLIST, ADEF, and ALIST attributes.

3.5.5.6 Special Class Attributes

The class object also possesses a few unique attributes including JOBPROLOG, JOBEPILOG, JOBTRIGGER, 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:

- $TIME - Time that the trigger launches
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**

```plaintext
# 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
```

**JOEBIPOLOG**

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, signalling 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.

**JOBTRIGGER**

Job triggers can be directly associated with jobs submitted into a class using the **JOBTRIGGER** attribute. Job triggers are described using the standard trigger description language specified in the Trigger overview section. In the example that follows, users submitting jobs to the class *debug* will be notified with a descriptive message anytime their job is preempted.

```plaintext
CLASSCFG[batch] JOBTRIGGER=
  atype=exec, etype=preempt, action
  =$HOME/tools/preemptnotify.pl $JOBID $OWNER $HOSTNAME
```

**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 **FALSE**. For all jobs outside of the specified classes, allocation enforcement will continue to be enforced.

```plaintext
# do not enforce allocations on low priority and debug jobs
CLASSCFG[lowprio] DISABLEAM=TRUE
CLASSCFG[debug] DISABLEAM=TRUE
```
3.5.5.7 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.

3.5.5.8 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. In the example that follows, a job requesting 4 processors and the node feature `fast` are assigned to the class `quick`.

```
# 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. Also note that OpenPBS and TORQUE 1.x support dynamic job queue modification, but this change is not persistent and will be lost if pbs_server is restarted.
- 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.

3.5.5.9 Class Attribute Overview

The following table enumerates the different parameters for `CLASSCFG`:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN.PROC</td>
<td>Minimum processors</td>
</tr>
<tr>
<td>MAX.PROC</td>
<td>Maximum processors</td>
</tr>
<tr>
<td>MIN.WCLIMIT</td>
<td>Minimum wall clock limit</td>
</tr>
<tr>
<td>MAX.WCLIMIT</td>
<td>Maximum wall clock limit</td>
</tr>
<tr>
<td>REQ.FEATURES</td>
<td>Requested features</td>
</tr>
<tr>
<td>REQ.FLAGS=INTERACTIVE</td>
<td>Requested flags, interactive</td>
</tr>
<tr>
<td>REQUIREDUSERLIST</td>
<td>Required user list</td>
</tr>
</tbody>
</table>

...
<table>
<thead>
<tr>
<th><strong>DEFAULT.ATTR</strong></th>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;ATTRIBUTE&gt;[,&lt;ATTRIBUTE&gt;]</code>...</td>
</tr>
<tr>
<td>Description:</td>
<td>One or more comma-delimited generic job attributes.</td>
</tr>
<tr>
<td>Example:</td>
<td>---</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>DEFAULT.DISK</strong></th>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;INTEGER&gt;</code></td>
</tr>
<tr>
<td>Description:</td>
<td>Default amount of requested disk space.</td>
</tr>
<tr>
<td>Example:</td>
<td>---</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>DEFAULT.EXT</strong></th>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;STRING&gt;</code></td>
</tr>
<tr>
<td>Description:</td>
<td>Default job RM extension.</td>
</tr>
<tr>
<td>Example:</td>
<td>---</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>DEFAULTFEATURES</strong></th>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Comma-delimited list of features.</td>
</tr>
<tr>
<td>Description:</td>
<td>Default list of requested node features (a.k.a, node properties). This only applies to compute resource reqs.</td>
</tr>
<tr>
<td>Example:</td>
<td>---</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>DEFAULT.GRES</strong></th>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;STRING&gt;[&lt;COUNT&gt;] [,&lt;STRING&gt;[&lt;COUNT&gt;]]...</code></td>
</tr>
<tr>
<td>Description:</td>
<td>Default list of per task required consumable generic resources.</td>
</tr>
<tr>
<td>Example:</td>
<td>CLASSCFG[viz] DEFAULT.GRES=viz:2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>DEFAULT.MEM</strong></th>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;INTEGER&gt;</code> (in MB)</td>
</tr>
<tr>
<td>Description:</td>
<td>Default amount of requested memory.</td>
</tr>
<tr>
<td>Example:</td>
<td>---</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>DEFAULT.NODE</strong></th>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;INTEGER&gt;</code></td>
</tr>
<tr>
<td>Description:</td>
<td>Default required node count.</td>
</tr>
<tr>
<td>Example:</td>
<td>---</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>DEFAULT.NODESET</strong></th>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;SETTYPE&gt;::&lt;SETATTR&gt;[::&lt;SETLIST&gt;[,&lt;SETLIST&gt;]]...</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>Default node set.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td>CLASSCFG[amd] DEFAULT.NODESET=ONEOF:FEATURE:ATHLON,OPTERON</td>
</tr>
</tbody>
</table>

**DEFAULT.PROC**

<table>
<thead>
<tr>
<th><strong>Format</strong></th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Default number of requested processors.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>---</td>
</tr>
</tbody>
</table>

**DEFAULT.TPN**

<table>
<thead>
<tr>
<th><strong>Format</strong></th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Default number of tasks per node.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>---</td>
</tr>
</tbody>
</table>

**DEFAULT.WCLIMIT**

<table>
<thead>
<tr>
<th><strong>Format</strong></th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Default wallclock limit.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>---</td>
</tr>
</tbody>
</table>

**EXCL.FEATURES**

<table>
<thead>
<tr>
<th><strong>Format</strong></th>
<th>Comma- or pipe-delimited list of node features.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Set of excluded (disallowed) features. If delimited by commas, reject job if all features are requested; if delimited by the pipe symbol (</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>CLASSCFG[intel] EXCL.FEATURES=ATHLON,AMD</td>
</tr>
</tbody>
</table>

**EXCL.FLAGS**

<table>
<thead>
<tr>
<th><strong>Format</strong></th>
<th>Comma-delimited list of job flags.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Set of excluded (disallowed) job flags. Reject job if any listed flags are set.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>CLASSCFG[batch] EXCL.FLAGS=INTERACTIVE</td>
</tr>
</tbody>
</table>

**EXCLUDEUSERLIST**

<table>
<thead>
<tr>
<th><strong>Format</strong></th>
<th>Comma-delimited list of users.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>List of users not permitted access to class.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>---</td>
</tr>
</tbody>
</table>

**FORCENODEACCESSPOLICY**

The number of unique users is limited by the Moab Maximum ACL limit, which defaults to 32.
| **CLASSCFG**[batch] FORCENODEACCESSPOLICY=SINGLEJOB |
| **FSCAP** |
| **FSTARGET** |
| **HOSTLIST** |
| **JOBEPILG** |
| **JOBFLAGS** |
| **JOBPROLOG** |
**JOBTRIGGER**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;STRING&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Scheduler level job trigger to be associated with jobs submitted to this class. (See special class attributes.)</td>
</tr>
<tr>
<td>Example:</td>
<td>---</td>
</tr>
</tbody>
</table>

**MANAGERS**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;USER&gt;[,&lt;USER&gt;]...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Users allowed to control, cancel, preempt, and modify jobs within class/queue. (See special class attributes.)</td>
</tr>
<tr>
<td>Example:</td>
<td>CLASSCFG[fast] MANAGERS=root,kerry,e43</td>
</tr>
</tbody>
</table>

**MAXJOB**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Maximum number of jobs allowed in the class.</td>
</tr>
<tr>
<td>Example:</td>
<td>---</td>
</tr>
</tbody>
</table>

**MAXPROCPERNODE**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Maximum number of processors requested per node.</td>
</tr>
<tr>
<td>Example:</td>
<td>---</td>
</tr>
</tbody>
</table>

**MAX.CPUTIME**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Maximum allowed utilized CPU time.</td>
</tr>
<tr>
<td>Example:</td>
<td>---</td>
</tr>
</tbody>
</table>

**MAX.NODE**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Maximum number of requested nodes per job. (Also used when REMAPCLASS is set to correctly route the job.)</td>
</tr>
<tr>
<td>Example:</td>
<td>CLASSCFG[batch] MAX.NODE=64</td>
</tr>
</tbody>
</table>

Deny jobs requesting over 64 nodes access to the class batch.

**MAX.PROC**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Maximum number of requested processors per job. (Also used when REMAPCLASS is set to correctly route the job.)</td>
</tr>
<tr>
<td>Example:</td>
<td>CLASSCFG[small] MAX.PROC[USER]=3,6</td>
</tr>
</tbody>
</table>
### 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.)

**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. (Also used when `REMAPCLASS` is set to correctly route the job.)

**Example:** ---

### MIN.WCLIMIT

**Format:** `[[[DD:]HH:]MM:]SS`

```
CLASSCFG[dev] MIN.WCLIMIT=[96:00:00]
```

---
### 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.

**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 QoS's 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] REQFEATURES=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).

```
The number of unique accounts is limited by the Moab Maximum ACL limit, which defaults to 32.
```

**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).

```
The number of unique users is limited by the Moab Maximum ACL limit, which defaults to 32.
```

**Example:**  
```
CLASSCFG[jasper] REQUIREDUSERLIST=john,u13,steve,guest
```

### REQUIREDQOSLIST

**Format:**  
Comma-delimited list of QoS's

**Description:**  
List of QoS's allowed to access and use a class (analogous to *LIST for other credentials).

```
The number of unique QoS's is limited by the Moab Maximum ACL limit, which defaults to 32.
```

**Example:**  
```
CLASSCFG[jasper] REQUIREDQOSLIST=hi,lo
```

### RMLIST

**Format:**  
[!]<RMID>,[!]<RMID>...  

**Description:**  
List of resource managers that can (or cannot) view or access the class. By default, all resource managers can view and access all queues/classes. If this attribute is specified, only listed resource managers can see the associated queue. If an exclamation point character (!) is specified in the value, then access is granted to all resource managers who are not listed. This feature is most commonly used in grid environments.

**Example:**  
```
CLASSCFG[special] RMLIST=LL,chemgrid
```
3.5.5.10 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
```

3.5.6 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. 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.

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

3.5.6.1 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.

3.5.6.2 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.

3.5.6.3 QoS Privilege Flags

QoS's 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.

3.5.6.4 QoS Charge Rate

Associated with the QoS's 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.

3.5.6.5 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.

See Also

- Identity Manager Interface
- Usage Limits
# Job Attributes/Flags Overview

## Job Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Format</th>
<th>Default</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FLAGS</strong></td>
<td>&lt;FLAG&gt;,&lt;FLAG&gt;...</td>
<td>---</td>
<td>specifies job specific flags</td>
<td><code>FLAGS=ADVRES,DEDICATED</code> (The job should only utilize reserved resources and should only use resources on hosts which can be exclusively dedicated)</td>
</tr>
<tr>
<td><strong>PDEF</strong></td>
<td>&lt;PARTITION_NAME&gt;</td>
<td>[DEFAULT]</td>
<td>specifies the default partition associated with the object.</td>
<td><code>PDEF=P1</code> (The object is assigned the default partition P1)</td>
</tr>
<tr>
<td><strong>PLIST</strong></td>
<td>&lt;PARTITION_NAME&gt;[:&lt;PARTITION_NAME&gt;[^</td>
<td>&amp;]]...</td>
<td>[ALL]</td>
<td>specifies the list of partitions the object can access. If no partition list is specified, the object is granted default access to all partitions.</td>
</tr>
<tr>
<td><strong>QDEF</strong></td>
<td>&lt;QOS_NAME&gt;</td>
<td>[DEFAULT]</td>
<td>specifies the default QOS associated with the object.</td>
<td><code>QDEF=premium</code> (The object is assigned the default QOS premium)</td>
</tr>
</tbody>
</table>
Format:  \(<\text{QOS\_NAME}>[^|&] \[\[|&\]\]\[::<\text{QOS\_NAME}>[^|&]\]\]\[::<\text{QOS\_NAME}>[^|&]\]\]...

Default:  \(<\text{QDEF}>\)

Description: specifies the list of QoS's the object can access. If no QoS list is specified, the object is granted access only to its default partition/

Example:  \(\text{QLIST=premium:express:bottomfeeder}\)

(The object can access any of the 3 QoS's listed)

*Note:  By default, jobs may access QoS's 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 QoS's 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., \(\text{JOBQLIST} = \text{USERQLIST} | \text{GROUPQLIST} | \text{CLASSQLIST}\)). 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:  \(\text{ADVRES}[::<\text{RESID}>]\)

Default:  Use available resources where ever found, whether inside a reservation or not.

Description: specifies the job may only utilize accessible, reserved resources. If \(<\text{RESID}>\) is specified, only resources in the specified reservation may be utilized.

Example:  \(\text{FLAGS=ADVRES:META.1}\)

(The job may only utilize resources located in the \text{META.1} reservation)

**BENCHMARK**

Format:  \(\text{BENCHMARK}\)

Default:  \(\text{N/A}\)

Description: \(\text{N/A}\)

Example:  \(\text{FLAGS=BENCHMARK}\)

**BESTEFFORT**

Format:  \(\text{BESTEFFORT}\)

Default:  \(\text{N/A}\)

Description: \(\text{N/A}\)

Example:  \(\text{FLAGS=BESTEFFORT}\)

**BYNAME**

Format:  \(\text{BYNAME}\)

Default:  \(\text{N/A}\)
<table>
<thead>
<tr>
<th>Flag</th>
<th>Format</th>
<th>Default</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEDICATED</td>
<td>DEDICATED</td>
<td>Use resources according to the global NODEACCESSPOLICY</td>
<td>specifies that the job should not share node resources with tasks from any other job</td>
<td>FLAGS=DEDICATED (The job will only allocate resources from nodes which can be exclusively dedicated to this job)</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>DYNAMIC</td>
<td>If set, active jobs may dynamically grow and shrink based on internal job requests or external scheduler driven requests based on application performance targets. Note: In order for a job to use this flag, the job's associated QOS credential must also have the DYNAMIC flag set within the QFLAGS attribute.</td>
<td>do not allow dynamic allocations for active jobs</td>
<td>FLAGS=DYNAMIC The job will be allowed to dynamically allocate/de-allocate resources according to internal requests or scheduler-specified performance targets.</td>
</tr>
<tr>
<td>IGNIDLEJOBRSV</td>
<td>IGNIDLEJOBRSV</td>
<td>N/A</td>
<td>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.</td>
<td>QOSCFG[standby] JOBFLAGS=IGNIDLEJOBRSV</td>
</tr>
<tr>
<td>NOQUEUE</td>
<td>NOQUEUE</td>
<td>Jobs remain queued until the are able to run</td>
<td>specifies that the job should be removed it is is unable to allocate resources and start execution immediately.</td>
<td>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</td>
</tr>
<tr>
<td><strong>PREEMPTEE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Format:</strong></td>
<td>PREEMPTEE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>Jobs may not be preempted by other jobs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Specifies that the job may be preempted by other jobs which have the PREEMPTOR flag set.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>FLAGS=PREEMPTEE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(The job may be preempted by other jobs which have the 'PREEMPTOR' flag set)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>PREEMPTOR</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
<td>PREEMPTOR</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>Jobs may not preempt other jobs</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Specifies that the job may preempt other jobs which have the PREEMPTEE flag set</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>FLAGS=PREEMPTOR</td>
</tr>
<tr>
<td></td>
<td>(The job may preempt other jobs which have the 'PREEMPTEE' flag set)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>PRESTART</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
<td>PRESTART</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>Jobs are started only after the first scheduling iteration</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Note: used only in simulation mode to pre-populate a system.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>FLAGS=PRESTART</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>RESTARTABLE</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
<td>RESTARTABLE</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>Jobs may not be restarted if preempted</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Specifies jobs can be requeued and later restarted if preempted</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>FLAGS=RESTARTABLE</td>
</tr>
<tr>
<td></td>
<td>(The associated job can be preempted and restarted at a later date)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>SHAREDRESOURCE</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
<td>SHAREDRESOURCE</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>SUSPENDABLE</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
<td>SUSPENDABLE</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>Jobs may not be suspended if preempted.</td>
</tr>
</tbody>
</table>
**Description:** Specifies jobs can be suspended and later resumed if preempted

**Example:**

```javascript
FLAGS=SUSPENDABLE
```

(The associated job can be suspended and resumed at a later date)

---

### SYSTEMJOB

<table>
<thead>
<tr>
<th>Format</th>
<th>SYSTEMJOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Description:** Creates an internal system job that does not require resources.

**Example:**

```javascript
FLAGS=SYSTEMJOB
```

---

### WIDERSVSEARCHALGO

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>---</td>
</tr>
</tbody>
</table>

**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 start time. 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 start time. The flag can be specified at submit time, or you can use `mjobctl -m` to modify the job after it has been submitted.

**Example:**

```bash
> msub -l flags=widersvsearchalgo
> mjobctl -m flags+=widersvsearchalgo job.1
```

---

### See Also

- Setting Per-Credential Job Flags
4.0 Scheduler Commands

- 4.1 Client Overview
- 4.2 Monitoring System Status
- 4.3 Managing Jobs
- 4.4 Managing Reservations
- 4.5 Configuring Policies
- 4.6 End-user Commands
4.1 Client Overview

Moab Workload Manager acts as a server to simple clients, referred to as commands. Each command queries the Moab Workload Manager to retrieve statistics, status or to update a parameter.

⚠️ The Commands Overview lists all available commands.
### 4.2 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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkjob</td>
<td>Displays detailed job information such as job state, resource requirements, environment, constraints, credentials, history, allocated resources, and resource utilization.</td>
</tr>
<tr>
<td>checknode</td>
<td>Displays detailed node information such as node state, resources, attributes, reservations, history, and statistics.</td>
</tr>
<tr>
<td>mdiag -f</td>
<td>Displays summarized fairshare information and any unexpected fairshare configuration.</td>
</tr>
<tr>
<td>mdiag -j</td>
<td>Displays summarized job information and any unexpected job state.</td>
</tr>
<tr>
<td>mdiag -n</td>
<td>Displays summarized node information and any unexpected node state.</td>
</tr>
<tr>
<td>mdiag -p</td>
<td>Displays summarized job priority information.</td>
</tr>
<tr>
<td>resetstats</td>
<td>Resets internal statistics.</td>
</tr>
<tr>
<td>showstats -f</td>
<td>Displays various aspects of scheduling performance across a job duration/job size matrix.</td>
</tr>
<tr>
<td>showq [-r</td>
<td>-i]</td>
</tr>
<tr>
<td>showstats -g</td>
<td>Displays current and historical usage on a per group basis.</td>
</tr>
<tr>
<td>showstats -u</td>
<td>Displays current and historical usage on a per user basis.</td>
</tr>
<tr>
<td>showstats -v</td>
<td>Displays high level current and historical scheduling statistics.</td>
</tr>
</tbody>
</table>
4.3 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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canceljob</td>
<td>Cancels existing job.</td>
</tr>
<tr>
<td>checkjob</td>
<td>Displays job state, resource requirements, environment, constraints, credentials, history, allocated resources, and resource utilization.</td>
</tr>
<tr>
<td>mdiag -j</td>
<td>Displays summarized job information and any unexpected job state.</td>
</tr>
<tr>
<td>releasehold -a</td>
<td>Removes job holds or deferals.</td>
</tr>
<tr>
<td>runjob</td>
<td>Starts job immediately, if possible.</td>
</tr>
<tr>
<td>sethold</td>
<td>Sets hold on job.</td>
</tr>
<tr>
<td>setqos</td>
<td>Sets/modifies QoS of existing job.</td>
</tr>
<tr>
<td>setspri</td>
<td>Adjusts job/system priority of job.</td>
</tr>
</tbody>
</table>

See Also

- Job State Definitions
## 4.4 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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdiag -r</td>
<td>Displays summarized reservation information and any unexpected state.</td>
</tr>
<tr>
<td>mrsvctl</td>
<td>Reservation control.</td>
</tr>
<tr>
<td>mrsvctl -r</td>
<td>Removes reservations.</td>
</tr>
<tr>
<td>mrsvctl -c</td>
<td>Creates an administrative reservation.</td>
</tr>
<tr>
<td>showres</td>
<td>Displays information regarding location and state of reservations.</td>
</tr>
</tbody>
</table>
4.5 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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mschedctl -l</td>
<td>Displays triggers, messages, and settings of all configuration parameters.</td>
</tr>
<tr>
<td>mschedctl</td>
<td>Controls the scheduler (behavior, parameters, triggers, messages).</td>
</tr>
<tr>
<td>mschedctl -m</td>
<td>Modifies system values.</td>
</tr>
</tbody>
</table>
4.6 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').

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canceljob</td>
<td>Cancels existing job.</td>
</tr>
<tr>
<td>checkjob</td>
<td>Displays job state, resource requirements, environment, constraints, credentials, history, allocated resources, and resource utilization.</td>
</tr>
<tr>
<td>msub</td>
<td>Submit a new job.</td>
</tr>
<tr>
<td>releaseres</td>
<td>Releases a user reservation.</td>
</tr>
<tr>
<td>setres</td>
<td>Create a user reservation.</td>
</tr>
<tr>
<td>showbf</td>
<td>Shows resource availability for jobs with specific resource requirements.</td>
</tr>
<tr>
<td>showq</td>
<td>Displays detailed prioritized list of active and idle jobs.</td>
</tr>
<tr>
<td>showstart</td>
<td>Shows estimated start time of idle jobs.</td>
</tr>
<tr>
<td>showstats</td>
<td>Shows detailed usage statistics for users, groups, and accounts, to which the end-user has access.</td>
</tr>
</tbody>
</table>

See Also

- Commands Overview
5.0 Prioritizing Jobs and Allocating Resources

- 5.1 Job Priority
- 5.2 Node Allocation
- 5.3 Node Access Policies
- 5.4 Node Availability
- 5.5 Task Distribution Policies
- 5.6 Scheduling Jobs When VMs Exist
5.1 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.

- 5.1.1 Priority Overview
- 5.1.2 Job Priority Factors
- 5.1.3 Common Priority Usage
- 5.1.4 Prioritization Strategies
- 5.1.5 Manual Priority Management

See Also

- mdiag -p (Priority Diagnostics)
5.1.1 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:

\[ \text{<COMPONENT WEIGHT> * <SUBCOMPONENT WEIGHT> * <PRIORITY SUBCOMPONENT VALUE>} \]

Each priority component contains one or more subcomponents as described in the section titled Priority Component Overview. 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 setspri 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.

See Also

- REJECTNEGPRIOJOBS parameter
5.1.2 Job Priority Factors

- 5.1.2.1 Credential (CRED) Component
- 5.1.2.2 Fairshare (FS) Component
- 5.1.2.3 Resource (RES) Component
- 5.1.2.4 Service (SERVICE) Component
- 5.1.2.5 Target Service (TARG) Component
- 5.1.2.6 Usage (USAGE) Component
- 5.1.2.7 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.

<table>
<thead>
<tr>
<th>Factor</th>
<th>SubFactor</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRED</td>
<td>USER</td>
<td>user-specific priority (See USERCFG)</td>
</tr>
<tr>
<td></td>
<td>GROUP</td>
<td>group-specific priority (See GROUPCFG)</td>
</tr>
<tr>
<td></td>
<td>ACCOUNT</td>
<td>account-specific priority (SEE ACCOUNTCFG)</td>
</tr>
<tr>
<td></td>
<td>QOS</td>
<td>QoS-specific priority (See QOSCFG)</td>
</tr>
<tr>
<td></td>
<td>CLASS</td>
<td>class/queue-specific priority (See CLASSCFG)</td>
</tr>
<tr>
<td>FS</td>
<td>FSUSER</td>
<td>user-based historical usage (See Fairshare Overview)</td>
</tr>
<tr>
<td></td>
<td>FSGROUP</td>
<td>group-based historical usage (See Fairshare Overview)</td>
</tr>
<tr>
<td></td>
<td>FSACCOUNT</td>
<td>account-based historical usage (See Fairshare Overview)</td>
</tr>
<tr>
<td></td>
<td>FSQOS</td>
<td>QoS-based historical usage (See Fairshare Overview)</td>
</tr>
<tr>
<td></td>
<td>FSCLASS</td>
<td>class/queue-based historical usage (See Fairshare Overview)</td>
</tr>
<tr>
<td></td>
<td>FSGUSER</td>
<td>imported global user-based historical usage (See ID Manager and Fairshare Overview)</td>
</tr>
<tr>
<td></td>
<td>FSGGROUP</td>
<td>imported global group-based historical usage (See ID Manager and Fairshare Overview)</td>
</tr>
<tr>
<td></td>
<td>FSGACCOUNT</td>
<td>imported global account-based historical usage (See ID Manager and Fairshare Overview)</td>
</tr>
<tr>
<td></td>
<td>FSJPU</td>
<td>current active jobs associated with job user</td>
</tr>
<tr>
<td></td>
<td>FSPPU</td>
<td>current number of processors allocated to active jobs associated with job user</td>
</tr>
<tr>
<td></td>
<td>FSPSPU</td>
<td>current number of processor-seconds allocated to active jobs associated with job user</td>
</tr>
<tr>
<td></td>
<td>WCACCURACY</td>
<td>user’s current historical job wallclock accuracy calculated as total processor-seconds dedicated / total processor-seconds requested</td>
</tr>
<tr>
<td>RES (requested job resources)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NODE number of nodes requested</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROC number of processors requested</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM total real memory requested (in MB)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWAP total virtual memory requested (in MB)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISK total local disk requested (in MB)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS total processor-seconds requested</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PE total processor-equivalent requested</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WALLTIME total walltime requested (in seconds)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUEUETIME time job has been queued (in minutes)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XFACTOR minimum job expansion factor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYPASS number of times job has been bypassed by backfill</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STARTCOUNT number of times job has been restarted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEADLINE proximity to job deadline</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPVIOLATION Boolean indicating whether the active job violates a soft usage limit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USERPRIO user-specified job priority</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERV (current service levels)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TARGET (target service levels)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TARGETQUEUETIME time until queuetime target is reached (exponential)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TARGETXFACOR distance to target expansion factor (exponential)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USAGE (consumed resources -- active jobs only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONSUMED processor-seconds dedicated to date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REMAINING processor-seconds outstanding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HUNGER processors needed to balance a dynamic job</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERCENT percent of required walltime consumed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXECUTIONTIME seconds since job started</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTR (job attribute-based prioritization)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTRATTR Attribute priority if specified job attribute is set (attributes may be user-defined or one of preemt, or preemptee). Default is 0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTRSTATE Attribute priority if job is in specified state (see Job States). Default is 0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTRGRES Attribute priority if a generic resource is requested. Default is 0.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*CAP parameters (FSCAP) 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.
5.1.2.1 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:

\[
\text{Priority} += \text{CREDWEIGHT} \times ( \\
\text{USERWEIGHT} \times \text{Job.User.Priority} + \\
\text{GROUPWEIGHT} \times \text{Job.Group.Priority} + \\
\text{ACCOUNTWEIGHT} \times \text{Job.Account.Priority} + \\
\text{QOSWEIGHT} \times \text{Job.Qos.Priority} + \\
\text{CLASSWEIGHT} \times \text{Job.Class.Priority})
\]

All user, group, account, QoS, and class weights are specified by setting the \text{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:

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREDWEIGHT</td>
<td>1</td>
</tr>
<tr>
<td>USERWEIGHT</td>
<td>1</td>
</tr>
<tr>
<td>GROUPWEIGHT</td>
<td>1</td>
</tr>
<tr>
<td>USERCFG[john]</td>
<td>PRIORITY=2000</td>
</tr>
<tr>
<td>USERCFG[paul]</td>
<td>PRIORITY=-1000</td>
</tr>
<tr>
<td>GROUPCFG[staff]</td>
<td>PRIORITY=1000</td>
</tr>
</tbody>
</table>

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.

5.1.2.2 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:

\[
\text{Priority} += \text{FSWEIGHT} \times \text{MIN(FSCAP, (} \\
\text{FSUSERWEIGHT} \times \Delta\text{UserFSUsage} + \\
\text{FSGROUPWEIGHT} \times \Delta\text{GroupFSUsage} + \\
\text{FSACCOUNTWEIGHT} \times \Delta\text{AccountFSUsage} + \\
\text{FSQOSWEIGHT} \times \Delta\text{QosFSUsage} + \\
\text{FSCLASSWEIGHT} \times \Delta\text{ClassFSUsage} + \\
\text{FSJPUWEIGHT} \times \text{ActiveUserJobs} + \\
\text{FSPPUWEIGHT} \times \text{ActiveUserProcs} + \\
\text{FSSPSUWEIGHT} \times \text{ActiveUserPS} + \\
\text{WCACCURACYWEIGHT} \times \text{UserWCAccuracy}))
\]

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 either 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.

5.1.2.3 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 when you try to put in the golf balls 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.

### 5.1.2.4 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> )
```

### 5.1.2.4.1 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 JOBPRIOACCURUAPOLICY 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
JOBPRIOACCRLPOLICY 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.

5.1.2.4.2 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:

\[
XFACTOR = 1 + \frac{\text{<QUEUETIME>}}{\text{<EXECUTIONTIME>}}
\]

However, a couple of aspects of this calculation make its use more difficult. First, the length of time the job
will actually run—\(<\text{EXECUTIONTIME}>\)—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 \(<\text{QUEUETIME}>\) to prevent various scheduler
abuses. Consequently, Moab uses the following modified equation:

\[
XFACTOR = 1 + \frac{\text{<EFFQUEUETIME>}}{\text{<WALLCLOCKLIMIT>}}
\]

In the equation Moab uses, \(<\text{EFFQUEUETIME}>\) is the effective queue time subject to the JOBPRIOACCRLPOLICY
parameter and \(<\text{WALLCLOCKLIMIT}>\) 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:

<table>
<thead>
<tr>
<th>Job Queue Time</th>
<th>1 hour</th>
<th>2 hours</th>
<th>4 hours</th>
<th>8 hours</th>
<th>16 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFactor for 1 hour job</td>
<td>[1 + \frac{1}{1} = 2.00]</td>
<td>[1 + \frac{2}{1} = 3.00]</td>
<td>[1 + \frac{4}{1} = 5.00]</td>
<td>[1 + \frac{8}{1} = 9.00]</td>
<td>[1 + \frac{16}{1} = 17.0]</td>
</tr>
<tr>
<td>XFactor for 4 hour job</td>
<td>[1 + \frac{1}{4} = 1.25]</td>
<td>[1 + \frac{2}{4} = 1.50]</td>
<td>[1 + \frac{4}{4} = 2.00]</td>
<td>[1 + \frac{8}{4} = 3.00]</td>
<td>[1 + \frac{16}{4} = 5.0]</td>
</tr>
</tbody>
</table>

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 XMINWCLIMIT parameter. If the latter
is used, the calculation for the XFactor subcomponent value becomes:

\[
XFACTOR = 1 + \frac{\text{<EFFQUEUETIME>}}{\text{MAX}(\text{<XMINWCLIMIT>}, \text{<WALLCLOCKLIMIT>})}
\]

Using the XMINWCLIMIT 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.

5.1.2.4.3 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.

5.1.2.4.4 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.

5.1.2.4.5 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.

5.1.2.4.6 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.

5.1.2.4.7 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 if ENABLEPOSUERPRIORITY TRUE is specified in moab.cfg.

5.1.2.5 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:

\[
\text{Priority} += \text{TARGETWEIGHT} \times ( \text{TARGETQUEUETIMEWEIGHT} \times \text{QueueTimeComponent} + \text{TARGETXFACTORWEIGHT} \times \text{XFactorComponent} )
\]

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.

5.1.2.6 Usage (USAGE) Component

The Usage component applies to active jobs only. The priority calculation for the usage priority factor is as follows:

\[
\text{Priority} += \text{USAGWEIGHT} \times ( \text{USAGECONSUMEDWEIGHT} \times \text{ProcSecondsConsumed} + \text{USAGEHUNGERWEIGHT} \times \text{ProcNeededToBalanceDynamicJob} + \text{USAGEREMAININGWEIGHT} \times \text{ProcSecRemaining} + \text{USAGEEXECUTIONTIMWEIGHT} \times \text{SecondsSinceStart} + \text{USAGEPERCENTWEIGHT} \times \text{WalltimePercent} )
\]

5.1.2.7 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**

```
ATTRWEIGHT     100
ATTRATTRWEIGHT 1
ATTRSTATEWEIGHT 1
ATTRGRESWEIGHT  5

# favor suspended jobs 
# disfavor preemptible jobs 
# favor jobs requesting 'matlab'


# map node features to job features

NODETOJOBATTRMAP  gpfs,pvfs

...
```

**See Also**

- 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.
5.1.3 Common Priority Usage

- 5.1.3.1 Credential Priority Factors
- 5.1.3.2 Service Level Priority Factors
- 5.1.3.3 Priority Factor Caps
- 5.1.3.4 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.

5.1.3.1 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 TARGETQUEUETIMWEIGHT).

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.

5.1.3.2 Service Level Priority Factors

In another example, suppose a site administrator wants to do the following:

- favor jobs in the low, medium, and high QoS's 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
QUEUETIMWEIGHT  10
TARGETQUEUETIMWEIGHT 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, QUEUEUTIMEWEIGHT was explicitly set to 10, overriding its default value of 1. Finally, the TARGETQUEUEUTIMEWEIGHT parameter is used in conjunction with the USERCFG line to specify a queue time target of 4 hours.

5.1.3.3 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 Cap Example**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>QOSWEIGHT</td>
<td>1</td>
</tr>
<tr>
<td>QOSCAP</td>
<td>10000</td>
</tr>
<tr>
<td>XFACTORWEIGHT</td>
<td>1</td>
</tr>
<tr>
<td>XFACTORCAP</td>
<td>1000</td>
</tr>
<tr>
<td>QUEUETIMEWEIGHT</td>
<td>10</td>
</tr>
<tr>
<td>QUEUETIMECAP</td>
<td>1000</td>
</tr>
</tbody>
</table>

5.1.3.4 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 ENABLEPOSUERPRIORITY TRUE is specified in moab.cfg.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERPRIOWEIGHT</td>
<td>100</td>
</tr>
</tbody>
</table>

> 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 QoS's 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.

**See Also**

- User Selectable Priority
5.1.4 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.
5.1.5 Manual Job Priority Adjustment

Batch administrators 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 `job.1294` to `1000000005`.

For more information, see Common Priority Usage - End-user Adjustment.
5.2 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 \texttt{NODEALLOCATIONPOLICY}. There are multiple node allocation policies to choose from allowing selection based on reservation constraints, node configuration, resource usage, preferences, and other factors. You can specify these policies with a system-wide default value or on a per-job basis. Please note that \texttt{LASTAVAILABLE} is the default policy.

Available algorithms are described in detail in the following sections and include \texttt{FIRSTAVAILABLE}, \texttt{LASTAVAILABLE}, \texttt{PRIORITY}, \texttt{CPULOAD}, \texttt{MINRESOURCE}, \texttt{CONTIGUOUS}, \texttt{MAXBALANCE}, \texttt{FASTEST}, and \texttt{LOCAL}.

- 5.2.1 Node Allocation Overview
  - 5.2.1.1 Heterogeneous Resources
  - 5.2.1.2 Shared Nodes
  - 5.2.1.3 Reservations or Service Guarantees
  - 5.2.1.4 Non-flat Network
- 5.2.2 Resource Based Algorithms
  - 5.2.2.1 CPULOAD
  - 5.2.2.2 FIRSTAVAILABLE
  - 5.2.2.3 LASTAVAILABLE
  - 5.2.2.4 PRIORITY
  - 5.2.2.5 MINRESOURCE
  - 5.2.2.6 CONTIGUOUS
  - 5.2.2.7 MAXBALANCE
  - 5.2.2.8 FASTEST
  - 5.2.2.9 LOCAL
- 5.2.3 Time Based Algorithms
- 5.2.4 Specifying Per Job Resource Preferences
  - 5.2.4.1 Specifying Resource Preferences
  - 5.2.4.2 Selecting Preferred Resources

5.2.1 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)

5.2.1.1 Heterogeneous Resources

If the available compute resources have differing configurations, and a subset of the submitted jobs cannot run on all of the nodes, then allocation decisions can significantly affect scheduling performance. For example, a system may be comprised of two nodes, A and B, which are identical in all respects except for RAM; node A has 256 MB and and node B has 1 GB of RAM. Two single processor jobs, X and Y, are submitted, one requesting at least 512 MB of RAM, the other, at least 128 MB. The scheduler could run job X on node A in which case job Y would be blocked until job X completes. A more intelligent approach may be to allocate node B to job X because it has the fewest available resources yet still meets the constraints. This is somewhat of a best fit approach in the configured resource dimension and is essentially what is done by the \texttt{MINRESOURCE} algorithm.

5.2.1.2 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.

```
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

# base interactive node allocation on load and jobs
NODEALLOCATIONPOLICY PRIORITY

NODECFG[interactive01] PRIORITYF='-20*LOAD - JOBCOUNT'
NODECFG[interactive02] PRIORITYF='-20*LOAD - JOBCOUNT'
```

```
/var/spool/torque/server_priv/nodes

interactive01 np=32
interactive02 np=32
compute01 np=4
compute02 np=4
```

```
/var/spool/torque/mom_priv/config

# interactive01
```
5.2.1.3 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.

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.

5.2.1.4 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 differences.

5.2.2 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. The current suite of algorithms is described in what follows.

5.2.2.1 CPULOAD

Nodes are selected that have the maximum amount of available, unused CPU power (\(<\text{# of CPU's}> - \langle\text{CPU load}\rangle\)). 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.

5.2.2.2 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, and very fast algorithm.
5.2.2.3 LASTAVAILABLE

This algorithm selects resources to minimize the amount of time after the job and before the trailing reservation. This algorithm is a best fit in time algorithm that 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.

5.2.2.4 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. Using node allocation priority, you can specify the following priority components:

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADISK</td>
<td>Local disk currently available to batch jobs.</td>
</tr>
<tr>
<td>AMEM</td>
<td>Real memory currently available to batch jobs.</td>
</tr>
<tr>
<td>APPLAFFINITY</td>
<td>Application affinity for the job being evaluated; learning algorithm will select best nodes for a given application and certain size of job based on historical statistics.</td>
</tr>
<tr>
<td>APPLFAILURERATE</td>
<td>Application failure rate for the job being evaluated. The failure rate for a node relative to a job requesting a specific application is computed as a ratio of the number of recorded failures on the node for jobs requesting the application to the total number of recorded jobs on the node that have requested the application.</td>
</tr>
<tr>
<td>APROCS</td>
<td>Processors currently available to batch jobs on node (configured procs - dedicated procs).</td>
</tr>
<tr>
<td>ARCH[&lt;ARCH&gt;]</td>
<td>Processor architecture</td>
</tr>
<tr>
<td>ASWAP</td>
<td>Virtual memory currently available to batch jobs.</td>
</tr>
<tr>
<td>CDISK</td>
<td>Total local disk allocated for use by batch jobs.</td>
</tr>
<tr>
<td>CMEM</td>
<td>Total real memory on node.</td>
</tr>
<tr>
<td>COST</td>
<td>Based on node CHARGERATE.</td>
</tr>
<tr>
<td>CPROCS</td>
<td>Total processors on node.</td>
</tr>
<tr>
<td>CSWAP</td>
<td>Total virtual memory configured on node.</td>
</tr>
<tr>
<td>FEATURE[&lt;FNAME&gt;]</td>
<td>Boolean; specified feature is present on node.</td>
</tr>
<tr>
<td>GMETRIC[&lt;GMNAME&gt;]</td>
<td>Current value of specified generic metric on node.</td>
</tr>
<tr>
<td>JOB_COUNT</td>
<td>Number of jobs currently running on node.</td>
</tr>
<tr>
<td>LOAD</td>
<td>Current 1 minute load average.</td>
</tr>
<tr>
<td>MTFB</td>
<td>Mean time between failure (in seconds).</td>
</tr>
<tr>
<td>NODE_INDEX</td>
<td>Node's nodeindex as specified by the resource manager.</td>
</tr>
<tr>
<td>OS</td>
<td>True if job compute requirements match node operating system.</td>
</tr>
<tr>
<td>PARAPROCS</td>
<td>Processors currently available to batch jobs within partition (configured procs - dedicated procs).</td>
</tr>
<tr>
<td>POWER</td>
<td>TRUE if node is ON.</td>
</tr>
<tr>
<td>PREF</td>
<td>Boolean; node meets job specific resource preferences.</td>
</tr>
</tbody>
</table>
**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 weighted factor cannot exceed 32768. The coefficient, if any, of the RANDOM component must precede, not follow, the component in order to work correctly. For example:

```
100 * RANDOM
```

**RSVAffinity**

Reservation affinity for job being evaluated (1 for positive affinity, 0 for neutral affinity, -1 for negative affinity).

**Speed**

If set, node processor speed (procspeed); otherwise, relative node speed.

**Suspended Job Count**

Number of suspended jobs currently on the node.

**Usage**

Percentage of time node has been running batch jobs since the last statistics initialization.

The node allocation priority function can be specified on a node by node or cluster wide basis. In both cases, the recommended approach is to specify the **PRIORITYF** attribute with the **NODECFG** parameter. Some examples follow.

**Example**: Favor the fastest nodes with the most available memory that are running the fewest jobs.

```
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT] PRIORITYF='SPEED + .01 * AMEM - 10 * JOBCOUNT'
...
```

If spaces are placed within the priority function for readability, the priority function value must be quoted to allow proper parsing.

**Example**: A site has a batch system consisting of two dedicated batchX nodes, as well as numerous desktop systems. The allocation function should favor batch nodes first, followed by desktop systems that are the least loaded and have received the least historical usage.

```
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT] PRIORITYF='-LOAD - 5*USAGE'
NODECFG[batch1] PRIORITY=1000 PRIORITYF='PRIORITY + APROCS'
NODECFG[batch2] PRIORITY=1000 PRIORITYF='PRIORITY + APROCS'
...
```

**Example**: Pack tasks onto loaded nodes first.

```
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT] PRIORITYF=JOBCOUNT
...
```

**Example**: 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]'
...
```
**Example**: Send jobs with a certain size and application to nodes that have historically executed similar jobs in an efficient manner.

```plaintext
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT] PRIORITYF='100*APPLAFFINITY'
...```

### 5.2.2.5 MINRESOURCE

This algorithm prioritizes nodes according to the configured resources on each node. Those nodes with the fewest configured resources that still meet the job’s resource constraints are selected.

### 5.2.2.6 CONTIGUOUS

This algorithm allocates nodes in contiguous (linear) blocks as required by the Compaq RMS system.

### 5.2.2.7 MAXBALANCE

This algorithm 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.

### 5.2.2.8 FASTEST

This algorithm selects nodes in the order of fastest node first order. Nodes are selected by node speed if specified. If node speed is not specified, nodes are selected by processor speed. If neither is specified, nodes are selected in a random order.

### 5.2.2.9 LOCAL

Calls the locally created `contrib` node allocation algorithm.

### 5.2.3 Time Based Algorithms

Time based algorithms allow the scheduler to optimize placement of jobs and reservations in time and are typically of greatest value in systems with the following criteria:

- large backlog
- large number of system or standing reservations
- heavy use of backfill

The `FIRSTAVAILABLE`, `LASTAVAILABLE`, and `PRIORITY` algorithms take into account a node's availability in time and should be considered in such cases.

### 5.2.4 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.

#### 5.2.4.1 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.

5.2.4.2 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:

```plaintext
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:

```plaintext
> qsub -l nodes=4,walltime=1:00:00,pref=feature:fast
```

See Also

- Generic Metrics
- Per Job Node Allocation Policy Specification via Resource Manager Extensions
5.3 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:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHARED</td>
<td>Tasks from any combination of jobs may use available resources.</td>
</tr>
<tr>
<td>SHAREDONLY</td>
<td>Only jobs requesting shared node access may use available resources.</td>
</tr>
<tr>
<td>SINGLEJOB</td>
<td>Tasks from a single job may use available resources.</td>
</tr>
<tr>
<td>SINGLETASK</td>
<td>A single task from a single job may run on the node.</td>
</tr>
<tr>
<td>SINGLEUSER</td>
<td>Tasks from any jobs owned by the same user may use available resources.</td>
</tr>
<tr>
<td>UNIQUEUSER</td>
<td>Any number of tasks from a single job may allocate resources from a node</td>
</tr>
<tr>
<td></td>
<td>but only if the user has no other jobs running on that node.</td>
</tr>
</tbody>
</table>

- This policy is useful in environments where job epilog/prologs scripts are used to clean up processes based on userid.

5.3.1 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.

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
```

See Also
- Per job `naccesspolicy` specification via Resource Manager Extensions
- `JOBNODEMATCHPOLICY` parameter
- `NODEAVAILABILITY` parameter
5.4 Node Availability Policies

- 5.4.1 Node Resource Availability Policies
- 5.4.2 Node Categorization
- 5.4.3 Node Failure/Performance Based Notification
- 5.4.4 Node Failure/Performance Based Triggers
- 5.4.5 Handling Transient Node Failures
- 5.4.6 Reallocating Resources When Failures Occur

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.

5.4.1 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 constitutes a node being
busy. The legal settings are listed in the following table:

<table>
<thead>
<tr>
<th>Availability Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEDICATED</td>
<td>The node is considered busy if dedicated resources equal or exceed configured resources.</td>
</tr>
<tr>
<td>UTILIZED</td>
<td>The node is considered busy if utilized resources equal or exceed configured resources.</td>
</tr>
<tr>
<td>COMBINED</td>
<td>The node is considered busy if either dedicated or utilized resources equal or exceed configured resources.</td>
</tr>
</tbody>
</table>

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
exceeds 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.

5.4.2 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:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Node is healthy and currently executing batch workload.</td>
</tr>
<tr>
<td>BatchFailure</td>
<td>Node is unavailable due to a failure in the underlying batch system (such as a resource manager server or resource manager node daemon).</td>
</tr>
<tr>
<td>Benchmark</td>
<td>Node is reserved for benchmarking.</td>
</tr>
<tr>
<td>EmergencyMaintenance</td>
<td>Node is reserved for unscheduled system maintenance.</td>
</tr>
<tr>
<td>GridReservation</td>
<td>Node is reserved for grid use.</td>
</tr>
<tr>
<td>HardwareFailure</td>
<td>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).</td>
</tr>
<tr>
<td>HardwareMaintenance</td>
<td>Node is reserved for scheduled system maintenance.</td>
</tr>
<tr>
<td>Idle</td>
<td>Node is healthy and is currently not executing batch workload.</td>
</tr>
<tr>
<td>JobReservation</td>
<td>Node is reserved for job use.</td>
</tr>
<tr>
<td>NetworkFailure</td>
<td>Node is unavailable due to a failure in its network adapter or in the switch.</td>
</tr>
<tr>
<td>Other</td>
<td>Node is in an uncategorized state.</td>
</tr>
<tr>
<td>OtherFailure</td>
<td>Node is unavailable due to a general failure.</td>
</tr>
<tr>
<td>PersonalReservation</td>
<td>Node is reserved for dedicated use by a personal reservation.</td>
</tr>
<tr>
<td>Site[1-8]</td>
<td>Site specified usage categorization.</td>
</tr>
<tr>
<td>SoftwareFailure</td>
<td>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).</td>
</tr>
<tr>
<td>SoftwareMaintenance</td>
<td>Node is reserved for software maintenance.</td>
</tr>
<tr>
<td>StandingReservation</td>
<td>Node is reserved by a standing reservation.</td>
</tr>
<tr>
<td>StorageFailure</td>
<td>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).</td>
</tr>
<tr>
<td>UserReservation</td>
<td>Node is reserved for dedicated use by a particular user or group and may or may not be actively executing jobs.</td>
</tr>
<tr>
<td>VPC</td>
<td>Node is reserved for VPC use.</td>
</tr>
</tbody>
</table>

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 `FORCERSVSBTYPE` parameter to require all administrative reservations be associated with a node category at reservation creation time.
Example

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.

### 5.4.3 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.

```plaintext
GEVENTCFG[nodedown] ACTION=notify REARM=00:20:00
```

### 5.4.4 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.

### 5.4.5 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
```

### 5.4.6 Reallocating Resources When Failures Occur

If a failure occurs within a collection of nodes allocated to a job or reservation, Moab can automatically reallocate replacement resources. For jobs, this can be configured with `JOBACTIONONNODEFAILURE`. For reservations, use the `RSVREALLOCPOLICY`.

#### 5.4.6.1 Allocated Resource Failure Policy for Jobs

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:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancel</td>
<td>Cancels job.</td>
</tr>
<tr>
<td>hold</td>
<td>Requeues and holds job.</td>
</tr>
<tr>
<td>ignore</td>
<td>Ignores failure and allows job to continue running.</td>
</tr>
<tr>
<td>migrate</td>
<td>Migrates failed task to new node.</td>
</tr>
<tr>
<td>notify</td>
<td>Notifies administrator and user of failure but takes no further action.</td>
</tr>
<tr>
<td>requeue</td>
<td>Requeues job and allows it to run when alternate resources become available.</td>
</tr>
</tbody>
</table>

**Policy Precedence**

- `cancel` takes precedence over `hold`, `ignore`, `migrate`, `notify`, and `requeue`.
- `hold` takes precedence over `ignore`, `migrate`, `notify`, and `requeue`.
- `ignore` takes precedence over `migrate`, `notify`, and `requeue`.
- `migrate` takes precedence over `notify` and `requeue`.
- `notify` takes precedence over `requeue`.
- `requeue` takes no precedence over other policies.
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:

<table>
<thead>
<tr>
<th>Object</th>
<th>Parameter</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job</td>
<td>resfailpolicy</td>
<td>&gt; qsub -l resfailpolicy=requeue</td>
</tr>
<tr>
<td>Class/Queue</td>
<td>RESFAILPOLICY attribute of CLASSCFG parameter</td>
<td>CLASSCFG[batch] RESFAILPOLICY=CANCEL</td>
</tr>
<tr>
<td>Partition</td>
<td>JOBACTIONNONNODEFailure attribute of PARCFG parameter</td>
<td>PARCFG[web3] JOBACTIONNONNODEFailure=NOTIFY</td>
</tr>
<tr>
<td>System</td>
<td>NODEALLOCRESFAILUREPOLICY parameter</td>
<td>NODEALLOCRESFAILUREPOLICY=MIGRATE</td>
</tr>
</tbody>
</table>

**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.

**See Also**

- 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)
5.5 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. See the TASKDISTRIBUTIONPOLICY parameter for more information.

See Also

- Node Set Overview
- Node Allocation Overview
- TASKDISTPOLICY Resource Manager Extension
5.6 Scheduling Jobs When VMs Exist

Each Job has a VM usage policy. This policy directs how Moab considers physical and virtual nodes when allocating resources for a job. These are the supported policies:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATEPERSISTENTVM</td>
<td>Makes the VM persist after the job completes. The VM is not terminated until the <code>mvmctl -d &lt;vmid&gt;</code> command is used.</td>
</tr>
<tr>
<td>CREATEVM</td>
<td>The job should create a one-time use virtual machine for the job to run on. Any virtual machines created by the job are destroyed when the job is finished. If specified, the job itself must request an OS so an appropriate virtual machine can be provisioned.</td>
</tr>
<tr>
<td>REQUIREPM</td>
<td>States that the job should run only on physical machines.</td>
</tr>
<tr>
<td>REQUIREVM</td>
<td>States that the job should run only on virtual machines that already exist.</td>
</tr>
</tbody>
</table>

If the HIDEVIRTUALNODES parameter is configured with a value of TRANSPARENT, jobs are given a default policy of REQUIREVM. Otherwise they are given a policy of REQUIREPM. These defaults can be overridden by using the extension resource VMUSAGEPOLICY or by setting the policy via a job template. An example of both is given below.

**Examples**

As an extension resource:

```
> msub -l vmusagepolicy=requirepm
```

As a template parameter:

```
JOBCFG[vmjob] VMUSAGE=requirevm
```

The VMUSAGEPOLICY of a job can be viewed by using `checkjob -v`. 
6.0 Managing Fairness - Throttling Policies, Fairshare, and Allocation Management

- 6.1 Fairness Overview
- 6.2 Usage Limits/Throttling Policies
- 6.3 Fairshare
- 6.4 Charging and Allocation Management
- 6.5 Internal Charging Facilities
6.1 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:

<table>
<thead>
<tr>
<th>Job Prioritization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> 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.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>SERVICEWEIGHT 1</td>
</tr>
<tr>
<td>QUEUETIMEWEIGHT 10</td>
</tr>
<tr>
<td>Causes jobs to increase in priority by 10 points for every minute they remain in the queue.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage Limits (Throttling Policies)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> Specifies limits on exactly what resources can be used at any given instant.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>USERCFG[john] MAXJOB=3</td>
</tr>
<tr>
<td>GROUPCFG[DEFAULT] MAXPROC=64</td>
</tr>
<tr>
<td>GROUPCFG[staff] MAXPROC=128</td>
</tr>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fairshare</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> Specifies usage targets to limit resource access or adjust priority based on historical cluster and grid level resource usage.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>USERCFG[steve] FSTARGET=25.0+</td>
</tr>
<tr>
<td>FSWEIGHT 1</td>
</tr>
<tr>
<td>FSUSERWEIGHT 10</td>
</tr>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Allocation Management</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> Specifies long term, credential-based resource usage limits.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>AMCFG[bank] TYPE=GOLD HOST=server.sys.net</td>
</tr>
<tr>
<td>Enables the GOLD allocation management interface. Within the allocation manager, project or</td>
</tr>
</tbody>
</table>
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).

**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/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?

See Also

• Job Prioritization
• Usage Limits (Throttling Policies)
• Fairshare
• Allocation Management
• Quality of Service
• Standing Reservations
• Class/Queue Constraints
6.2 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.

- 6.2.1 Fairness via Throttling Policies
  - 6.2.1.1 Basic Fairness Policies
  - 6.2.1.2 Multi-Dimension Fairness Policies
- 6.2.2 Override Limits
- 6.2.3 Idle Job Limits
- 6.2.4 System Job Limits
- 6.2.5 Hard and Soft Limits
- 6.2.6 Per-partition Limits

6.2.1 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.

6.2.1.1 Basic Fairness Policies

- 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, QoS's may be configured 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.

<table>
<thead>
<tr>
<th>MAXJOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units: # of jobs</td>
</tr>
<tr>
<td>Description: Limits the number of jobs a credential may have active (starting or running) at any given time.</td>
</tr>
</tbody>
</table>

* MAXJOB=0 is not supported. You can, however, achieve similar results by using the HOLD attribute of the USERCFG parameter:

```
USERCFG[john] HOLD=yes
```

<table>
<thead>
<tr>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERCFG[DEFAULT] MAXJOB=8</td>
</tr>
<tr>
<td>GROUPCFG[staff] MAXJOB=2,4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MAXMEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units:</td>
</tr>
<tr>
<td>Description:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERCFG[DEFAULT] MAXMEM=8</td>
</tr>
<tr>
<td>GROUPCFG[staff] MAXMEM=2,4</td>
</tr>
<tr>
<td>Units:</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>Description:</td>
</tr>
<tr>
<td>Example:</td>
</tr>
</tbody>
</table>

**MAXNODE**

<table>
<thead>
<tr>
<th>Units:</th>
<th># of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Limits the total number of compute nodes that can be in use by active jobs at any given time.</td>
</tr>
<tr>
<td>Example:</td>
<td>CLASSCFG[batch] MAXNODE=64</td>
</tr>
</tbody>
</table>

**MAXPE**

<table>
<thead>
<tr>
<th>Units:</th>
<th># of processor equivalents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Limits the total number of dedicated processor-equivalents that can be allocated by active jobs at any given time.</td>
</tr>
<tr>
<td>Example:</td>
<td>QOSCFG[base] MAXPE=128</td>
</tr>
</tbody>
</table>

**MAXPROC**

<table>
<thead>
<tr>
<th>Units:</th>
<th># of processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Limits the total number of dedicated processors that can be allocated by active jobs at any given time.</td>
</tr>
<tr>
<td>Example:</td>
<td>CLASSCFG[debug] MAXPROC=32</td>
</tr>
</tbody>
</table>

**MAXPS**

<table>
<thead>
<tr>
<th>Units:</th>
<th>&lt;# of processors&gt; * &lt;walltime&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>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.</td>
</tr>
<tr>
<td>Example:</td>
<td>USERCFG[DEFAULT] MAXPS=720000</td>
</tr>
</tbody>
</table>

**MAXWC**

<table>
<thead>
<tr>
<th>Units:</th>
<th>job duration [[DD::HH::MM::]SS]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Limits the cumulative remaining walltime a credential may have associated with active jobs. It</td>
</tr>
</tbody>
</table>
behaves identically to the **MAXPS** 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**.

**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.

### 6.2.1.2 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. Moab supports the most commonly used multi-dimensional fairness policies (listed in the table below) using the following format:

```
*CFG[X] <LIMITTYPE>[<CRED>]=<LIMITVALUE>
```

The "*CFG" is one of USERCFG, GROUPCFG, ACCOUNTCFG, QOSC, 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 <CREDTYP>[[:<VALUE>]] with CREDTYP 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
```
Multi-dimensional policies cannot be applied on *DEFAULT* credentials.

The table below lists the 112 currently implemented, multi-dimensional usage limit permutations. The "slmt" stands for "Soft Limit" and "hlmt" stands for "Hard Limit."

<table>
<thead>
<tr>
<th>Multi-Dimension Usage Limit Permutations</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNTCFG[name] MAXJOB[QOS]=hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXJOB[QOS:qosname]=hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXIPROC[QOS]=hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXIPROC[QOS:qosname]=hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXJOB[QOS]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXJOB[QOS:qosname]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXJOB[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXJOB[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXMEM[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXMEM[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXNODE[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXNODE[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXPROC[QOS]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXPROC[QOS:qosname]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXPROC[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXPROC[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXPS[QOS]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXPS[QOS:qosname]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXPS[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXPS[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXWC[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>ACCOUNTCFG[name] MAXWC[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name] MAXJOB[GROUP]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name] MAXJOB[GROUP:groupname]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name] MAXJOB[QOS:qosname]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name] MAXJOB[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name] MAXJOB[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name] MAXMEM[GROUP]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name] MAXMEM[GROUP:groupname]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name] MAXMEM[QOS:qosname]=slmt,hlmt</td>
</tr>
</tbody>
</table>

# allow users steve and bob to use up to 3 and 4 total processors respectively within class
<table>
<thead>
<tr>
<th>CLASSCFG[name]</th>
<th>MAXMEM[USER]=slmt,hlmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXMEM[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXNODE[GROUP]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXNODE[GROUP:groupname]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXNODE[QOS:qosname]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXNODE[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXNODE[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXPROC[GROUP]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXPROC[GROUP:groupname]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXPROC[QOS:qosname]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXPROC[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXPROC[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXPS[GROUP]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXPS[GROUP:groupname]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXPS[QOS:qosname]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXPS[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>CLASSCFG[name]</td>
<td>MAXPS[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXJOB[CLASS:classname]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXJOB[QOS:qosname]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXJOB[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXJOB[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXMEM[CLASS:classname]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXMEM[QOS:qosname]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXMEM[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXMEM[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXNODE[CLASS:classname]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXNODE[QOS:qosname]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXNODE[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXNODE[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXPROC[CLASS:classname]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXPROC[QOS:qosname]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXPROC[USER]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXPROC[USER:username]=slmt,hlmt</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXPS[CLASS:classname]=slmt,hlmt</td>
</tr>
<tr>
<td>Configuration Level</td>
<td>Configuration</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------</td>
</tr>
<tr>
<td><strong>GROUPCFG[name]</strong></td>
<td>MAXPS</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXPS[USER]</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXPS[USER:username]</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXWC</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXWC[QOS:qosname]</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXWC[USER]</td>
</tr>
<tr>
<td>GROUPCFG[name]</td>
<td>MAXWC[USER:username]</td>
</tr>
<tr>
<td><strong>QOSCFG[name]</strong></td>
<td>MAXJOB[ACCT]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXJOB[ACCT:accountname]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXIPROC[ACCT]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXIPROC[ACCT:accountname]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXJOB[USER]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXJOB[USER:username]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXMEM[USER]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXMEM[USER:username]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXNODE[USER]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXNODE[USER:username]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXPROC[ACCT]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXPROC[ACCT:accountname]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXPROC[USER]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXPROC[USER:username]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXPS[ACCT]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXPS[ACCT:accountname]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXPS[USER]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXPS[USER:username]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXWC[USER]</td>
</tr>
<tr>
<td>QOSCFG[name]</td>
<td>MAXWC[USER:username]</td>
</tr>
<tr>
<td><strong>USERCFG[name]</strong></td>
<td>MAXJOB[GROUP]</td>
</tr>
<tr>
<td>USERCFG[name]</td>
<td>MAXJOB[GROUP:groupname]</td>
</tr>
<tr>
<td>USERCFG[name]</td>
<td>MAXMEM[GROUP]</td>
</tr>
<tr>
<td>USERCFG[name]</td>
<td>MAXMEM[GROUP:groupname]</td>
</tr>
<tr>
<td>USERCFG[name]</td>
<td>MAXNODE[GROUP]</td>
</tr>
<tr>
<td>USERCFG[name]</td>
<td>MAXNODE[GROUP:groupname]</td>
</tr>
<tr>
<td>USERCFG[name]</td>
<td>MAXPROC[GROUP]</td>
</tr>
<tr>
<td>USERCFG[name]</td>
<td>MAXPROC[GROUP:groupname]</td>
</tr>
<tr>
<td>USERCFG[name]</td>
<td>MAXPS[GROUP]</td>
</tr>
<tr>
<td>USERCFG[name]</td>
<td>MAXPS[GROUP:groupname]</td>
</tr>
<tr>
<td>USERCFG[name]</td>
<td>MAXWC[GROUP]</td>
</tr>
<tr>
<td>USERCFG[name]</td>
<td>MAXWC[GROUP:groupname]</td>
</tr>
</tbody>
</table>
6.2.2 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 (MAXJOB) to be ignored. Override limits are applied by preceding the limit specification with the capital letter $O$. The following configuration provides an example of this in the last line:

<table>
<thead>
<tr>
<th>UserCFG [steve]</th>
<th>MAXJOB=2</th>
<th>MAXNODE=30</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupCFG [staff]</td>
<td>MAXJOB=5</td>
<td></td>
</tr>
<tr>
<td>ClassCFG [default]</td>
<td>MAXNODE=16</td>
<td></td>
</tr>
<tr>
<td>ClassCFG [batch]</td>
<td>MAXNODE=32</td>
<td></td>
</tr>
<tr>
<td>QosCFG [hiprio]</td>
<td>OMAXJOB=3</td>
<td>OMAXNODE=64</td>
</tr>
</tbody>
</table>

The preceding configuration is identical to the earlier example with the exception of the final QOSCFG line. In this case, the QOSCFG parameter does two things:

- Only 3 $\text{hiprio}$ QoS jobs may run simultaneously.
- $\text{hiprio}$ 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 $\text{steve}$, group $\text{staff}$, class $\text{batch}$, and QoS $\text{hiprio}$.

Such a job will start so long as running it does not lead to any of the following conditions:

- Total nodes used by user $\text{steve}$ does not exceed 64.
- Total active jobs associated with user $\text{steve}$ does not exceed 2.
- Total active jobs associated with group $\text{staff}$ does not exceed 5.
- Total nodes dedicated to class $\text{batch}$ does not exceed 64.
- Total active jobs associated with QoS $\text{hiprio}$ 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.

6.2.3 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 queuetime 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. For example, to limit the number of (eligible) idle jobs a given user could have at once, the following parameter could be used:

| UserCFG [DEFAULT] | MAXIJOB=20 |

As just shown, 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 MAXJOB[USER] and MAXPROC[USER] policies on a per class basis. (See Basic Fairness Policies.)

Example:

<table>
<thead>
<tr>
<th>UserCFG [steve]</th>
<th>MAXJOB=2 MAXINODE=30</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupCFG [staff]</td>
<td>MAXJOB=5</td>
</tr>
<tr>
<td>ClassCFG [DEFAULT]</td>
<td>MAXINODE=16</td>
</tr>
<tr>
<td>QosCFG [hiprio]</td>
<td>MAXJOB=3 MAXINODE=64</td>
</tr>
</tbody>
</table>

### 6.2.4 System Job Limits

System job limits, also known as individual job limits, constrain the quantity of resources that may be requested by any single job. Unlike active and idle job usage limits, system job limits are specified on a global basis. The following table details the associated parameters.

<table>
<thead>
<tr>
<th>Limit</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration</td>
<td>SYSTEMMAXJOBWALLTIME</td>
<td>Limits the maximum requested wallclock time per job.</td>
</tr>
<tr>
<td>processors</td>
<td>SYSTEMMAXPROCPERJOB</td>
<td>Limits the maximum requested processors per job.</td>
</tr>
<tr>
<td>processor-seconds</td>
<td>SYSTEMMAXPROCSECONDPERJOB</td>
<td>Limits the maximum requested processor-seconds per job.</td>
</tr>
</tbody>
</table>

### 6.2.5 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:

<table>
<thead>
<tr>
<th>UserCFG [steve]</th>
<th>MAXJOB=2,4 MAXNODE=15,30</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupCFG [staff]</td>
<td>MAXJOB=2,5</td>
</tr>
<tr>
<td>ClassCFG [DEFAULT]</td>
<td>MAXNODE=16,32</td>
</tr>
<tr>
<td>ClassCFG [batch]</td>
<td>MAXNODE=12,32</td>
</tr>
</tbody>
</table>
Job preemption status can be adjusted based on whether the job violates a soft policy using the `ENABLESPVIOLATIONPREEMPTION` parameter.

### 6.2.6 Per-partition Limits

Per-partition scheduling can set limits and enforce credentials and policies 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`:

```
PERPARTITION_SCHEDULING TRUE
JOBMIGRATEPOLICY JUSTINTIME
```

#### 6.2.6.1 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`:

```
CLASSCFG[DEFAULT] MAXJOB=2
```

#### 6.2.6.2 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
```
See Also

- Managing Job Resource Limit Violations
- RESOURCELIMITPOLICY parameter
- FSTREE parameter (set usage limits within share tree hierarchy)
- Credential Overview
6.3 Fairshare

Fairshare is a mechanism that allows historical resource utilization information to be incorporated into job feasibility and priority decisions. Moab's fairshare implementation allows organizations to set system utilization targets for users, groups, accounts, classes, and QoS levels. You can use both local and global (multi-cluster) fairshare information to make local scheduling decisions.

- 6.3.1 Overview
- 6.3.2 Fairshare Parameters
  - 6.3.2.1 FSPOLICY - Specifying the Metric of Consumption
  - 6.3.2.2 Specifying Fairshare Timeframe
  - 6.3.2.3 Managing Fairshare Data
- 6.3.3 Using Fairshare Information
  - 6.3.3.1 Fairshare Targets
  - 6.3.3.2 Fairshare Caps
  - 6.3.3.3 Priority Based Fairshare
  - 6.3.3.4 Per-Credential Fairshare Weights
  - 6.3.3.5 Extended Fairshare Examples
- 6.3.4 Hierarchical Fairshare/Share Trees
  - 6.3.4.1 Defining the Tree
  - 6.3.4.2 Controlling Tree Evaluation
- 6.3.5 Importing Fairshare Data

6.3.1 Overview

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.

6.3.2 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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSPOLICY</td>
<td>Metric to use when tracking fairshare usage.</td>
</tr>
<tr>
<td>FSINTERVAL</td>
<td>Duration of each fairshare window.</td>
</tr>
<tr>
<td>FSDEPTH</td>
<td>Number of fairshare windows factored into current fairshare utilization.</td>
</tr>
<tr>
<td>FSDECAY</td>
<td>Decay factor applied to weighting the contribution of each fairshare window.</td>
</tr>
</tbody>
</table>

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.
6.3.2.1 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.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEDICATEDPES</td>
<td>Usage tracked by processor-equivalent seconds dedicated to each job. (Useful in dedicated and shared nodes environments).</td>
</tr>
<tr>
<td>DEDICATEDPS</td>
<td>Usage tracked by processor seconds dedicated to each job. (Useful in dedicated node environments.)</td>
</tr>
<tr>
<td>PDEDICATEDPS</td>
<td>Usage tracked by processor seconds dedicated to each job with per node usage scaled by the node processor speed attribute. (Useful in dedicated and shared nodes environments with heterogeneous compute nodes.)</td>
</tr>
<tr>
<td>SDEDICATEDPES</td>
<td>Usage tracked by processor-equivalent seconds dedicated to each job with per node usage scaled by the node speed attribute. (Useful in dedicated and shared nodes environments with heterogeneous compute nodes.)</td>
</tr>
<tr>
<td>UTILIZEDPS</td>
<td>Usage tracked by processor seconds used by each job. (Useful in shared node/SMP environments.)</td>
</tr>
</tbody>
</table>

Example

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

Percentage Based Fairshare

By default, when comparing fairshare usage against fairshare targets, Moab calculates usage as a percentage
of delivered cycles. To change the usage calculation to be based on available cycles, rather than delivered cycles, the percent (%) character can be specified at the end of the FSPOLICY value as in the preceding example.

### 6.3.2.2 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.

<table>
<thead>
<tr>
<th>Decay Factor</th>
<th>Win0</th>
<th>Win1</th>
<th>Win2</th>
<th>Win3</th>
<th>Win4</th>
<th>Win5</th>
<th>Win6</th>
<th>Win7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>0.80</td>
<td>100%</td>
<td>80%</td>
<td>64%</td>
<td>51%</td>
<td>41%</td>
<td>33%</td>
<td>26%</td>
<td>21%</td>
</tr>
<tr>
<td>0.75</td>
<td>100%</td>
<td>75%</td>
<td>56%</td>
<td>42%</td>
<td>31%</td>
<td>23%</td>
<td>17%</td>
<td>12%</td>
</tr>
<tr>
<td>0.50</td>
<td>100%</td>
<td>50%</td>
<td>25%</td>
<td>13%</td>
<td>6%</td>
<td>3%</td>
<td>2%</td>
<td>1%</td>
</tr>
</tbody>
</table>

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.

### 6.3.2.3 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 showfs command located in the tools directory.

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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSINTERVAL</td>
<td>12:00:00</td>
</tr>
<tr>
<td>FSDEPTH</td>
<td>4</td>
</tr>
<tr>
<td>FSDECAY</td>
<td>0.5</td>
</tr>
</tbody>
</table>
Further assume that the fairshare usage intervals have the following usage amounts:

<table>
<thead>
<tr>
<th>Fairshare Interval</th>
<th>Total User john Usage</th>
<th>Total Cluster Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>60</td>
<td>110</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>125</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>150</td>
</tr>
</tbody>
</table>

Based on this information, the current fairshare usage for user john would be calculated as follows:

\[
\text{Usage} = \frac{60 + 0.5^1 \times 0 + 0.5^2 \times 10 + 0.5^3 \times 50}{110 + 0.5^1 \times 125 + 0.5^2 \times 100 + 0.5^3 \times 150}
\]

- 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.

### 6.3.3 Using Fairshare Information

#### 6.3.3.1 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, site administrators can configure how fairshare information impacts scheduling behavior. This is done through specification of fairshare targets. 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**

<table>
<thead>
<tr>
<th>Target Modifier:</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job Impact:</td>
<td>Priority</td>
</tr>
<tr>
<td>Format:</td>
<td>Percentage Usage</td>
</tr>
<tr>
<td>Description:</td>
<td>Adjusts job priority down when usage exceeds target.</td>
</tr>
</tbody>
</table>

**Target type — Floor**

<table>
<thead>
<tr>
<th>Target Modifier:</th>
<th>+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job Impact:</td>
<td>Priority</td>
</tr>
<tr>
<td>Format:</td>
<td>Percentage Usage</td>
</tr>
<tr>
<td>Description:</td>
<td>Adjusts job priority up when usage falls below target.</td>
</tr>
</tbody>
</table>

**Target type — Target**

<table>
<thead>
<tr>
<th>Target Modifier:</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job Impact:</td>
<td>Priority</td>
</tr>
<tr>
<td>Format:</td>
<td>Percentage Usage</td>
</tr>
<tr>
<td>Description:</td>
<td>Adjusts job priority when usage does not meet target.</td>
</tr>
</tbody>
</table>
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
FSIZEIGHT   1
FSUSERWEIGHT 100
USERCFG[john] FSTARGET=16.5+
USERCFG[DEFAULT] FSTARGET=10
...```

### 6.3.3.2 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 QoS's 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.

<table>
<thead>
<tr>
<th>Absolute Cap</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cap Modifier:</td>
<td>^</td>
<td></td>
</tr>
<tr>
<td>Job Impact:</td>
<td>Feasibility</td>
<td></td>
</tr>
<tr>
<td>Format:</td>
<td>Absolute Usage</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Constrains job eligibility as an absolute quantity measured according to the scheduler charge metric as defined by the FSPOLICY parameter</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relative Cap</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cap Modifier:</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>Job Impact:</td>
<td>Feasibility</td>
<td></td>
</tr>
<tr>
<td>Format:</td>
<td>Percentage Usage</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Constrains job eligibility as a percentage of total delivered cycles measured according to the scheduler charge metric as defined by the FSPOLICY parameter.</td>
<td></td>
</tr>
</tbody>
</table>

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
FSIZEINTERVAL 12:00:00
FSDEPTH 14
ACCOUNTCFG[marketing] FSCAP=16500^ ACCOUNTCFG[DEFAULT] FSCAP=10
...```
6.3.3.3 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 \texttt{FSWEIGHT} component weight parameter and the target type subcomponent weight (such as \texttt{FSUSERWEIGHT} and \texttt{FSGROUPWEIGHT}) be specified.

If these weights are not set, the fairshare mechanism will be enabled but have no effect on scheduling behavior. See the \texttt{Job Priority Factor Overview} for more information on setting priority weights.

Example

```plaintext
# set relative component weighting
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+

FSWEIGHT 1000
ACCOUNTCFG[orion1] FSWEIGHT=100
ACCOUNTCFG[orion2] FSWEIGHT=200
ACCOUNTCFG[orion3] FSWEIGHT=-100
GROUPCFG[staff] FSWEIGHT=10
```

Job preemption status can be adjusted based on whether the job violates a fairshare target using the \texttt{ENABLEFSVIOLATIONPREEMPTION} parameter.

6.3.3.4 Credential Specific Fairshare Weights

Credential-specific fairshare weights can be set using the \texttt{FSWEIGHT} attribute of the ACCOUNT, GROUP, and QOS credentials as in the following example:

```plaintext
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 \texttt{FSWEIGHT} attribute is only enabled for ACCOUNT, GROUP, and QOS credentials.
6.3.3.5 Extended Fairshare Examples

Example 1: 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.

```plaintext
#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
```

6.3.4 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.

6.3.4.1 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:

<table>
<thead>
<tr>
<th>Shares</th>
<th>Format</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;COUNT&gt;[[@&lt;PARTITION&gt;],&lt;COUNT&gt;[[@&lt;PARTITION&gt;]]... where &lt;COUNT&gt; is a double and &lt;PARTITION&gt; is a specified partition name.</td>
<td>Specifies the node target usage or share.</td>
<td></td>
</tr>
</tbody>
</table>
### 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[Eng] SHARES=1500.5
FSTREE[Sales] SHARES=2800
MEMBERLIST=user:john,user:steve,user:bob
MEMBERLIST=user:kellyp,user:sam
MEMBERLIST=user:ux43,user:ux44,user:ux45
MEMBERLIST=user:robert,user:tjackson
```

Current tree configuration and monitored usage distribution is available using the `mdiat -f -v` commands.

### 6.3.4.2 Controlling Tree Evaluation

Moab provides multiple policies to customize how the share tree is evaluated.

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSTREEISPROPORTIONAL</td>
<td>Allows fs tree priority weighting to be proportional to usage discrepancies.</td>
</tr>
<tr>
<td>FSTREETIERMULTIPLIER</td>
<td>Decreases the value of sub-level usage discrepancies.</td>
</tr>
<tr>
<td>FSTREECAP</td>
<td>Caps lower level usage factors to prevent them from exceeding upper tier discrepancies.</td>
</tr>
</tbody>
</table>

### 6.3.4.2.1 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.

### 6.3.4.2.2 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 `SHARES` attribute of the FSTREE parameter and using the per-partition share specification.

In the following example, partition 1 is shared by the engineering and research departments, all organizations are allowed to use various portions of partition 2, and partition 3 is only accessible by research and sales.

```
FSTREE[root] SHARES=10000
MEMBERLIST=eng,research,sales
FSTREE[eng] SHARES=5000@par1,1000@par2
MEMBERLIST=user:johnt,user:stevek
FSTREE[research] SHARES=10000@par1,5000@par2,2000@par3
MEMBERLIST=user:barry,user:jsmith,user:bf4
FSTREE[sales] SHARES=5000@par2,10000@par3
MEMBERLIST=user:jen,user:lisa
```

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.

6.3.4.2.3 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 Manager Interface`. This interface can be used to load current information at startup and periodically synchronize this information with the master source.

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 (truncated) output:

```
FSTREE[tree]
 <fstree>
  <tnode partition="g02" name="root" type="acct" share="100">
   ...
  </tnode>
 </fstree>
```

6.3.4.2.4 Specifying Share Tree Based Limits

Limits can be specified on internal nodes of the share tree using standard credential limit semantics as shown in the following example:

```
FSTREE[sales] SHARES=400 MAXJOB=15 MAXPROC=200 MEMBERLIST=s1,s2,s3
FSTREE[s1] SHARES=150 MAXJOB=4 MAXPROC=40
MEMBERLIST=user:ben,user:jum3
FSTREE[s2] SHARES=50 MAXJOB=1 MAXPROC=50
MEMBERLIST=user:carol,user:johnson
FSTREE[s3] SHARES=200 MAXPS=4000 MAXPROC=150
MEMBERLIST=s3a,s3b,s3c
```

6.3.4.2.5 Other Uses of Share Trees

If a share tree is defined, it can be used for purposes beyond fairshare. These include 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.

6.3.5 Importing Fairshare Data

Moab can import fairshare data from external sources. Global fairshare data can be imported using the `Identity Manager` interface. To import global fairshare data, the total global fairshare usage must be imported on the "sched" object through the identity manager in addition to the global fairshare usage and target for particular credentials.

The following example shows a sample moab.cfg file that incorporates fairshare data from an external source and factors it into job priority:

```
IDCFG[gfs] SERVER="file:///HOME/tools/id.txt"
REFRESHPERIOD=minute
FSPOLICY DEDICATEDPS
```
In this example, Moab imports fairshare information from an external source and uses it to calculate a job's priority.

**See Also**

- *mdiaog -f* command
  - provides diagnosis and monitoring of the fairshare facility
- **FSENABLECAPPRIORITY** parameter
- **ENABLEFSPREEMPTION** parameter
- **ENABLESPPREEMPTION** parameter
- **FSTARGETISABSOLUTE** parameter
**Sample FairShare Data File**

FS.<EPOCHTIME>

<table>
<thead>
<tr>
<th># FS Data File (Duration: 43200 seconds) Starting: Sat Jul 8 06:00:20</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
</tr>
<tr>
<td>user</td>
</tr>
<tr>
<td>user</td>
</tr>
<tr>
<td>user</td>
</tr>
<tr>
<td>user</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>group</td>
</tr>
<tr>
<td>group</td>
</tr>
<tr>
<td>group</td>
</tr>
<tr>
<td>group</td>
</tr>
<tr>
<td>acct</td>
</tr>
<tr>
<td>acct</td>
</tr>
<tr>
<td>acct</td>
</tr>
<tr>
<td>acct</td>
</tr>
<tr>
<td>qos</td>
</tr>
<tr>
<td>qos</td>
</tr>
<tr>
<td>qos</td>
</tr>
<tr>
<td>class</td>
</tr>
<tr>
<td>sched</td>
</tr>
</tbody>
</table>

**Note:** 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 (i.e., USERA + USERB + USERC + USERD = GROUPA + GROUPB = ACCTA + ACCTB + ACCTC = QOS0 + QOS1 + QOS2 = SCHED).
6.4 Charging and Allocation Management

- 6.4.1 Charging and Allocation Management Overview
- 6.4.2 Using an External Allocation Manager
  - 6.4.2.1 Configuring the Allocation Manager Interface
  - 6.4.2.3 Allocation Management Policies
  - 6.4.2.3 Allocation Charge Rates
- 6.4.3 Allocation Manager Details
  - 6.4.3.1 Gold Allocation Manager
  - 6.4.3.2 Native Allocation Manager
  - 6.4.3.3 File Allocation Manager

6.4.1 Charging and Allocation Management Overview

Charging is the process of assigning a value to the use of resources and tracking this usage on a per consumer basis. Often, charging is accompanied by a corresponding assignment of resources (an allocation) to each consumer. Within Moab, charging can be quite flexible. Moab supports the following:

1. Assignment of fixed, expirable allocations to users, groups, and projects
2. Assignment of fixed, non-expirable allocations to users, groups, and projects
3. Assignment of dynamic allocations available within a sliding window
4. Specification of Quality of Service levels with distinct service targets and charging rates
5. Management over which consumers can request/access which Quality of Service levels
6. Ability to specify the metric of consumption for charging (i.e., CPU hours, dedicated node hours, PE's, etc.)
7. Ability to charge by requested QoS, delivered QoS, or other factors
8. Creation of complete persistent internal record of services delivered and resources allocated
9. Ability to call out to external auditing/accounting systems in real-time to authorize usage
10. Ability to call out to external auditing/accounting systems in real-time to register usage
11. Ability to adjust charge rates according to the configuration of resources allocated (i.e., processor speed, RAM installed, etc.)

6.4.2 Using an External Allocation Manager

An allocation manager (also known as an allocation bank or CPU bank) is a software system that manages resource allocations. A resource allocation grants a job a right to use a particular amount of resources. While full details of each allocation manager may be found within its respective documentation, the following brief review highlights a few of the values of using such a system.

An allocation manager functions much like a bank in that it provides a form of currency that allows jobs to run on an HPC system. The owners of the resource (cluster/supercomputer) determine how they want the system to be used (often via an allocations committee) over a particular time frame—often a month, quarter, or year. To enforce their decisions, they distribute allocations to various projects via accounts and assign each account an account manager. These allocations can be used for particular machines or globally. They can also have activation and expiration dates associated with them. All transaction information is typically stored in a database or directory server allowing extensive statistical and allocation tracking.

Each account manager determines how the allocations are made available to individual users within a project. Allocation managers such as Gold (from U.S. Dept of Energy) allow the account manager to dedicate portions of the overall allocation to individual users, specify some allocations as shared by all users, and hold some of the allocations in reserve for later use.

When using an allocations manager, each job must be associated with an account. To accomplish this with minimal user impact, the allocation manager could be set up to handle default accounts on a per user basis. However, as is often the case, some users may be active on more than one project and thus have access to more than one account. In these situations, a mechanism, such as a job command file keyword, should be provided to allow a user to specify which account should be associated with the job.

The amount of each job's allocation charge is directly associated with the amount of resources used (processors) by that job and the amount of time it was used. Optionally, the allocation manager can also be
configured to charge accounts varying amounts based on the QoS desired by the job, the type of compute
resources used, and the time when the resources were used (both in terms of time of day and day of week).

The allocation manager interface provides near real-time allocation management, giving a great deal of
flexibility and control over how available compute resources are used over the medium- and long-term, and
works hand-in-hand with other job management features such as Moab's usage limit policies and fairshare
mechanism.

The ENFORCEACCOUNTACCESS parameter controls whether the scheduler enforces account
constraints.

### 6.4.2.1 Configuring the Allocation Manager Interface

Moab's allocation manager interface(s) are defined using the AMCFG parameter. This parameter allows
specification of key aspects of the interface as shown in the following table:

<table>
<thead>
<tr>
<th>APPENDMACHINENAME</th>
<th>CHARGEPOLICY</th>
<th>FALLBACKACCOUNT</th>
<th>FALLBACKQOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLUSHERINTERVAL</td>
<td>FLAGS</td>
<td>NODECHARGEDEVICE</td>
<td>SERVER</td>
</tr>
<tr>
<td>SOCKETPROTOCOL</td>
<td>STRICTQUOTE</td>
<td>TIMEOUT</td>
<td>WIREPROTOCOL</td>
</tr>
<tr>
<td>JOBFailureACTION</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**APPENDMACHINENAME**

Format: **BOOLEAN**

Default: **FALSE**

Description: If specified, Moab appends the machine name to the consumer account to create a unique
account name per cluster.

Example: `AMCFG[tg13] APPENDMACHINENAME=TRUE`

Moab appends the machine name to each account before making a debit from the allocation manager.

**CHARGEPOLICY**

Format: one of **DEBITALLWC**, **DEBITALLCPU**, **DEBITALLPE**, **DEBITALLBLOCKED**, **DEBITSUCCESSFULWC**, **DEBITSUCCESSFULCPU**, **DEBITSUCCESSFULPE**, or **DEBITSUCCESSFULBLOCKED**

Default: **DEBITSUCCESSFULWC**

Description: Specifies how consumed resources should be charged against the consumer's credentials. See Charge Policy Overview for details.

Example: `AMCFG[bank] CHARGEPOLICY=DEBITALLCPU`

Allocation charges are based on actual CPU usage only, not dedicated CPU resources.

If the **LOCALCOST** flag (`AMCFG[] FLAGS=LOCALCOST`) is set, Moab uses the information
gathered with **CHARGEPOLICY** to calculate charges. If **LOCALCOST** is not set, Moab
sends this information to Gold to calculate charges.

**FALLBACKACCOUNT**

Format: **STRING**

Default: **---**

Description: If specified, Moab verifies adequate allocations for all new jobs. If adequate allocations are not
available in the job's primary account, Moab changes the job's credentials to use the fallback
account. If not specified, Moab places a hold on jobs that do not have adequate allocations in their primary account.

Example:

<table>
<thead>
<tr>
<th>AMCFG[bank] FALLBACKACCOUNT=freecycle</th>
</tr>
</thead>
</table>

Moab assigns the account `freecycle` to jobs that do not have adequate allocations in their primary account.

When both `FALLBACKACCOUNT` and `FALLBACKQOS` are specified, only `FALLBACKACCOUNT` takes effect.

### FALLBACKQOS

**Format:** STRING

**Default:** ---

**Description:** If specified, Moab verifies adequate allocations for all new jobs. If adequate allocations are not available in the job's primary QoS, Moab changes the job's credentials to use the fallback QoS. If not specified, Moab places a hold on jobs that do not have adequate allocations in their primary QoS.

Example:

<table>
<thead>
<tr>
<th>AMCFG[bank] FALLBACKQOS=freecycle</th>
</tr>
</thead>
</table>

Moab assigns the QoS `freecycle` to jobs that do not have adequate allocations in their primary QoS.

When both `FALLBACKACCOUNT` and `FALLBACKQOS` are specified, only `FALLBACKACCOUNT` takes effect.

### FLAGS

**Format:** <STRING>

**Default:** ---

**Description:** AMCFG flags are used to enable special services.

Example:

<table>
<thead>
<tr>
<th>AMCFG[xxxx] FLAGS=LOCALCOST</th>
</tr>
</thead>
</table>

Moab calculates the charge for the job locally and sends that as a charge to Gold, which then charges that amount for the job. This prevents Gold from having to calculate the charge for the job itself.

### FLUSHINTERVAL

**Format:** [[[DD:]HH:]MM:]SS

**Default:** 24:00:00

**Description:** Indicates the amount of time between allocation manager debits for long running reservation and job based charges.

Example:

<table>
<thead>
<tr>
<th>AMCFG[bank] FLUSHINTERVAL=12:00:00</th>
</tr>
</thead>
</table>

Moab updates its charges every twelve hours for long running jobs and reservations.

### JOBFAILUREACTION
<table>
<thead>
<tr>
<th><strong>SERVERFAILUREACTION</strong></th>
<th>&lt;SERVERFAILUREACTION&gt;[,&lt;FUNDSFAILUREACTION&gt;] where the action is one of CANCEL, HOLD, IGNORE, or RETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default</strong></td>
<td>IGNORE,HOLD</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The server failure action is taken if the allocation manager is down or otherwise unresponsive. The funds failure action is taken if the allocation manager reports that insufficient allocations are available to execute the job under the given user and account. If the action is set to CANCEL, Moab cancels the job; if set to HOLD, Moab defers the job; if set to IGNORE, Moab ignores the failure and continues to start the job; if set to RETRY, Moab does not start the job on this attempt but will attempt to start the job at the next opportunity.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>AMCFG[wg13] JOBFAILUREACTION=HOLD</td>
</tr>
</tbody>
</table>

Allocation management is strictly enforced, preventing jobs from starting if the allocation manager is unavailable.

<table>
<thead>
<tr>
<th><strong>NODECHARGEPOICY</strong></th>
<th>one of AVG, MAX, or MIN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default</strong></td>
<td>MIN</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>When charging for resource usage, the allocation manager will charge by node allocation according to the specified policy. For AVG, MAX, and MIN, the allocation manager will charge by the average, maximum, and minimum node charge rate of all allocated nodes. (Also see CHARGEPOLICY attribute.)</td>
</tr>
</tbody>
</table>
| **Example**         | NODECFG[node01] CHARGERATE=1.5  
|                     | NODECFG[node02] CHARGERATE=1.75  
|                     | AMCFG[wg13] NODECHARGEPOLICY=MAX  |

Allocation management charges jobs by the maximum allocated node's charge rate.

<table>
<thead>
<tr>
<th><strong>SERVER</strong></th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Specifies the type and location of the allocation manager service. If the keyword ANY is specified instead of a URL, Moab will use the local service directory to locate the allocation manager.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>AMCFG[.bio-sys] SERVER=gold://tiny.supercluster.org:4368</td>
</tr>
</tbody>
</table>

The URL protocol must be one of file or gold.

<table>
<thead>
<tr>
<th><strong>SOCKETPROTOCOL</strong></th>
<th>one of SUTCP, SSS-HALF, HTTP, or SSS-CHALLENGE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default</strong></td>
<td>SSS-HALF</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Specifies the socket protocol to be used for scheduler-allocation manager communication.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>AMCFG[bank] SOCKETPROTOCOL=SSS-CHALLENGE</td>
</tr>
</tbody>
</table>
The first step to configure the allocation manager involves specifying where the allocation service can be found. This is accomplished by setting the AMCFG parameter’s SERVER attribute to the appropriate URL.

After the interface URL is specified, secure communications between scheduler and allocation manager must be enabled. As with other interfaces, this is configured using the CLIENTCFG parameter within the moab-private.cfg file as described in the Security Appendix. In the case of an allocation manager, the KEY and AUTHTYPE attributes should be set to values defined during initial allocation manager build and configuration as in the following example:

```
CLIENTCFG[AM:bank] KEY=mysecr3t AUTHTYPE=HMAC64
```

### 6.4.2.2 AMCFG Flags

AMCFG flags can be used to enable special services and to disable default services. These services are enabled/disabled by setting the AMCFG FLAGS attribute.

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNTFAILASFUNDs</td>
<td>When this flag is set, logic failures within the Allocation Manager are treated as fund failures and are canceled. When ACCOUNTFAILASFUNDs is not set, Allocation Manager failures are treated as a server failure and the result is a job which requests an account to which the user does not have access.</td>
</tr>
<tr>
<td>LOCALCOST</td>
<td>Moab calculates the charge for the job locally and sends that as a charge to Gold, which then charges the amount for the job, instead of calculating the charge in Gold. This flag has only been tested for the Gold allocation manager.</td>
</tr>
<tr>
<td>STRICTQUOTE</td>
<td>Sends an estimated process count from Moab to Gold when an initial quote is requested for a newly-submitted job.</td>
</tr>
</tbody>
</table>

### 6.4.2.3 Allocation Management Policies

In most cases, the scheduler interfaces with a peer service. (If the protocol FILE is specified, the allocation manager transactions are written to the specified flat file.) With all peer services based allocation managers, the scheduler checks with the allocation manager before starting any job. For allocation tracking to work, however, each job must specify an account to charge or the allocation manager must be set up to handle default accounts on a per user basis.
Under this configuration, when Moab starts a job, it contacts the allocation manager and requests an allocation reservation (or lien) be placed on the associated account. This allocation reservation is equivalent to the total amount of allocation that could be consumed by the job (based on the job's wallclock limit) and is used to prevent the possibility of allocation over subscription. Moab then starts the job. When the job completes, Moab debits the amount of allocation actually consumed by the job from the job's account and then releases the allocation reservation, or lien.

These steps should be transparent to users. Only when an account has insufficient allocations to run a requested job will the presence of the allocation manager be noticed. If preferred, an account may be specified for use when a job's primary account is out of allocations. This account, specified using the AMCFG parameter's FALLBACKACCOUNT attribute, is often associated with a low QoS privilege and priority, and is often configured to run only when no other jobs are present.

The scheduler can also be configured to charge for reservations. One of the big hesitations with dedicating resources to a particular group is that if the resources are not used by that group, they go idle and are wasted. By configuring a reservation to be chargeable, sites can charge every idle cycle of the reservation to a particular project. When the reservation is in use, the consumed resources will be associated with the account of the job using the resources. When the resources are idle, the resources will be charged to the reservation's charge account. In the case of standing reservations, this account is specified using the parameter SRCFG attribute CHARGEACCOUNT. In the case of administrative reservations, this account is specified via a command line flag to the setres command.

Moab only interfaces to the allocation manager when running in NORMAL mode. However, this behavior can be overridden by setting the environment variable MOABAMTEST to any value. With this variable set, Moab attempts to interface to the allocation manager regardless of the scheduler's mode of operation.

Charge Metrics

The allocation manager interface allows a site to charge accounts in a number of different ways. Some sites may wish to charge for all jobs regardless of whether the job completed successfully. Sites may also want to charge based on differing usage metrics, such as dedicated wallclock time or processors actually used. Moab supports the following charge policies specified via the CHARGEPOLICY attribute:

- **DEBITALLWC** - Charges all jobs regardless of job completion state using processor weighted wallclock time dedicated as the usage metric.
- **DEBITALLCPU** - Charges all jobs based on processors used by job.
- **DEBITALLPE** - Charges all jobs based on processor-equivalents dedicated to job.
- **DEBITALLBLOCKED** - Charges all jobs based on processors dedicated and blocked according to node access policy or QoS node exclusivity.
- **DEBITREQUESTEDWC** - Charges for reservations based on requested wallclock time. Only applicable when using virtual private clusters.
- **DEBITSUCCESSFULWC** - Charges only jobs that successfully complete using processor weighted wallclock time dedicated as the usage metric. This is the default metric.
- **DEBITSUCCESSFULCPU** - Charges only jobs that successfully complete using CPU time as the usage metric.
- **DEBITSUCCESSFULPE** - Charges only jobs that successfully complete based on processors dedicated and blocked according to node access policy or QoS node exclusivity.

On systems where job wallclock limits are specified, jobs that exceed their wallclock limits and are subsequently canceled by the scheduler or resource manager are considered to have successfully completed as far as charging is concerned, even though the resource manager may report these jobs as having been removed or canceled.

If machine-specific allocations are created within the allocation manager, the allocation manager machine name should be synchronized with the Moab resource manager name as specified with the RMCFG parameter, such as the name orion in RMCFG[orion] TYPE=PBS.

To control how jobs are charged when heterogeneous resources are allocated and per resource charges may vary within the job, use the NODECHARGEPOLICY attribute.

When calculating the cost of the job, Moab will use the most restrictive node access policy. See NODEACCESSPOLICY for more information.

Allocation Management Example
In the following example, Moab charges allocations according to blocked resources and records these charges in the specified file.

```plaintext
AMCFG[local] SERVER=file://opt/moab/chargelog.txt
CHARGEPOLICY=DEBITALLBLOCKED
NODEACCESSPOLICY SINGLEJOB
...
```

### 6.4.2.3 Allocation Charge Rates

By default, Moab refers the decision of how much to charge to the allocation manager itself. However, if using the FILE Allocation Manager, job and reservation charge rates can be specified on a per-QoS basis using the DEDRESCOST parameter. If using the Gold Allocation Manager, per-QoS charge rates can be configured in Gold as demonstrated in these examples.

### 6.4.3.1 Gold Allocation Manager

**Gold** is an accounting and allocation management system developed at PNNL under the DOE Scalable Systems Software (SSS) project. Gold supports a dynamic approach to allocation tracking and enforcement with reservations, quotations, and so forth. It offers more flexible controls for managing access to computational resources and exhibits a more powerful query interface. Gold supports hierarchical project nesting. Journaling allows preservation of all historical state information.

Gold is dynamically extensible. New object/record types and their fields can be dynamically created and manipulated through the regular query language turning this system into a generalized accounting and information service. This capability offers custom accounting, meta-scheduler resource-mapping, and an external persistence interface.

Gold supports strong authentication and encryption and role based access control. Gold features a powerful web-based GUI for easy remote access for users, managers and administrators. Gold supports interaction with peer accounting systems with a traceback feature enabling it to function in a meta-scheduling or grid environment.

To configure a Gold allocation manager interface, set the `SERVER` attribute to point to the Gold server host and port; example follows:

**moab.cfg:**

```plaintext
AMCFG[bank] SERVER=gold://master.ufl.edu JOBFAILUREACTION=IGNORE
TIMEOUT=15
...
```

**moab-private.cfg:**

```plaintext
CLIENTCFG[AM:bank] KEY=mysecr3t AUTHTYPE=HMAC64
...
```

Create the secret key by running `make auth_key` during configuration.

```plaintext
[root]# make auth_key
```

### Monitor Mode

Gold can be enabled in an effective monitor-only mode where resource consumption is tracked but under no cases are jobs blocked or delayed based on allocation status. In this mode, full Gold reporting and accounting information is available.

1. Create an account that is valid for all projects, users, and machines.
2. Create an allocation with massive funds and no time bounds (using the account number created by the previous command).

   > gdeposit -a 5 1000000000000
   Successfully deposited 1000000000000 credits into account 5

3. To prevent failures due to unknown users, users that don't belong to the specified projects, and so forth, edit goldd.conf to automatically create users, projects, and machines.

   user.autogen = true
   project.autogen = true
   machine.autogen = true

### 6.4.3.2 Native Allocation Manager

The native allocation manager interface model has not been tested with HPC workload; it has only been tested with VPC style clouds.

The native allocation manager permits Moab to interface with a separate allocation manager to perform allocation management functions such as charging, billing, charge queries, and so forth so long as the separate allocation manager uses a native Wiki interface.

The design for the native allocation manager interface (NAMI) is different from Gold. NAMI extracts logic from Moab and places it in the native software. Moab acts as the event engine for the native software. That is, Moab sends XML that defines an object to a variety of URLs that signify events. Moab currently supports the following URLs:

- Create
- Delete
- Quote
- Reserve
- Charge

A user runs the `mshow` command and Moab calls to NAMI to get the QUOTE for the requested resources. If the TID is committed, then Moab calls the CREATE and RESERVE URLs for each object (reservation or job). Depending on the flush interval Moab periodically calls out to the CHARGE URL. When the object has reached its end of life Moab calls out to CHARGE and finally DELETE. Moab keeps track of the last time the object was charged, but it does not re-create reservations when restarting nor for intermittent charging. If Moab is down during a flush interval, then Moab does not attempt to catch up; it simply charges double the next flush interval.

The following is sample XML for the life of a particular object from Quote to Delete:

<table>
<thead>
<tr>
<th>URL</th>
<th>XML</th>
</tr>
</thead>
</table>
| Quote | `<Reservation>
   <ObjectId>cost</ObjectId>
   <Processors>1</Processors>
   <WallDuration>12960000</WallDuration>
   <ChargeDuration>12960000</ChargeDuration>
</Reservation>` |
| Create | `<Reservation>
   <ObjectId>host.1</ObjectId>
   <User>test</User>` |
Note that only the Quote URL should return any information. It should return nothing more than the cost of the object—no words—just the cost.

The following is a representation of how you might set up the native allocation manager interface in the Moab configuration file (moab.cfg):

```
AMCFG[bank] TYPE=NATIVE
AMCFG[bank] ChargeURL=exec://$HOME/tools/bank.charge.pl
AMCFG[bank] DeleteURL=exec://$HOME/tools/bank.delete.pl
AMCFG[bank] CreateURL=exec://$HOME/tools/bank.create.pl
AMCFG[bank] ReserveURL=exec://$HOME/tools/bank.reserve.pl
AMCFG[bank] QuoteURL=exec://$HOME/tools/bank.quote.pl
AMCFG[bank] FLUSHINTERVAL=hour
```

To view URL output, run `mdiag -R -v`. The following shows sample output from running the `mdiag` command:

```
<Reservation>
  <ObjectID>host.1</ObjectID>
  <User>test</User>
  <Account>blue</Account>
  <Processors>1</Processors>
  <WallDuration>12959999</WallDuration>
  <ChargeDuration>12959999</ChargeDuration>
  <Var name="VPCHOSTLIST">n05</Var>
  <Var name="VPCID">vpc.1</Var>
</Reservation>

<Reservation>
  <ObjectID>host.2</ObjectID>
  <User>test</User>
  <Account>blue</Account>
  <Processors>1</Processors>
  <WallDuration>12959999</WallDuration>
  <ChargeDuration>12959999</ChargeDuration>
  <Var name="VPCHOSTLIST">n05,GLOBAL</Var>
  <Var name="VPCID">vpc.1</Var>
  <GRes name="storage">100</GRes>
</Reservation>

<Reservation>
  <ObjectID>host.1</ObjectID>
  <User>test</User>
  <Account>blue</Account>
  <Processors>1</Processors>
  <WallDuration>12959999</WallDuration>
  <ChargeDuration>12959999</ChargeDuration>
  <Var name="VPCHOSTLIST">n05</Var>
  <Var name="VPCID">vpc.1</Var>
  <GRes name="storage">100</GRes>
</Reservation>

<Reservation>
  <ObjectID>host.2</ObjectID>
  <User>test</User>
  <Account>blue</Account>
  <Processors>1</Processors>
  <WallDuration>12959999</WallDuration>
  <ChargeDuration>12959999</ChargeDuration>
  <Var name="VPCHOSTLIST">n05,GLOBAL</Var>
  <Var name="VPCID">vpc.1</Var>
  <GRes name="storage">100</GRes>
</Reservation>
```
6.4.3.3 File Allocation Manager

The file allocation manager protocol allows a site to append job allocation records directly to a local file for batch processing by local allocation management systems. These records are line delimited with whitespace delimited attributes. Specifically, the file job usage record uses the following format:

WITHDRAWAL TYPE=job MACHINE=<MACHINENAME> ACCOUNT=<PROJECTNAME> USER=<USERNAME> PROCS=<PROCCOUNT> PROCCRATE=<PROCRATE> RESOURCETYPE=<NODETYPE> DURATION=<WALLDURATION> REQUESTID=<JOBID>

For example, the following record might be created:

WITHDRAWAL TYPE=job MACHINE=ia; ACCOUNT=s USER=jb PROCS=64 PROCCRATE=0.93 RESOURCETYPE=ia64 DURATION=60 REQUESTID=1632

To configure a file allocation manager interface, set the SERVER attribute to point to the local file pathname as follows:

AMCFG[local] SERVER=file:///opt/data/alloc.txt

See Also

- Internal Charging
- Per Class DISABLEAM attribute
- Charging for Reservations
6.5 Internal Charging Facilities

6.5.1 Internal Charging Overview

Moab provides internal support for a number of allocation management tasks and policies. In terms of charging, these facilities include the ability to assign per resource, per user, and per QoS charging rates. For resource allocation, it supports the ability to allocate fixed per user, group, project, and QoS allocations as well as the ability to enable sliding window based resource allocations.

Per resource charging rates are specified using the CHARGERATE attribute of the NODECFG parameter. This attribute is supported for both DEFAULT and specific node configurations.

Per QoS charging is covered in detail in the QoS Charging section.

Sliding window based resource allocations are enabled by configuring the fairshare facility and enabling cap based targets as documented in the Fairshare Targets section.

Credits can be granted to accounts using the CREDITS attribute of the ACCOUNTCFG parameter (ACCOUNTCFG[<name>] CREDITS=<FLOAT>). When a job finishes, its cost (determined by the per QoS charging facility) is debited from the credits of the account under which it ran. This allows sites to allocate credits to certain accounts.

**Example**

The following configuration highlights some of these capabilities. The first two lines define the charging policy. In this case, jobs are charged based on wallclock time and requested quality of service. The next two lines indicate that jobs requesting the special services available within the premium QoS are charged at a rate 10x that of other jobs. The ACCOUNTCFG lines provide a number of consumable credits to the specified accounts. Resources used by jobs associated with these accounts are charged against these credits. Finally, for auditing purposes, the AMCFG file is added to cause Moab to report all charging based actions to an allocation manager events file.

```
SCHEDCFG[sched] CHARGERATEPOLICY=QOSREQ
SCHEDCFG[sched] CHARGEMETRICPOLICY=DEBITALLWC
QOSCFG[premium] DEDRESCOST=10.0
QOSCFG[DEFAULT] DEDRESCOST=1.0
ACCOUNTCFG[marketing] CREDITS=85000
ACCOUNTCFG[sales] CREDITS=13500
ACCOUNTCFG[development] CREDITS=50000
AMCFG[local] SERVER=FILE:///opt/moab/charging.log
```

This setup charges each job 10.0 credits for each second it runs inside the QoS premium. After each job runs, its cost is added to the total amount of credits used.

Unlike full allocation management systems, Moab’s internal charging facility filters jobs to verify adequate allocations exist before starting but to prevent oversubscription, does not create a true real-time allocation reservation at job start time.

6.5.2 Managing Internal Charging

Current internal charging credit status can be viewed using the Moab Cluster Manager graphical administrator tool or by issuing mdiag -a at a command line.

```
> mdiag -a
evaluating acct information
Name Priority Flags QDef QOSList
PartitionList Target Limits
```
Both base credit and used credit values can be dynamically adjusted using the `mschedctl -m` command as in the following example:

```
# set available credits to 15K for sales
> mschedctl -m --flags=pers "ACCOUNTCFG[sales] CREDITS=15000"

# give an additional 125K credits to marketing
> mschedctl -m --flags=pers "ACCOUNTCFG[marketing] CREDITS+=125000"

# give marketing a refund of 5K credits
> mschedctl -m --flags=pers "ACCOUNTCFG[marketing] USED_CREDITS-=5000"
```

**See Also**

- Allocation Management Overview
7.0 Controlling Resource Access - Reservations, Partitions, and QoS Facilities

- 7.1 Advance Reservations
- 7.2 Partitions
- 7.3 QoS Facilities
7.1 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, QOS support, and grid scheduling.

- 7.1.1 Reservation Overview
- 7.1.2 Administrative Reservations
- 7.1.3 Standing Reservations
- 7.1.4 Reservation Policies
- 7.1.5 Configuring and Managing Reservations
- 7.1.6 Enabling Reservations for End-users
7.1.1 Reservation Overview

- 7.1.1.1 Resources
- 7.1.1.2 TimeFrame
- 7.1.1.3 Access Control List
- 7.1.1.4 Job to Reservation Binding
- 7.1.1.5 Reservation Specification
- 7.1.1.6 Reservation Behavior
- 7.1.1.7 Reservation Group

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.

7.1.1.1 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. 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.

7.1.1.2 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.

7.1.1.3 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,
### 7.1.1.4 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.

![Diagram showing job allocation from reservations A and B](image)

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:

```bash
> 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.

```bash
> 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:

```bash
> msub -l nodes=1,walltime=1:00:00,flags=ADVRES testjob.cmd
```

To enable job to reservation mapping via QoS, the QoS flag **USERRESERVED** should be set in a similar manner.

Use the reservation **BYNAME** flag to require explicit binding for reservation access.

### 7.1.1.5 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.

### 7.1.1.6 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.

### 7.1.1.7 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** and **VPCs** 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**.

**See Also**

- Reservation Allocation Policies
- Reservation Re-Allocation Policies
7.1.2 Administrative Reservations

- 7.1.2.1 Annotating Administrative Reservations
- 7.1.2.2 Using Reservation Profiles
- 7.1.2.3 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.

7.1.2.1 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]'
```

7.1.2.2 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:

```
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 2: 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
```

7.1.2.3 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.

**See Also**

- Backfill
- Preemption
- mrsvctl command
7.1.3 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.

See Also

- SRCFG
- Moab Workload Manager for Grids
- mdiag -s (diagnose standing reservations)
7.1.4 Reservation Policies

- 7.1.4.1 Controlling Priority Reservation Creation
- 7.1.4.2 Managing Resource Failures
- 7.1.4.3 Resource Allocation Policy
- 7.1.4.4 Resource Re-Allocation Policy
- 7.1.4.5 Charging for Reserved Resources

7.1.4.1 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. Moab’s dynamic job prioritization allows sites to prioritize jobs so that their priority order can change over time. 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGHEST</td>
<td>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 and 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.</td>
</tr>
<tr>
<td>CURRENTHIGHEST</td>
<td>Only the current top &lt;RESERVATIONDEPTH&gt; priority jobs receive reservations. If a job had a reservation but has been bypassed in priority by another job so that it no longer qualifies as being among the top &lt;RESERVATIONDEPTH&gt; jobs, it loses its reservation.</td>
</tr>
<tr>
<td>NEVER</td>
<td>No priority reservations are made.</td>
</tr>
</tbody>
</table>

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 QoS's. For example, the following configuration creates two reservation depth groupings:

<table>
<thead>
<tr>
<th><strong>RESERVATIONDEPTH[0]</strong></th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESERVATIONQOSLIST[0]</strong></td>
<td>highprio, interactive, debug</td>
</tr>
<tr>
<td><strong>RESERVATIONDEPTH[1]</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>RESERVATIONQOSLIST[1]</strong></td>
<td>batch</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th><strong>RESERVATIONDEPTH[DEFAULT]</strong></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESERVATIONQOSLIST[DEFAULT]</strong></td>
<td>ALL</td>
</tr>
</tbody>
</table>

This allows one job with the highest priority to get a reservation. These values can be overwritten by modifying the **DEFAULT** policy.

**7.1.4.2 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.

7.1.4.3 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.

7.1.4.4 Resource Re-Allocation Policy

Over time, Moab maintains the reservation on the initially allocated resources. However, in some cases, it is best to allow Moab to be more flexible in the management of these resources. In these cases, the RSVREALLOCPOLICY parameter can be used to specify the best behavior. This parameter supports the following policies:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAILURE</td>
<td>Only replace allocated resources that have failed (marked down).</td>
</tr>
<tr>
<td>NEVER</td>
<td>Do not dynamically reallocate resources to a reservation maintaining the collection of resources allocated at reservation creation time.</td>
</tr>
<tr>
<td>OPTIMAL</td>
<td>Dynamically reallocate reservation resources to minimize the reservation cost and maximize reservation preferences (for Moab 5.0 and later).</td>
</tr>
<tr>
<td>REMAP</td>
<td>Dynamically reallocate reservation resources to minimize the reservation footprint on idle nodes by allocating nodes that are in use by consumers that match the reservation’s ACL.</td>
</tr>
</tbody>
</table>

7.1.4.5 Charging for Reserved Resources

By default, resources consumed by jobs are tracked and charged to an allocation manager. However, resources dedicated to a reservation are not charged although they are recorded within the reservation event record. In particular, total processor-seconds reserved by the reservation are recorded as are total unused processor-seconds reserved (processor-seconds not consumed by an active job). While this information is available in real-time using the mdiag -r command (see the Active PH field), it is not written to the event log until reservation completion.

To enable direct charging, accountable credentials should be associated with the reservation. If using mrsvctl, the attributes aaccount, auser, aqos, and agroup can be set using the -S flag. If specified, these credentials are charged for all unused cycles reserved by the reservation.

Example: Assigning Accountable Credentials to a Reservation

```
> mrsvctl -c -h node003 -a user=john,user=steve -S aaccount=jupiter
```

Moab allocation management interface allows charging for reserved idle resources to be exported in real-time to peer services or to a file. To export this charge information to a file, use the file server type as in the following example configuration:

Example: Setting up a File Based Allocation Management Interface

```
AMCFG [local] server=file://$HOME/charge.dat
```

As mentioned, by default, Moab only writes out charge information upon completion of the reservation. If more timely information is needed, the FLUSHINTERVAL attribute can be specified.
See Also

- Reservation Overview
- Backfill
7.1.5 Configuring and Managing Reservations

- 7.1.5.1 Reservation Attributes
  - 7.1.5.1.1 Start/End Time
  - 7.1.5.1.2 Access Control List (ACL)
  - 7.1.5.1.3 Selecting Resources
  - 7.1.5.1.4 Flags
- 7.1.5.2 Configuring and Managing Standing Reservations
  - 7.1.5.2.1 Standing Reservation Overview
  - 7.1.5.2.2 Specifying Reservation Resources
  - 7.1.5.2.3 Enforcing Policies Via Multiple Reservations
  - 7.1.5.2.4 Affinity
  - 7.1.5.2.5 ACL Modifiers
  - 7.1.5.2.6 Reservation Ownership
  - 7.1.5.2.7 Partitions
  - 7.1.5.2.8 Resource Allocation Behavior
  - 7.1.5.2.9 Rolling Reservations
  - 7.1.5.2.10 Modifying Resources with Standing Reservations
- 7.1.5.3 Managing Administrative Reservations

7.1.5.1 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.

7.1.5.1.1 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.

7.1.5.1.2 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. 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"

7.1.5.1.3 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**

```
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**

```
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**

```
SRCFG[test] HOSTLIST=node01,node1[3-5]
```

**Node Features**

Node features can be specified to constrain which resources are considered.

**Example**

```
SRCFG[test] NODEFEATURES=fastos
```

**Partition**

A partition may be specified to constrain which resources are considered.

**Example**

```
SRCFG[test] PARTITION=core3
```

**7.1.5.1.4 Flags**

Reservation flags allow specification of special reservation attributes or behaviors. Supported flags are listed in the following table:

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACLOVERLAP</td>
<td>When reservations are first created, they will by default only allocate free or idle nodes. If the ACLOVERLAP flag is set, a reservation may also</td>
</tr>
<tr>
<td><strong>ADVRESJOBDESTROY</strong></td>
<td>All jobs that have an ADVRES matching this reservation are canceled when the reservation is destroyed.</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ALLOWPRSV</strong></td>
<td>Personal reservations can be created within the space of this standing reservation (and <strong>ONLY</strong> this standing reservation); by default, when a standing reservation is given the flag ALLOWPRSV it is given the ACL USER==ALL+ allowing all jobs and all users access.</td>
</tr>
<tr>
<td><strong>BYNAME</strong></td>
<td>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.)</td>
</tr>
<tr>
<td><strong>DEDICATEDRESOURCE (aka EXCLUSIVE)</strong></td>
<td>Reservation only placed on resources that are not reserved by any other reservation including jobs and other reservations.</td>
</tr>
<tr>
<td><strong>IGNRSV</strong></td>
<td>Request ignores existing resource reservations allowing the reservation to be forced onto available resources even if this conflicts with other reservations.</td>
</tr>
<tr>
<td><strong>IGNJOBSRV</strong></td>
<td>Functions like IGNRSV but only ignores job reservations. User and system reservation conflicts are still valid.</td>
</tr>
<tr>
<td><strong>IGNSTATE</strong></td>
<td>Reservation ignores node state when assigning nodes.</td>
</tr>
<tr>
<td><strong>NOCHARGE</strong></td>
<td>By default, Moab charges Gold for unused cycles in a standing reservation. Setting the NOCHARGE flag prevents Moab from charging Gold for standing reservations.</td>
</tr>
<tr>
<td><strong>NOVMMIGRATION</strong></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 novmmigrations flag, then VMs will be migrated.</td>
</tr>
<tr>
<td><strong>OWNERPREEMPT</strong></td>
<td>Jobs by the reservation owner are allowed to preempt non-owner jobs using reservation resources.</td>
</tr>
<tr>
<td><strong>OWNERPREEMPTIGNOREMINTIME</strong></td>
<td>Allows the OWNERPREEMPT flag to &quot;trump&quot; 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.</td>
</tr>
<tr>
<td></td>
<td>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><strong>REQFULL</strong></td>
<td>Reservation is only created when all resources can be allocated.</td>
</tr>
<tr>
<td><strong>SINGLEUSE</strong></td>
<td>Reservation is automatically removed after completion of the first job to use the reserved resources.</td>
</tr>
<tr>
<td>SPACEFLEX</td>
<td>Reservation is allowed to adjust resources allocated over time in an attempt to optimize resource utilization*.</td>
</tr>
<tr>
<td>TIMEFLEX</td>
<td>Reservation is allowed to adjust the reserved time frame in an attempt to optimize resource utilization*.</td>
</tr>
<tr>
<td>VPC</td>
<td>Reservation defines a virtual private cluster.</td>
</tr>
</tbody>
</table>

Reservations must explicitly request the ability to float for optimization purposes by using the **SPACEFLEX** flag.

Except for SPACEFLEX, most flags can be associated with a reservation via the `mrvctl -c -F` command or the `SRCFG` parameter.

### 7.1.5.2 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. All standing reservation attributes are specified via the `SRCFG` parameter using the attributes listed in the table below.

#### Standing Reservation Attributes

<table>
<thead>
<tr>
<th><strong>ACCESS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> DEDICATED or SHARED</td>
</tr>
<tr>
<td><strong>Default:</strong> ---</td>
</tr>
<tr>
<td><strong>Description:</strong> If set to <strong>SHARED</strong>, allows a standing reservation to use resources already allocated to other non-job reservations. Otherwise, these other reservations block resource access.</td>
</tr>
</tbody>
</table>

**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.**

<table>
<thead>
<tr>
<th><strong>ACCOUNTLIST</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> List of valid, comma delimited account names (see ACL Modifiers).</td>
</tr>
<tr>
<td><strong>Default:</strong> ---</td>
</tr>
<tr>
<td><strong>Description:</strong> Specifies that jobs with the associated accounts may use the resources contained within this reservation.</td>
</tr>
</tbody>
</table>

**Example:**

```
SRCFG[test] ACCOUNTLIST=ops,staff
```

Jobs using the account *ops* or *staff* are granted access to the resources in standing reservation *test*.

<table>
<thead>
<tr>
<th><strong>CHARGEACCOUNT</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> Any valid accountname.</td>
</tr>
</tbody>
</table>
### Description
Specifies the account to which Moab will charge all idle cycles within the reservation (via the allocation manager).

**CHARGEACCOUNT** must be used in conjunction with **CHARGEUSER**.

### Example
```
SRCFG[sr_gold1] HOSTLIST=kula
SRCFG[sr_gold1] PERIOD=INFINITY
SRCFG[sr_gold1] OWNER=USER:admin
SRCFG[sr_gold1] CHARGEACCOUNT=math
SRCFG[sr_gold1] CHARGEUSER=john
```

Moab charges all idle cycles within reservations supporting standing reservation *sr_gold1* to account *jupiter*.

---

### CHARGEUSER

**Format:** Any valid username.

**Default:** ---

**Description:** Specifies the user to which Moab will charge all idle cycles within the reservation (via the allocation manager).

**CHARGEUSER** must be used in conjunction with **CHARGEACCOUNT**.

### Example
```
SRCFG[sr_gold1] HOSTLIST=kula
SRCFG[sr_gold1] PERIOD=INFINITY
SRCFG[sr_gold1] OWNER=USER:admin
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:** ---
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Specifies that jobs originating within the listed clusters may use the resources contained within this reservation.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example:</strong></td>
<td><code>SRCFG[test] CLUSTERLIST=orion2,orion7</code> Moab grants jobs from the listed peer clusters access to the reserved resources.</td>
</tr>
</tbody>
</table>

**COMMENT**

<table>
<thead>
<tr>
<th><strong>Format:</strong></th>
<th><code>&lt;STRING&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default:</strong></td>
<td>---</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Specifies a descriptive message associated with the standing reservation and all child reservations.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td><code>SRCFG[test] COMMENT='rsv for network testing'</code> 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.</td>
</tr>
</tbody>
</table>

**DAYS**

<table>
<thead>
<tr>
<th><strong>Format:</strong></th>
<th>One or more of the following (comma delimited): Mon, Tue, Wed, Thu, Fri, Sat, Sun or [ALL].</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default:</strong></td>
<td>[ALL]</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Specifies which days of the week the standing reservation is active.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td><code>SRCFG[test] DAYS=Mon,Tue,Wed,Thu,Fri</code> Standing reservation test is active Monday through Friday.</td>
</tr>
</tbody>
</table>

**DEPTH**

<table>
<thead>
<tr>
<th><strong>Format:</strong></th>
<th><code>&lt;INTEGER&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default:</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Specifies the depth of standing reservations to be created, starting at depth 0 (one per period).</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td><code>SRCFG[test] PERIOD=DAY DEPTH=6</code> Specifies that seven (0 indexed) reservations will be created for standing reservation test.</td>
</tr>
</tbody>
</table>

**DISABLE**

<table>
<thead>
<tr>
<th><strong>Format:</strong></th>
<th><code>&lt;BOOLEAN&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default:</strong></td>
<td>FALSE</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Specifies that the standing reservation should no longer spawn child reservations.</td>
</tr>
</tbody>
</table>
### 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>` (see [note](#) that follows).

**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

---

**Example:**

```
SRCFG[test] PERIOD=DAY DEPTH=7 DISABLE=TRUE
SRCFG[test] STARTTIME=8:00:00
SRCFG[test] ENDTIME=17:00:00
SRCFG[test] PERIOD=DAY
SRCFG[test] FLAGS=BYNAME,DEDICATEDRESOURCE
SRCFG[test] GROUPLIST=staff,ops,special
SRCFG[test] CLASSLIST=interactive
```
hosts are reserved.

**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.

**JOBATTRLIST**

**Format:** Comma delimited list of one or more of the following job attributes: **PREEMPTEE, INTERACTIVE**, or any generic attribute configured through **NODECFG**.

**Default:** ---

**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.

### OS

**Format:** Operating system.

**Default:** ---

**Description:** 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.

**Example:**
```
SRCFG[test] OS=SUSE91
```

The resources allocated to the reservation are provisioned to SuSE 9.1 during the life of the reservation and restored to their original OS at the end of the reservation.

### 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.

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]

**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.
<table>
<thead>
<tr>
<th>Default:</th>
<th>DAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td><strong>Specifies the period of the standing reservation.</strong></td>
</tr>
</tbody>
</table>
| Example: | SRCFG[test] PERIOD=WEEK  
Each standing reservation covers a one week period. |

### PROCLIMIT

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;QUALIFIER&gt;</code>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;QUALIFIER&gt;</code></td>
<td>may be one of the following &lt;, &lt;=, ==, &gt;=, &gt;</td>
</tr>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td><strong>Specifies the processor limit for jobs requesting access to this standing reservation.</strong></td>
</tr>
</tbody>
</table>
| Example: | SRCFG[test] PROCLIMIT<=4  
Jobs requesting 4 or fewer processors are allowed to run. |

### PSLIMIT

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;QUALIFIER&gt;</code>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;QUALIFIER&gt;</code></td>
<td>may be one of the following &lt;, &lt;=, ==, &gt;=, &gt;</td>
</tr>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td><strong>Specifies the processor-second limit for jobs requesting access to this standing reservation.</strong></td>
</tr>
</tbody>
</table>
| 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. |

### 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 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 for more info about quality of service and service level agreements; also see Rollback Reservation Overview.

**Example:**

```
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
```

The standing reservation guarantees access to up to 32 processors within 24 hours to jobs from the ajax account.

### 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 7.1.1.7 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:**

```bash
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:**

```bash
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:**

```bash
SRCFG[test] TIMELIMIT=1:00:00
```

Moab allows jobs to access up to one hour of resources in the standing reservation.

---

**TIMELOGIC**

**Format:** **AND** or **OR**

**Default:** **OR**

**Description**

TIMELOGIC is not supported in Moab. Under Maui, this attribute specifies how TIMELIMIT access status will be combined with other standing reservation access methods to determine job access. If TIMELOGIC is set to OR, a job is granted access to the reserved resources if it meets the TIMELIMIT criteria or any other access criteria (such as USERLIST). If TIMELOGIC is set to AND, a job is granted access to the reserved resources only if it meets the TIMELIMIT criteria and at least one other access criteria.

As TIMELOGIC is not supported in Moab, use the required ACL marker, * instead. Equivalent functionality can be enabled by setting something like the following:

```bash
SRCFG[special] TIMELIMIT=1:00:00*
```
### TPN (Tasks Per Node)

| Format:    | <INTEGER> |
| Default:   | 0 (no TPN constraint) |
| Description | Specifies the minimum 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 at least four tasks on each node that is to be part of the reservation. That is, each node included in standing reservation 2 must have at least 8 processors and 1 GB of memory available.

### TRIGGER

| Format:              | See 19.1 Triggers 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 19.1 Triggers 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.

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$`.

### 7.1.5.2.1 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 \texttt{DEPTH} attribute of the \texttt{SRCFG} parameter.

The following examples demonstrate possible configurations specified with the \texttt{SRCFG} parameter.

**Example 1: Basic Business Hour Standing Reservation**

- \texttt{SRCFG[interactive] TASKCOUNT=6 RESOURCES=PROCS:1,MEM:512}
- \texttt{SRCFG[interactive] PERIOD=DAY DAYS=MON,TUE,THU,FRI}
- \texttt{SRCFG[interactive] STARTTIME=9:00:00 ENDTIME=17:00:00}
- \texttt{SRCFG[interactive] CLASSLIST=interactive}

When using the \texttt{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 \texttt{SRCFG[test] CLASSLIST=classA,classB}.

Only one \texttt{STARTTIME} and one \texttt{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 preceding example fully specifies a reservation including the quantity of resources requested using the \texttt{TASKCOUNT} and \texttt{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 \texttt{TASKCOUNT} attribute specifies the number of these tasks that should be reserved by the reservation. In conjunction with this attribute, the \texttt{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 \texttt{PERIOD} attribute may be set to a value of \texttt{DAY}, \texttt{WEEK}, or \texttt{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 \texttt{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: \texttt{MON} or \texttt{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 \texttt{STARTTIME} and \texttt{ENDTIME} attributes. These attributes are specified in standard military time HH:MM:SS format and both \texttt{STARTTIME} and \texttt{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 \texttt{CLASSLIST} attribute is used to indicate that jobs requesting the class \texttt{interactive} should be allowed to use this reservation.

**Example 2: Host Constrained Standing Reservation**

- \texttt{SRCFG[interactive] TASKCOUNT=6 RESOURCES=PROCS:1,MEM:512}
- \texttt{SRCFG[interactive] PERIOD=DAY DAYS=MON,TUE,THU,FRI}
- \texttt{SRCFG[interactive] STARTTIME=9:00:00 ENDTIME=17:00:00}
- \texttt{SRCFG[interactive] CLASSLIST=interactive}

An examination of the parameters documentation show that the default value of \texttt{PERIOD} is \texttt{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.

**7.1.5.2.2 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, \texttt{RESOURCES} is set to \texttt{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 \texttt{RESOURCES} attribute provides a powerful means of specifying an exact, multi-dimensional resource set.

An examination of the parameters documentation show that the default value of \texttt{PERIOD} is \texttt{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.
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.

### 7.1.5.2.3 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: 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
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 rolling, 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 4: 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:
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 satisfy 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.

### 7.1.5.2.4 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-`.

### 7.1.5.2.5 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 table:

<table>
<thead>
<tr>
<th>Not</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symbol:</strong> ! (exclamation point)</td>
<td>If attribute is met, the requestor is denied access regardless of any other satisfied ACLs.</td>
</tr>
<tr>
<td><strong>Example:</strong> SRCFG[test] GROUPLIST=staff USERLIST=!steve</td>
<td>Allow access to all staff members other than Steve.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Required</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symbol:</strong> * (asterisk)</td>
<td></td>
</tr>
</tbody>
</table>

**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.

**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.

**Example:**

```
SRCFG[test] USERLIST=&john
```

All of 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 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. 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 4 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 (l!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 5: 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
```

---

**7.1.5.2.6 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 6: Limited Shared Access**

```plaintext
ACCOUNTCFG[jupiter] PRIORITY=10000
SRCFG[jupiter] HOSTLIST=node0[1-9]
SRCFG[jupiter] PERIOD=INFINITY
SRCFG[jupiter] ACCOUNTLIST=jupiter,saturn-
SRCFG[jupiter] OWNER=ACCOUNT:jupiter
SRCFG[jupiter] FLAGS=OWNERPREEMPT
```

### 7.1.5.2.7 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 7: 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*.

```plaintext
SRCFG[smallrsv] PARTITION=small MAXTIME=3:00:00:00 PSLIMIT<=230400
HOSTLIST=ALL
SRCFG[bigrsv] PARTITION=big MAXTIME=3:00:00:00 HOSTLIST=ALL
```

### 7.1.5.2.8 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:**

```plaintext
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 8: 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.

```bash
SRCFG[shortpool] OWNER=ACCOUNT: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—float—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.

7.1.5.2.9 Rolling Reservations

Rolling 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.
Example 9: Rollback Reservations

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
```

7.1.5.2.10 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[ajax] PERIOD=INFINITY ACCOUNTLIST=ajax
SRCFG[ajax] ROLLBACKOFFSET=24:00:00 TASKCOUNT=32
SRCFG[ajax] PERIOD=INFINITY ACCOUNTLIST=ajax

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
```
7.1.5.3 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 `mdia` 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.

<table>
<thead>
<tr>
<th>Action</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>create reservation</td>
<td><code>mrsvctl -c &lt;RSV_DESCRIPTION&gt;</code></td>
</tr>
<tr>
<td>list reservations</td>
<td><code>mrsvctl -l</code></td>
</tr>
<tr>
<td>release reservation</td>
<td><code>mrsvctl -r &lt;RSVID&gt;</code></td>
</tr>
<tr>
<td>modify reservation</td>
<td><code>mrsvctl -m &lt;ATTR&gt;=&lt;VAL&gt; &lt;RSVID&gt;</code></td>
</tr>
<tr>
<td>query reservation configuration</td>
<td><code>mdia -r &lt;RSVID&gt;</code></td>
</tr>
<tr>
<td>display reservation hostlist</td>
<td><code>mrsvctl -q resources &lt;RSVID&gt;</code></td>
</tr>
</tbody>
</table>

See Also

- **SRCFG** (configure standing reservations)
- **RSVPROFILE** (create reservation profiles)
7.1.6 Personal Reservations (a.k.a User Reservations) - Enabling Reservations for End Users

- 7.1.6.1 Enabling Personal Reservation Management
- 7.1.6.2 Reservation Accountability and Defaults
  - 7.1.6.2.1 Reservation Allocation and Charging
  - 7.1.6.2.2 Setting Reservation Default Attributes
- 7.1.6.3 Reservation Limits
  - 7.1.6.3.1 Reservation Usage Limits
  - 7.1.6.3.2 Reservation Masks (a.k.a. Personal Reservation Sandbox)
- 7.1.6.4 Reservation and Job Binding
  - 7.1.6.4.1 Constraining a job to only run in a particular reservation
  - 7.1.6.4.2 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.

### 7.1.6.1 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:

```plaintext
QOS_CFG[default] QFLAGS=ENABLEUSERRSV # allow 'default' QOS jobs to create user reservations
USER_CFG[USER] QDEF=default            # allow all users to access 'default' 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:

```plaintext
> 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: Allow All Users in Engineering Group to Create Personal Reservations**

```plaintext
QOS_CFG[rsv] QFLAGS=ENABLEUSERRSV # allow 'rsv' QOS jobs to create user reservations
GROUP_CFG[engineering] QDEF=rsv      # allow all users in group engineering to access 'rsv' QOS
...
```

**Example: Allow Specific Users to Create Personal Reservations**

```plaintext
# special qos has higher job priority and ability to create user reservations
```
7.1.6.2 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 mrsvct1 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.

7.1.6.2.1 Reservation Allocation and Charging

If an allocation management system is used, Moab can be configured to charge users for both the creation of the reservation and the reservation itself. Using the QoS RSVCREATIONCOST and DEDRESCOST attributes, a site may specify a per-QoS cost for each reservation created. The RSVCREATIONCOST cost is a flat charge in allocation units applied regardless of whether the reservation is used. The DEDRESCOST tracks reserved (or dedicated) resources allowing sites to charge by the quantity of resources (in processor-seconds) that are actually reserved. As with jobs, these charges are applied against the personal reservation account.

7.1.6.2.2 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.

7.1.6.3 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.

While time spanning throttling policies are a significant step in the direction of end-user advance reservation management, it is important to track actual site usage of the advance reservation facility. It is still likely that further usage policies are required at each site to prevent reservation misuse and provide an optimally useful system.

7.1.6.3.1 Reservation Usage Limits

Personal reservation usage limits may be enforced using the attributes specified in the following table. These limits can be used to constrain the resources reserved in any single reservation as well as the total quantity of resources that may be reserved cumulatively across all 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 usage limits are independent of job limits imposed on users, groups, and other credentials.
<table>
<thead>
<tr>
<th>Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMAXDURATION</td>
<td>Limits the duration (in seconds) of any single personal reservation.</td>
</tr>
<tr>
<td>RMAXPROC</td>
<td>Limits the size (in processors) of any single personal reservation.</td>
</tr>
<tr>
<td>RMAXPS</td>
<td>Limits the size (in processor-seconds) of any single personal reservation.</td>
</tr>
<tr>
<td>RMAXCOUNT</td>
<td>Limits the total number of personal reservations a credential may have active at any given moment.</td>
</tr>
<tr>
<td>RMAXTOTALDURATION</td>
<td>Limits the total duration of personal reservations a credential may have active at any given moment.</td>
</tr>
<tr>
<td>RMAXTOTALPROC</td>
<td>Limits the total number of processors a credential may reserve active at any given moment.</td>
</tr>
<tr>
<td>RMAXTOTALPS</td>
<td>Limits the total number of processor-seconds a credential may reserve active at any given moment.</td>
</tr>
</tbody>
</table>

Example: Constrain Size/Duration of Personal Reservations

```plaintext
# special qos can only make reservations up to 5 procs and 2 hours
QOSCFG[special] QFLAGS=ENABLEUSERRSV RMAXPROC=5 RMAXDURATION=2:00:00
```

7.1.6.3.2 Reservation Masks (a.k.a. Personal Reservation Sandbox)

Additional control over when and where personal reservations can be created can be enabled through the use of a mask reservation. To create a mask reservation, a standard standing reservation can be created with the flag `ALLOWPRSV` set. If such a reservation exists, the scheduler constrains all personal reservation requests to only use resources within this reservation. This effectively allows a site to constrain the times of the day and days of the week during which a personal reservation can be created and limits the requests to a specific subset of resources. Note that reservation masks can be used in conjunction with reservation usage limits to provide a finer level of control. Also note that reservation masks still support the concept of the reservation ACL, which can constrain exactly who or what may request the resources within the reservation mask. To enable global access to a standing reservation with the flag `ALLOWPRSV`, the ACL `USER==ALL+` is assigned to the reservation by default.

Personal reservation masks/sandboxes only impact users who are not Moab administrators. Moab administrators are allowed to create personal reservations on any resource they can normally access.

Example: Create Personal Reservation Sandbox

```plaintext
# allow personal reservations to only reserve nodes 1-8 on Tues and Weds
SRCFG[sandbox] HOSTLIST=node0[1-8] FLAGS=ALLOWPRSV
SRCFG[sandbox] PERIOD=DAY DAYS=TUE,WED
```

7.1.6.4 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.

7.1.6.4.1 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 1: Bind job to reservation
7.1.6.4.2 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 1**: Configure a reservation to accept only jobs that are bound to it

```bash
> mjobctl -m flags+=advres:grid.3 job1352
```

**Example 2**: Release job from reservation binding

```bash
> mjobctl -m flags-=advres job1352
```

**Example 3**: Configure a reservation to accept a specific job

```bash
> mrsvctl -m -a JOB=3456 grid.3
```

**Example 4**: Remove a specific reservation to job binding

```bash
> mrsvctl -m -a JOB=3456 grid.3 --flags=unset
```
7.2 Partitions

- 7.2.1 Partition Overview
- 7.2.2 Defining Partitions
- 7.2.3 Managing Partition Access
- 7.2.4 Requesting Partitions
- 7.2.5 Per-Partition Settings
- 7.2.6 Miscellaneous Partition Issues

7.2.1 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, QoS’s 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.

7.2.2 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
...
```
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.

7.2.3 Managing Partition Access

Partition access can be constrained by credential ACLs and by limits based on job resource requirements.

7.2.3.1 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
USERCFG[DEFAULT] PLIST=general
USERCFG[steve]  PLIST=general:test PDEF=test
GROUPCFG[staff] PLIST=general:test PDEF=general
GROUPCFG[mgmt]  PLIST=general PDEF=general
```

By default, the system partition access list allows global access to all partitions. If using logical OR based partition access lists, the system partition list should be explicitly constrained using the SYSCFG parameter.

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 ANDed 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.

7.2.3.2 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 the PARCFG parameter as described in the following table:

<table>
<thead>
<tr>
<th>LIMIT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX.CPULIMIT</td>
<td>Specifies maximum cpulimit required per job.</td>
</tr>
</tbody>
</table>
**MAX.NODE** Specifies maximum nodes required per job.

**MAX.PROC** Specifies maximum processors required per job.

**MAX.PS** Specifies maximum nodes processor-seconds required per job.

**MAX.WCLIMIT** Specifies maximum walltime required per job (FORMAT: [[DD:]HH:]MM:SS).

**MIN.NODE** Specifies minimum nodes required per job.

**MIN.PROC** Specifies minimum processors required per job.

**MIN.WCLIMIT** Specifies minimum walltime required per job (FORMAT: [[DD:]HH:]MM:SS).

**Examples**

<table>
<thead>
<tr>
<th>PARCFG</th>
<th>MAX.PROC=16</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARCFG</td>
<td>MAX.WCLIMIT=12:00:00 MIN.PROC=4</td>
</tr>
<tr>
<td>PARCFG</td>
<td>MIN.NODE=12</td>
</tr>
</tbody>
</table>

### 7.2.4 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.

### 7.2.5 Per-Partition Settings

The following settings can be specified on a per-partition basis using the `PARCFG` parameter:

<table>
<thead>
<tr>
<th>SETTING</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOBNODEMATCHPOLICY</td>
<td>Specifies the JOBNODEMATCHPOLICY to be applied to jobs that run in the specified partition.</td>
</tr>
<tr>
<td>NODEACCESSPOLICY</td>
<td>Specifies the NODEACCESSPOLICY to be applied to jobs that run in the specified partition.</td>
</tr>
<tr>
<td>USETTC</td>
<td>Specifies whether TTC specified at submission should be used and displayed by the scheduler.</td>
</tr>
<tr>
<td>VMCREATEDURATION</td>
<td>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.</td>
</tr>
<tr>
<td>VMDELETEDURATION</td>
<td>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.</td>
</tr>
<tr>
<td>VMMIGRATEDURATION</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 7.2.6 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.
See Also

- Standing Reservations
- Node Sets
- FEATUREPARTITIONHEADER parameter
- PARCFG parameter
7.3 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 QoS's 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.

7.3.1 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 QoS's 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 QoS's. A given job will have a default QoS and may have access to several additional QoS's. 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 QoS's MEMBERLIST attribute.

The `mdiaq -q` command can be used to obtain information about the current QoS configuration including specified credential access.

7.3.2 QoS Enabled Privileges

The privileges enabled via QoS settings may be broken into the following categories:

- Special Prioritization
- Service Access and Constraints
Override Policies and Policy Exemptions

All privileges are managed via the QOSCFG parameter.

### 7.3.2.1 Special Prioritization

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSTARGET</td>
<td>Specifies QoS fairshare target.</td>
</tr>
<tr>
<td>FSWEIGHT</td>
<td>Sets QoS fairshare weight offset affecting a job's fairshare priority component.</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>Assigns priority to all jobs requesting particular QoS.</td>
</tr>
<tr>
<td>QTTARGET</td>
<td>Sets QoS queuetime target affecting a job's target priority component and QoS delivered.</td>
</tr>
<tr>
<td>QTWEIGHT</td>
<td>Sets QoS queuetime weight offset affecting a job's service priority component.</td>
</tr>
<tr>
<td>XFTARGET</td>
<td>Sets QoS XFactor target affecting a job's target priority component and QoS delivered.</td>
</tr>
<tr>
<td>XFWEIGHT</td>
<td>Sets QoS XFactor weight offset affecting a job's service priority component.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
# assign priority for all qos geo jobs
QOSCFG[geo]  PRIORITY=10000
```

### 7.3.2.2 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.

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEADLINE</td>
<td>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.)</td>
</tr>
<tr>
<td>DEDICATED</td>
<td>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.</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>Moab allows jobs to dynamically allocate/de-allocate resources while the job is actively running.</td>
</tr>
<tr>
<td>ENABLEUSERRSSV</td>
<td>Allow user or personal reservations to be created and managed.</td>
</tr>
<tr>
<td>IGNALL</td>
<td>Scheduler ignores all resource usage policies for jobs associated with this QoS.</td>
</tr>
<tr>
<td>NOBF</td>
<td>Job is not considered for backfill.</td>
</tr>
<tr>
<td>NORESERVATION</td>
<td>Job should never reserve resources regardless of priority.</td>
</tr>
<tr>
<td>NTR</td>
<td>Job is prioritized to run next.</td>
</tr>
<tr>
<td>PREEMPTCONFIG</td>
<td>User jobs may specify options to alter how preemption impacts the job such as minpreempttime.</td>
</tr>
<tr>
<td>PREEMPTTEE</td>
<td>Job may be preempted by higher priority PREEMPTOR jobs.</td>
</tr>
<tr>
<td>PREEMPTFSV</td>
<td>Job may be preempted by higher priority PREEMPTOR jobs if it exceeds its fairshare target when started.</td>
</tr>
<tr>
<td>PREEMPTOR</td>
<td>Job may preempt lower priority PREEMPTTEE jobs.</td>
</tr>
</tbody>
</table>
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 preemtpre.

RUNNOW overrides resource restrictions such as MAXJOB or MAXPROC.

TRIGGER  
The job is able to directly specify triggers.

USERESERVED[:<RSVID>]  
Job may only use resources within accessible reservations. If <RSVID> is specified, job may only use resources within the specified reservation.

Example: For lowprio QoS job, disable backfill and make job preemptible

```
QOSCFG[lowprio] QFLAGS=NOBF,PREEMPTEE
```

Example: 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.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSVCREATIONCOST</td>
<td>The base cost to users when creating reservations.</td>
</tr>
<tr>
<td>SYSPRIO</td>
<td>Sets the system priority on jobs associated with this QoS.</td>
</tr>
</tbody>
</table>

Example: 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
...
```

Setting Per QoS Network Service Levels

Minimum network service levels can be specified using the MINBANDWIDTH attribute. If specified, Moab does not allow a job to start until these criteria can be satisfied and allocates or reserves specific resources or combinations of resources to fulfill these constraints. Further, if the Network Management System supports it, Moab dynamically creates network fabric partitions with guaranteed service levels.

```
QOSCFG[highspeed] MINBANDWIDTH=1GB
...
```
7.3.2.3 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 QoS's, an override limit may also be specified and with this limit, jobs are constrained by the override, regardless of other limits specified. In particular, the following policies may be overridden:

- MAXJOB
- MAXNODE
- MAXPE
- MAXPROC
- MAXPS

Usage limits are overridden by specifying the QoS attribute `<LIMIT>`, as in `OMAXJOB` or `OMAXPROC`. To override system/job limits, use the `OMAXJ<LIMIT>`, as in `OMAXJPS` or `OMAXJWC`.

In addition to job usage limits, QoS's (as with other credentials), can also limit personal reservations. Attributes such as `RMAXDURATION`, `RMAXPROC`, and `RMAXTOTALPROC` can be used to constrain resources available to personal reservations. See Personal Reservation Usage Limits for more information.

Example

```bash
# staff QoS should have a limit of 48 jobs, ignoring the user limit
USERCFG[DEFAULT]   MAXJOB=10
QOSCFG[staff]      OMAXJOB=48
```

7.3.2.4 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:

<table>
<thead>
<tr>
<th>Threshold Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREEMPTQTHRESHOLD</td>
<td>A job with this QoS becomes a preemtorn if the specified queuetime threshold is reached.</td>
</tr>
<tr>
<td>PREEMPTXFTHRESHOLD</td>
<td>A job with this QoS becomes a preemtorn if the specified XFactor threshold is reached.</td>
</tr>
<tr>
<td>RSVQTTHRESHOLD</td>
<td>A job with this QoS can create a job reservation to guarantee resource access if the specified queuetime threshold is reached.</td>
</tr>
<tr>
<td>RSVXFTHRESHOLD</td>
<td>A job with this QoS can create a job reservation to guarantee resource access if the specified XFactor threshold is reached.</td>
</tr>
<tr>
<td>ACLQTTHRESHOLD</td>
<td>A job with this QoS can access reservations with a corresponding QoS ACL only if the specified queuetime threshold is reached.</td>
</tr>
<tr>
<td>ACLXFTHRESHOLD</td>
<td>A job with this QoS can access reservations with a corresponding QoS ACL only if the specified XFactor threshold is reached.</td>
</tr>
<tr>
<td>TRIGGERQTHRESHOLD</td>
<td>If a job with this QoS fails to run before this threshold is reached, any failure triggers associated with this QoS will fire.</td>
</tr>
</tbody>
</table>

7.3.2.5 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:
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 `PREEMPTIONPOLICY` 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 `PREEMPTIONPOLICY` 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.

### 7.3.3 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:

```plaintext
# 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 QoS's geo, chem, staff, or normal).

#### QOS Access via Logical AND

If desired, QoS access can be masked or logically ANDed if the QoS access list is specified with a terminating ampersand (`&`) as in the following example:

```plaintext
# 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 QoS's 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.

7.3.4 Configuring QoS-Specific Charging

Charging policies and rates may be configured on a per-QoS basis. This capability provides a powerful way of allowing end-users to select the level of service they desire and the price they are willing to pay for that service.

Several policies allow control over exactly how a job gets charged; these include CHARGERATEPOLICY and CHARGEMETRICPOLICY. The first parameter, CHARGERATEPOLICY, allows a site to specify how jobs are charged for the resources they use.

A job's total cost can be adjusted on a per QoS basis using the DEDRESCOST attribute:

<table>
<thead>
<tr>
<th>QOS Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEDRESCOST</td>
<td>Cost multiplier for dedicated resources associated with using the specified QoS.</td>
</tr>
</tbody>
</table>

Example

```
QOSCFG[fast] DEDRESCOST=4.0
QOSCFG[slow] DEDRESCOST=0.5
...                             
```

These charging parameters only impact internal charging job costs written to accounting records. They do not currently affect charges sent to an external allocation manager.

7.3.5 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 submittor 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
```

7.3.6 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
```

See Also
- Credential Overview
- Allocation Management Overview
- Rollback Reservations
- Job Deadlines
8.0 Optimizing Scheduling Behavior - Backfill, Node Sets, and Preemption

- 8.1 Optimization Overview
- 8.2 Backfill
- 8.3 Node Sets
- 8.4 Preemption
8.1 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:

- **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 task distribution, node sets and node allocation)
- **Job Preemption** - preempting jobs to allow the most important jobs to receive the best response time (see preemption)
- **Utilizing Flexible Policies** - using policies that minimize blocking and resource fragmentation while enforcing needed constraints (see soft throttling policies and reservations)
8.2 Backfill

- 8.2.1 Backfill Overview
- 8.2.2 Backfill Algorithms
- 8.2.3 Configuring Backfill

8.2.1 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.

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 B 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.

### 8.2.2 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, lets 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 C). The second window contains only one node but has no time limit because this node is not blocked by the reservation for job C. It is important to note that these backfill windows overlap.

![Backfill Windows Diagram](image)

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 the **BESTFIT** algorithm is applied, the following steps are taken:

1. The list of feasible backfill jobs is filtered, selecting only those that actually fit in the current backfill window.
2. The *degree of fit* of each job is determined based on the **BACKFILLMETRIC** parameter (processors, seconds, processor-seconds).
3. The job with the best fit starts.
4. While backfill jobs and idle resources remain, repeat step 1.

If the **GREEDY** algorithm is applied, the following steps are taken:

1. The list of feasible backfill jobs is filtered, selecting only those that actually fit in the current backfill window.
2. All possible combinations of jobs are evaluated, and the *degree of fit* of each combination is determined based on the **BACKFILLMETRIC** parameter (processors, seconds, processor-seconds).
3. Each job in the combination with the best fit starts.
4. While backfill jobs and idle resources remain, repeat step 1.

If the **PREEMPT** algorithm is applied, the following steps are taken:

1. The list of feasible backfill jobs is filtered, selecting only those that actually fit in the current backfill window.
2. Jobs are filtered according to the priority set by the **BFPRIORITYPOLICY** parameter.
3. The highest priority backfill job is started as a preemptee.
4. 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.

### 8.2.2.1 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.

### 8.2.3 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 **BESTFIT**, **GREEDY**, **PREEMPT** or **NONE** (disabled).

**Reservations**

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 set to PREEMPT, the virtually scaled job stops execution and re-queues.

If the BFVIRTUALWALLTIMECONFLICTPOLICY parameter is set to NONE or is not specified, the overlapped job reservations are rescheduled.

See Also

- BACKFILLDEPTH Parameter
- BACKFILLMETRIC Parameter
- BFMINVIRTUALWALLTIME
- Reservation Policy Overview
8.3 Node Set Overview

- 8.3.1 Node Set Usage Overview
- 8.3.2 Node Set Configuration
  - 8.3.2.1 Node Set Policy
  - 8.3.2.2 Node Set Attribute
  - 8.3.2.3 Node Set Constraint Handling
  - 8.3.2.4 Node Set List
  - 8.3.2.5 Node Set Tolerance
  - 8.3.2.6 Node Set Priority
  - 8.3.2.7 NODESETPLUS
  - 8.3.2.8 Nested Node Sets
- 8.3.3 Requesting Node Sets for Job Submission
- 8.3.4 Configuring Node Sets for Classes

8.3.1 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 and can be based on node processor speed, memory, network interfaces, or locally defined node attributes. 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 \texttt{ANYOF:PROCSPEED:450,500,650} to request nodes in the range of 450 to 650 MHz. Alternatively, if a simple procspeed-homogeneous node set is desired, \texttt{ONEOF:PROCSPEED} may be specified. On the other hand, a communication sensitive job may request a network based node set with the configuration \texttt{ONEOF:NETWORK:via,myrinet,ethernet}, in which case Moab will first attempt to locate adequate nodes where all nodes contain via network interfaces. If such a set cannot be found, Moab will look for sets of nodes containing the other specified network interfaces. 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 \texttt{NODESET*} parameters while per job specification occurs via the resource manager extensions. In all cases, node sets are a dynamic construct, created on a per job basis and built only of nodes that meet all of a job's requirements.

\begin{itemize}
\item The \texttt{GLOBAL} node is included in all feature node sets.
\end{itemize}

8.3.2 Node Set Configuration

Global node sets are defined using the \texttt{NODESETPOLICY, NODESETATTRIBUTE, NODESETLIST, NODESETISOPTIONAL, and NODESETTOLERANCE} parameters.

The use of these parameters may be best highlighted with an 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:
8.3.2.1 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 nodeset policies are listed in the following table:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANYOF</td>
<td>Select resources from all sets contained in node set list.</td>
</tr>
<tr>
<td>FIRSTOF</td>
<td>Select resources from first set to match specified constraints.</td>
</tr>
<tr>
<td>ONEOF</td>
<td>Select all sets that contain adequate resources to support job.</td>
</tr>
</tbody>
</table>

8.3.2.2 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.

<table>
<thead>
<tr>
<th>Set Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCH</td>
<td>Base node set boundaries on node's architecture attribute.</td>
</tr>
<tr>
<td>CLASS</td>
<td>Base node set boundaries on node's classes.</td>
</tr>
<tr>
<td>FEATURE</td>
<td>Base node set boundaries on node's node feature attribute.</td>
</tr>
<tr>
<td>MEMORY</td>
<td>Base node set boundaries on node's configured real memory.</td>
</tr>
<tr>
<td>PROCSPEED</td>
<td>Base node set boundaries on node's reported processor speed.</td>
</tr>
</tbody>
</table>

8.3.2.3 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.

8.3.2.4 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.

8.3.2.5 Node Set Tolerance

On occasion, site administrators may want to allow a less strict interpretation of nodes sets. In particular, many sites seek to enforce a more liberal `PROCSPEED` based node set policy, where almost balanced node allocations are allowed but wildly varying node allocations are not. In such cases, the parameter `NODESETTOLERANCE` may be used. This parameter allows specification of the percentage difference between the fastest and slowest node that can be within a nodeset using the following calculation:

\[
\frac{\text{Speed.Max} - \text{Speed.Min}}{\text{Speed.Min}} \leq \text{NODESETTOLERANCE}
\]
Thus setting \texttt{NODESETTOLERANCE} to 0.5 would allow the fastest node in a particular node set to be up to 50\% faster than the slowest node in that set. With a 0.5 setting, a job may allocate a mix of 500 and 750 MHz nodes but not a mix of 500 and 900 MHz nodes. Currently, tolerances are only supported when the \texttt{NODESETATTRIBUTE} parameter is set to \texttt{PROCSPACE}. The \texttt{MAXBALANCE} node allocation algorithm is often used in conjunction with tolerance based node sets.

\subsection*{8.3.2.6 Node Set Priority}

When resources are available in more than one resource set, the \texttt{NODESETPRIORITYTYPE} parameter allows control over how the best resource set is selected. Legal values for this parameter are described in the following table:

<table>
<thead>
<tr>
<th>Priority Type</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{AFFINITY}</td>
<td>Select the resource set that does not have a negative affinity reservation.</td>
<td>Jobs will avoid resource sets that have a negative \texttt{affinity}.</td>
</tr>
<tr>
<td>\texttt{BESTFIT}</td>
<td>Select the smallest resource set possible.</td>
<td>Minimizes fragmentation of larger resource sets.</td>
</tr>
<tr>
<td>\texttt{BESTRESOURCE}</td>
<td>Select the resource set with the best nodes.</td>
<td>Only supported when \texttt{NODESETATTRIBUTE} is set to \texttt{PROCSPACE}. Selects the fastest possible nodes for the job.</td>
</tr>
<tr>
<td>\texttt{MINLOSS}</td>
<td>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.)</td>
<td>\texttt{MINLOSS} works only when using one of the following configurations:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \texttt{NODESETATTRIBUTE FEATURE}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \texttt{NODESETATTRIBUTE PROCSPACE}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \texttt{NODESETTOLERANCE} is &gt;0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This algorithm is highly useful in environments with mixed speed compute nodes and a non load-balancing parallel workload.</td>
</tr>
<tr>
<td>\texttt{WORSTFIT}</td>
<td>Select the largest resource set possible.</td>
<td>Minimizes the creation of small resource set fragments but fragments larger resource sets.</td>
</tr>
</tbody>
</table>

\subsection*{8.3.2.7 NODESETPLUS}

Moab supports additional NodeSet behavior by specifying the \texttt{NODESETPLUS} parameter. Possible values when specifying this parameter are \texttt{SPANEVENLY} and \texttt{DELAY}.

\textbf{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 \texttt{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 \texttt{NODESETDELAY}.

\textbf{DELAY}

Moab attempts to fit all jobs within the best possible SMP machine (when scheduling nodeboards in an Altix environment) unless doing so delays the job beyond the requested \texttt{NODESETDELAY}.
8.3.2.8 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,
```

The accompanying cluster would look like this:

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).

8.3.3 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.

8.3.4 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
...`

See Also

- Resource Manager Extensions
- CLASSCFG
- Partition Overview
8.4 Preemption Management

- 8.4.1 Enabling Preemption
  - 8.4.1.1 Manual Preemption with Admin Commands
  - 8.4.1.2 QoS Based Preemption
  - 8.4.1.3 Preemption Based Backfill
  - 8.4.1.4 Trigger Based and Context Sensitive Job Preemption
- 8.4.2 Types of Preemption
  - 8.4.2.1 Job Requeue
  - 8.4.2.2 Job Suspend
  - 8.4.2.3 Job Checkpoint
  - 8.4.2.4 Job Cancel
  - 8.4.2.5 Resource Manager Preemption Constraints
- 8.4.3 Testing and Troubleshooting Preemption

Many sites possess workloads of varying importance. While it may be critical that some jobs obtain resources immediately, other jobs are less turnaround time sensitive but have an insatiable hunger for compute cycles, consuming every available cycle. These latter jobs often have turnaround times on the order of weeks or months. The concept of cycle stealing handles such situations well and enables systems to run low priority, preemptible jobs whenever something more pressing is not running. These other systems are often employed on compute farms of desktops where the jobs must vacate anytime interactive system use is detected.

8.4.1 Enabling Preemption

Preemption can be enabled in one of three ways. These include manual intervention, QoS based configuration, and use of the preemption based backfill algorithm. Note that for all of these cases, a single preemptor is limited to 32 preemptees.

8.4.1.1 Admin Preemption Commands

The mjobctl command can be used to preempt jobs. Specifically, the command can be used to modify a job’s execution state in the following ways:

<table>
<thead>
<tr>
<th>Action</th>
<th>Flag</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancel</td>
<td>-c</td>
<td>Terminate and remove job from queue.</td>
</tr>
<tr>
<td>Checkpoint</td>
<td>-C</td>
<td>Terminate and checkpoint job leaving job in queue.</td>
</tr>
<tr>
<td>Requeue</td>
<td>-R</td>
<td>Terminate job leaving job in queue.</td>
</tr>
<tr>
<td>Resume</td>
<td>-r</td>
<td>Resume suspended job.</td>
</tr>
<tr>
<td>Start (execute)</td>
<td>-x</td>
<td>Start idle job.</td>
</tr>
<tr>
<td>Suspend</td>
<td>-s</td>
<td>Suspend active job.</td>
</tr>
</tbody>
</table>

In general, users are allowed to suspend or terminate jobs they own. Administrators are allowed to suspend, terminate, resume, and execute any queued jobs.

8.4.1.2 QoS Based Preemption

Moab's QoS-based preemption system allows a site the ability to specify preemption rules and control access to preemption privileges. These abilities can be used to increase system throughput, improve job response time for specific classes of jobs, or enable various political policies. All policies are enabled by specifying some QoS’s with the flag PREEMPTOR, and others with the flag PREEMPTEE. For example, to enable a cycle stealing high throughput cluster, a QoS can be created for high priority jobs and marked with the flag PREEMPTOR; another QoS can be created for low priority jobs and marked with the flag PREEMPTEE.

If desired, the RESERVATIONPOLICY parameter can be set to NEVER. With this configuration, low priority, preemptee jobs can be started whenever idle resources are available. These jobs are allowed to run until a
To specify the desired type of preemption, use the PREEMPTPOLICY parameter.

It is important to note the rules of QoS based preemption. Preemption only occurs when the following 3 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.

Use of the preemption system need not be limited to controlling low priority jobs. Other uses include optimistic scheduling and development job support.

**Example:**

In the below example, high priority jobs are configured to always be able to preempt low priority jobs but not med or other high priority jobs.

```
PREEMPTPOLICY REQUEUE
# enable qos priority to make preemptors higher priority than preemptees
QOSWEIGHT 1
QOSCFG[high] QFLAGS=PREEMPTOR PRIORITY=1000
QOSCFG[med]
QOSCFG[low] QFLAGS=PREEMPTEE
# associate class 'special' with QOS high
CLASSCFG[special] QDEF=high&
```

As in the previous example, any class can be bound to a particular QoS using the QDEF attribute of the CLASSCFG parameter with the & marker.

**Preventing Thrashing**

In environments where job checkpointing or job suspension incur significant overhead, it may be desirable to constrain the rate at which job preemption is allowed. The parameter JOBPREEMPTMINACTIVETIME can be used 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 the specified time frame. Conversely, jobs can be excluded from preemption after running for a certain amount of time using the JOBPREEMPTMAXACTIVETIME parameter.

**8.4.1.3 Preemption Based Backfill**

The PREEMPT backfill policy allows a site to take advantage of optimistic scheduling. By default, backfill only allows jobs to run if they are guaranteed to have adequate time to run to completion. However, statistically, most jobs do not use their full requested wallclock limit. The PREEMPT backfill policy allows the scheduler to start backfill jobs even if required walltime is not available. If the job runs too long and interferes with another job that was guaranteed a particular timeslot, the backfill job is preempted and the priority job is allowed to run. When another potential timeslot becomes available, the preempted backfill job will again be optimistically executed. In environments with checkpointing or with poor wallclock accuracies, this algorithm has potential for significant savings. See the backfill section for more information.

**8.4.1.4 Trigger and Context Based Preemption Policies**

Rules regarding which jobs can be preemptors and which are preemptees can be configured to take into
account aspects of the compute environment. Some of these context sensitive rules are listed here:

- Mark a job a preemptor if its delivered or expected response time exceeds a specified threshold.
- Mark a job preemptible if it violates soft policy usage limits or fairshare targets.
- Mark a job a preemptor if it is running in a reservation it owns.
- Preempt a job as the result of a specific user, node, job, reservation, or other object event using object triggers.
- Preempt a job as the result of an external generic event or generic metric.

### 8.4.2 Types of Preemption

How the scheduler preempts a job is controlled by the PREEMPTPOLICY parameter. This parameter allows preemption to be enforced using one of the following methods: suspend, checkpoint, requeue, or cancel.

Moab uses preemption escalation to free up resources. This means if the PREEMPTPOLICY is set to **suspend**, then Moab will use this method if available but will escalate to something potentially more disruptive if necessary to preempt and free up resources. The precedence of preemption methods from least to most disruptive is **suspend**, **checkpoint**, **requeue**, and **cancel**.

#### 8.4.2.1 Job Requeue

Under this policy, active jobs are terminated and returned to the job queue in an idle state.

For a job to be requeued, it must be marked as **restartable**. If not, it will be canceled. If supported by the resource manager, the job restartable flag can be set when the job is submitted by using the `msub -r` option. Otherwise, this can be accomplished using the FLAGS attribute of the associated class or QoS credential.

```
CLASSCFG[low] JOBFLAGS=RESTARTABLE
```

#### 8.4.2.2 Job Suspend

Suspend causes active jobs to stop executing but to remain in memory on the allocated compute nodes. While a suspended job frees up processor resources, it may continue to consume swap and other resources. Suspended jobs must be resumed to continue executing.

If suspend based preemption is selected, then the signal used to initiate the job suspend may be specified by setting the SUSPENDSIG attribute of the RMCFG parameter.

For a job to be suspended, it must be marked as **suspendable**. If not, it will be requeued or canceled. If supported by the resource manager, the job suspendable flag can be set when the job is submitted. Otherwise, this can be accomplished using the JOBFLAGS attribute of the associated class credential as in the following example:

```
CLASSCFG[low] JOBFLAGS=SUSPENDABLE
```

#### 8.4.2.3 Job Checkpoint

Systems that support job checkpointing allow a job to save off its current state and either terminate or continue running. A checkpointed job may be restarted at any time and resume execution from its most recent checkpoint.

Checkpointing behavior can be tuned on a per resource manager basis by setting the CHECKPOINTSIG and CHECKPOINNTTIMEOUT attributes of the RMCFG parameter.

See Checkpoint/Restart Facilities for more information.

#### 8.4.2.4 Job Cancel

Under this policy, active jobs are canceled.
8.4.2.5 RM Preemption Constraints

Moab is only able to use preemption if the underlying resource manager/OS combination supports this capability. The following table displays current preemption limitations:

<table>
<thead>
<tr>
<th>Resource Manager</th>
<th>TORQUE 1.2+/OpenPBS 2.3+</th>
<th>PBSPro 5.2</th>
<th>Loadleveler 3.1</th>
<th>LSF 5.2</th>
<th>SGE 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancel</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>???</td>
</tr>
<tr>
<td>Requeue</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>???</td>
</tr>
<tr>
<td>Suspend</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>???</td>
</tr>
<tr>
<td>Checkpoint</td>
<td>(yes on IRIX)</td>
<td>(yes on IRIX)</td>
<td>yes</td>
<td></td>
<td>???</td>
</tr>
</tbody>
</table>

8.4.3 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 preemptors to preempt preemptees in the right way. To diagnose this, use the following checklist:

- Are preemptor jobs marked with the PREEMPTOR flag (verify with checkjob <JOBID> | grep Flags)?
- Are preemptee jobs marked with the PREEMPTEE flag (verify with checkjob <JOBID> | grep Flags)?
- Is the start priority of the preemptor higher than the priority of the preemptee (verify with checkjob <JOBID> | grep Priority)?
- Do the resources allocated to the preemptee match those requested by the preemptor?
- Is the preemptor within the 32-preemptee limit?
- Are any policies preventing preemption from occurring (verify with checkjob -v -n <NODEID> <JOBID>)?
- Is the PREEMPTPOLICY parameter properly set?
- Is the preemptee properly marked as restartable, suspendable, or checkpointable (verify with checkjob <JOBID> | grep Flags)?
- Is the resource manager properly responding to preemption requests (use mdiag -R)?
- If there is a resource manager level race condition, is Moab properly holding target resources (verify with mdiag -S and set RESERVATIONRETRYTIME if needed)?

See Also

- QoS Overview
- Managing QoS Access (control who can preempt)
- JOBMAXPREEMPTPERITERATION parameter
- Job Preemption with Reservation Triggers
- Checkpoint Overview
- ENABLESPVIOLATIONPREEMPTION parameter
- PREEMPTPROJOBSELECTWEIGHT parameter (adjust cost of preemptable jobs)
- PREEMPTSEARCHDEPTH parameter
- USAGEEXECUTIONTIMEWEIGHT parameter (control priority of suspended jobs)
- IGNOREPREEMPTPRIORITY parameter (relative job priority is ignored in preemption decisions)
- DISABLSAMEQOSPREEMPTION parameter (jobs cannot preempt other jobs with the same QOS)
- PREEMPTRTIMEWEIGHT parameter (add remaining time of jobs to preemption calculation)
9.0 Evaluating System Performance - Statistics, Profiling, Testing, and Simulation

- 9.1 Moab Performance Evaluation Overview
- 9.2 Job and System Statistics
- 9.3 Testing New Versions and Configurations
- 9.4 Answering *What If?* Questions with the Simulator
9.1 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.
9.2 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.

- 9.2.1 Accounting Overview
- 9.2.2 Real-Time Statistics
- 9.2.3 FairShare Usage Statistics

9.2.1 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.

9.2.1.1 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 16.3.3 Workload Traces.

9.2.1.2 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.

9.2.1.3 Credential Accounting

Current and historical usage for users, groups, account, QoS's, 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
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.

### 9.2.2 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 `resetstats` 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.

### 9.2.3 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.

### See Also

- Simulation Overview
- Generic Consumable Resources
- Object Variables
- Generic Event Counters
9.3 Testing New Versions and Configurations

- 9.3.1 MONITOR Mode
- 9.3.2 INTERACTIVE Mode

9.3.1 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.

9.3.2 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.

See Also

- Testing New Releases and Policies
- Cluster Simulations
- Side-by-Side Mode
9.4 Answering *What If?* Questions with the Simulator

Moab Workload Manager can answer hypothetical situations through simulations. (See 16.0 Simulations.) Once Resource and Workload Traces have been collected, any number of configurations can be tested without disturbing the system.
10.0 Managing Shared Resources - Usage Based Limits, Policies, and SMP Issues

- 10.1 Consumable Resource Handling
- 10.2 Load Balancing Features
- 10.3 Resource Usage Limits
- 10.4 General SMP Issues
### 10.1 Consumable Resource Handling

Moab is designed to inherently handle consumable resources. Nodes possess resources, and workload consumes resources. Moab tracks any number of consumable resources on a per node and per job basis. Floating cluster resources can be handled as well; see [Managing Shared Cluster Resources](#). When a job is started on a set of nodes, Moab tracks how much of each available resource must be dedicated to the tasks of the job. This allows Moab to prevent per node over-subscription of any resource, be it CPU, memory, swap, or local disk.

Recent resource managers (such as Loadleveler version 3.x) provide the ability to exercise this capability. These capabilities allow a user to specify per task consumable resources and per node available resources. For example, a job may be submitted requiring 20 tasks with 2 CPUs and 256 MB per task. Thus, Moab would allow a node with 1 GB of memory and 16 processors to run 4 of these tasks because 4 tasks would consume all of the available memory. Consumable resources allow more intelligent allocation of resources allowing better management of shared node resources.

No scheduler level configuration is required to enable this capability as Moab detects the needed information automatically from the underlying resource manager.

**See Also**

- [Managing Generic Consumable Resources](#)
- [Floating Generic Resources](#)
10.2 Load Balancing Features

Load balancing is generally defined as the incorporation of resource load information into scheduling decisions. Moab supports load balancing in a number of ways allowing sites to use node load information to both determine resource availability and to control job resource allocation.

10.2.1 Resource Availability

Moab only schedules jobs onto available nodes. Using Moab's node availability policies, a site can specify exactly what criteria determine the node's availability. For load balancing purposes, site administrators may want to configure availability criteria for processors, memory, and swap. Various settings can enable over-committing resources if desired while others can constrain nodes to only accept jobs if resources exist to meet the maximum needs of all concurrent job requests.

10.2.2 Prioritizing Node Allocation

The second major aspect of load balancing has to do with the selection of resources for new jobs. With Moab, load information can be incorporated into the node allocation decision by using the PRIORITY node allocation policy. This policy allows specification of which aspects of a node's configuration contribute to its allocation priority. For load balancing purposes, a site would want to favor nodes with the most available processors and the lowest load and job count. The node allocation priority function is set using the PRIORITYF attribute of the NODECFG parameter as shown in the following example:

```
NODEALLOCATIONPOLICY    PRIORITY
NODECFG[DEFAULT]         PRIORITYF='10 * APROCS - LOAD - JOBCOUNT'
```

Other node aspects that may be of value in configuring load-balancing based node allocation include SPEED and CPROCS.

See Also

- NODEAVAILABILITYPOLICY parameter
- NODEMAXLOAD parameter
10.3 Resource Usage Limits

- 10.3.1 Configuring Actions
- 10.3.2 Specifying Hard and Soft Policy Violations
- 10.3.3 Constraining Walltime Usage

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.

10.3.1 Configuring Actions

The RESOURCELIMITPOLICY parameter accepts a number of policies, resources, and actions using the format and values defined below.

Format

```
RESOURCELIMITPOLICY <RESOURCE>[:<SPOLICY>,]<HPOLICY>[:<SACTION>,]<HACTION>[:<SVIOLATIONTIME>,<HVIOLATIONTIME>]...
```

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU TIME</td>
<td>Maximum total job proc-seconds used by any single job (allows scheduler enforcement of cpulimit).</td>
</tr>
<tr>
<td>DISK</td>
<td>Local disk space (in MB) used by any single job task.</td>
</tr>
<tr>
<td>JOBMEM</td>
<td>Maximum real memory/RAM (in MB) used by any single job.</td>
</tr>
<tr>
<td>JOBPROC</td>
<td>Maximum processor load associated with any single job.</td>
</tr>
<tr>
<td>MEM</td>
<td>Maximum real memory/RAM (in MB) used by any single job task.</td>
</tr>
<tr>
<td>MINJOBPROC</td>
<td>Minimum processor load associated with any single job (action taken if job is using 5% or less of potential CPU usage).</td>
</tr>
<tr>
<td>NETWORK</td>
<td>Maximum network load associated with any single job task.</td>
</tr>
<tr>
<td>PROC</td>
<td>Maximum processor load associated with any single job task.</td>
</tr>
<tr>
<td>SWAP</td>
<td>Maximum virtual memory/SWAP (in MB) used by any single job task.</td>
</tr>
<tr>
<td>WALLTIME</td>
<td>Requested job walltime.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALWAYS</td>
<td>take action whenever a violation is detected</td>
</tr>
<tr>
<td>EXTENDEDVIOLATION</td>
<td>take action only if a violation is detected and persists for greater than the specified time limit</td>
</tr>
<tr>
<td>BLOCKEDWORKLOADONLY</td>
<td>take action only if a violation is detected and the constrained resource is required by another job</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANCEL</td>
<td>terminate the job</td>
</tr>
<tr>
<td>CHECKPOINT</td>
<td>checkpoint and terminate job</td>
</tr>
<tr>
<td>MIGRATE</td>
<td>requeue the job and requires a different set of nodes for execution</td>
</tr>
<tr>
<td>NOTIFY</td>
<td>notify admins and job owner regarding violation</td>
</tr>
</tbody>
</table>
REQUEUE  terminate and requeue the job

SUSPEND  suspend the job and leave it suspended for an amount of time defined by the X parameter

**Example - 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 - 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 - Cancel jobs that use 5% or less of potential CPU usage for more than 5 minutes**

```
RESOURCELIMITPOLICY MINJOBPROC:EXTENDEDVIOLATION:CANCEL:5:00
```

**Example - Migrating a job when it blocks other workload**

```
RESOURCELIMITPOLICY JOBPROC:BLOCKEDWORKLOADONLY:MIGRATE
```

### 10.3.2 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
```

### 10.3.3 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 `*CFG` credential parameters.

```
JOBMAXOVERRUN 00:10:00
CLASSCFG[debug] overrun=00:00:30
```
- **Usage Limits/Throttling Policies**
- **JOBMAXOVERRUN** parameter
- **WCVIOLATIONACTION** parameter
- **RESOURCESLIMITMULTIPLIER** parameter
10.4 General SMP Issues

Shared vs Dedicated

SMP nodes are often used to run jobs that do not use all available resources on that node. How Moab handles these unused resources is controlled by the parameter NODEACCESSPOLICY. If this parameter is set to SHARED, Moab allows tasks of other jobs to use the resources.
11.0 General Job Administration

- 11.1 Job Holds
- 11.2 Job Priority Management
- 11.3 Suspend/Resume Handling
- 11.4 Checkpoint/Restart Facilities
- 11.5 Job Dependencies
- 11.6 Setting Job Defaults and Per Job Limits
- 11.7 General Job Policies
- 11.8 Using a Local Queue
- 11.9 Job Deadlines
- 11.10 Job Templates
- 11.11 Job Arrays
11.1 Job Holds

11.1.1 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.

11.1.2 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.

11.1.3 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.

11.1.4 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.

11.1.5 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.

See Also

- DEFERSTARTCOUNT - number of job start failures allowed before job is deferred
11.2 Job Priority Management

Job priority management is controlled via both configured and manual intervention mechanisms.

- Priority Configuration - see Job Prioritization
- Manual Intervention with setspri
11.3 Suspend/Resume Handling

When supported by the Resource Manager, Moab can suspend and resume jobs. By default, a job is suspended for one minute before it can resume. You can modify this default time using the \texttt{MINADMINSTIME} parameter.

Moab also supports both manual and automatic job preemption, topics covered in greater detail in the following sections:

- manual preemption with the \texttt{mjobctl} command
- QoS based job preemption
- Preemption based backfill
11.4 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.

See Also

- Job Preemption Overview
- PREEMPTPOLICY Parameter
- Resource Manager CHECKPOINTSIG Attribute
- Resource Manager CHECKPOINTTIMEOUT Attribute
11.5 Job Dependencies

- 11.5.1 Basic Job Dependency Support
  - 11.5.1.1 Job Dependency Syntax

11.5.1 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 TORQUE's `qsub` and the Moab `msub` command, the semantics listed in the section below can be used with the `-W x=depend:<STRING>` flag. For other resource managers, consult the resource manager specific documentation.

Situations can arise where idle job limits are set and the dependee is blocked out. To avoid this, use the `BLOCKLIST DEPEND` parameter.

11.5.1.1 Job Dependency Syntax

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>after</td>
<td>after:&lt;job&gt;[:&lt;job&gt;]]...</td>
<td>Job may start at any time after specified jobs have started execution.</td>
</tr>
<tr>
<td>afterany</td>
<td>afterany:&lt;job&gt;[:&lt;job&gt;]]...</td>
<td>Job may start at any time after all specified jobs have completed regardless of completion status.</td>
</tr>
<tr>
<td>afterok</td>
<td>afterok:&lt;job&gt;[:&lt;job&gt;]]...</td>
<td>Job may be start at any time after all specified jobs have successfully completed.</td>
</tr>
<tr>
<td>afternotok</td>
<td>afternotok:&lt;job&gt;[:&lt;job&gt;]]...</td>
<td>Job may start at any time after all specified jobs have completed unsuccessfully.</td>
</tr>
<tr>
<td>before</td>
<td>before:&lt;job&gt;[:&lt;job&gt;]]...</td>
<td>Job may start at any time before specified jobs have started execution.</td>
</tr>
<tr>
<td>beforeany</td>
<td>beforeany:&lt;job&gt;[:&lt;job&gt;]]...</td>
<td>Job may start at any time before all specified jobs have completed regardless of completion status.</td>
</tr>
<tr>
<td>beforeok</td>
<td>beforeok:&lt;job&gt;[:&lt;job&gt;]]...</td>
<td>Job may start at any time before all specified jobs have successfully completed.</td>
</tr>
<tr>
<td>beforenotok</td>
<td>beforenotok:&lt;job&gt;[:&lt;job&gt;]]...</td>
<td>Job may start at any time before any specified jobs have completed unsuccessfully.</td>
</tr>
<tr>
<td>on</td>
<td>on:&lt;count&gt;</td>
<td>Job may start after &lt;count&gt; dependencies on other jobs have been satisfied.</td>
</tr>
</tbody>
</table>

* `synccount` and `syncwith` are not currently supported by Moab.

* `<job>={jobname|jobid}`

* 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.
See Also

- Job Deadlines
11.6 Job Defaults and Per Job Limits

11.6.1 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT.FEATURES</td>
<td>comma delimited list of node features</td>
<td>CLASSCFG[batch] DEFAULT.FEATURES=fast,io (jobs submitted to class batch will request nodes fast and io)</td>
</tr>
<tr>
<td>DEFAULT.WCLIMIT</td>
<td>[[DD:]HH:]MM:]SS</td>
<td>CLASSCFG[batch] DEFAULT.WCLIMIT=1:00:00 (jobs submitted to class batch will request one hour of walltime by default.)</td>
</tr>
</tbody>
</table>

11.6.2 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX.WCLIMIT</td>
<td>[[DD:]HH:]MM:]SS</td>
<td>CLASSCFG[batch] MAX.WCLIMIT=1:00:00 (jobs submitted to class batch can request no more than one hour of walltime.)</td>
</tr>
</tbody>
</table>

11.6.3 Per Job Minimum Limits

Furthermore, minimum job defaults can be specified with the CLASSCFG parameter. The following table shows the applicable attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN.PROC</td>
<td>&lt;integer&gt;</td>
<td>CLASSCFG[batch] MIN.PROC=10 (jobs submitted to class batch can request no less than ten processors.)</td>
</tr>
</tbody>
</table>

See Also

- Resource Usage Limits
11.7 General Job Policies

- 11.7.1 Multi-Node Support
- 11.7.2 Multi-Req Support
- 11.7.3 Job Size Policy
- 11.7.4 Malleable Job Support
- 11.7.5 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.

11.7.1 Multi-Node Support

You can configure the ability to allocate resources from multiple nodes to a job with the MAX.NODE limit.

11.7.2 Multi-Req Support

By default, jobs are only allowed to specify a single type of resource for allocation. For example, a job could request 4 nodes with 256 MB of memory or 8 nodes with feature fast present. However, the default behavior does not allow submission of a single job that requests both of these resource types. The parameter ENABLEMULTIREQJOBS can be set to TRUE to remove this constraint.

11.7.3 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.

11.7.4 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.

11.7.5 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.

SSH Proxy Settings
In the example above, 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.

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 dynamic modified, or rejected in a custom manner depending on the site or the nature of the failure.

**See Also**

- [Usage Limits](#)
# 11.8 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.

## 11.8.1 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:

<table>
<thead>
<tr>
<th>Default queue</th>
<th>Format:</th>
<th>Description:</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RMCFG[internal] DEFAULTCLASS=&lt;CLASSID&gt;</td>
<td>The job class/queue assigned to the job if one is not explicitly requested by the submittor.</td>
<td>RMCFG[internal] DEFAULTCLASS=batch</td>
</tr>
</tbody>
</table>

- **Class default resource requirements**

<table>
<thead>
<tr>
<th>Format:</th>
<th>Description:</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASSCFG[&lt;CLASSID&gt;] DEFAULT.FEATURES=&lt;X&gt;</td>
<td>The settings assigned to the job if not explicitly set by the submittor. Default values are available for node features, per task memory, node count, nodeset configuration, processor count, and wallclock limit.</td>
<td>CLASSCFG[batch] DEFAULT.WCLIMIT=4 DEFAULT.FEATURES=matlab</td>
</tr>
<tr>
<td>CLASSCFG[&lt;CLASSID&gt;] DEFAULT.MEM=&lt;X&gt;</td>
<td></td>
<td>CLASSCFG[batch] DEFAULT.WCLIMIT=4</td>
</tr>
<tr>
<td>CLASSCFG[&lt;CLASSID&gt;] DEFAULT.NODE=&lt;X&gt;</td>
<td></td>
<td>CLASSCFG[batch] DEFAULT.FEATURES=matlab</td>
</tr>
<tr>
<td>CLASSCFG[&lt;CLASSID&gt;] DEFAULT.NODESET=&lt;X&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLASSCFG[&lt;CLASSID&gt;] DEFAULT.PROC=&lt;X&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLASSCFG[&lt;CLASSID&gt;] DEFAULT.WCLIMIT=&lt;X&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Class maximum resource limits**

<table>
<thead>
<tr>
<th>Format:</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASSCFG[&lt;CLASSID&gt;] MAX.FEATURES=&lt;X&gt;</td>
<td></td>
</tr>
<tr>
<td>CLASSCFG[&lt;CLASSID&gt;] MAX.NODE=&lt;X&gt;</td>
<td></td>
</tr>
</tbody>
</table>
**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.
11.9 Job Deadlines

11.9.1 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.

11.9.2 Absolute Job Deadlines

A job may request a specific completion time if, and only if, it requests and is allowed to access a QoS with the DEADLINE flag set. If so, a 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. For example, consider the following configuration which sets a deadline for a job to finish by 8 a.m. on March 1st, 2008:

```
msub -l deadline=08:00:00_03/01/08
```

11.9.3 Relative Job Deadlines

QoS's may be set up with both the DEADLINE flag and a response time target. For job's requesting these qualities of service, Moab identifies and sets job deadlines to satisfy the corresponding response time targets. For example, consider the following configuration which sets a queue time response target of 1 hour:

```
QOS_CFG[ special ] QFLAGS=DEADLINE QTTARGET=1:00:00
```

Given this configuration, a two-hour job requesting QoS special has a completion time deadline set to 3 hours after the job's submission time.

11.9.4 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.

11.9.5 Conflict Policies

When a job cannot make a requested deadline Moab, by default, sets a hold on the job. The specific policy can be configured using the DEADLINEPOLICY parameter.

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANCEL</td>
<td>The job is canceled and the user is notified that the deadline could not be satisfied.</td>
</tr>
<tr>
<td>HOLD</td>
<td>The job has a batch hold placed on it indefinitely. The administrator can then decide what action to take.</td>
</tr>
<tr>
<td>RETRY</td>
<td>The job continually retries each iteration to meet its deadline; note that when used with QTTARGET</td>
</tr>
</tbody>
</table>
the job's deadline continues to slide with relative time.

ignore The job has its request ignored and is scheduled as normal.

See Also

- QoS Facilities
- Job Submission Eligible Start Time constraints
11.10 Job Templates

- 11.10.1 Job Template Overview
- 11.10.2 Applying Job Templates
  - 11.10.2.1 Matching
  - 11.10.2.2 Selecting
- 11.10.3 Job Template Extension Attributes
- 11.10.4 Resource Manager Based Templates
- 11.10.5 Job Template Examples
- 11.10.6 Managing Job Templates

11.10.1 Job Template Overview

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. Job templates can be used in many aspects of scheduling but they are most commonly applied in the area of Peer Based Grid usage policy. Job templates are defined using the JOBCFG configuration parameter.

11.10.2 Applying Job Templates

11.10.2.1 Matching

The JOBMATCHCFG parameter allows relationships to be established 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. The table in section 11.10.3 Job Template Extension Attributes 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMAX</td>
<td>A potential job is rejected if it has matching attributes set or has resource requests that exceed those specified in this template.</td>
</tr>
<tr>
<td>JMIN</td>
<td>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.</td>
</tr>
<tr>
<td>JDEF</td>
<td>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.</td>
</tr>
<tr>
<td>JSET</td>
<td>A matching job has the specified attributes forced to these values and these values override any values specified by the submittor at job submission time.</td>
</tr>
<tr>
<td>JSTAT</td>
<td>A matching job has its usage statistics reported into this template.</td>
</tr>
</tbody>
</table>

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.

```
# limit all users to a total of two non-interactive jobs
USERCFG[DEFAULT]  MAXJOB=2

# reservation configuration
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]

# job template configuration
```
In the preceding example, a reservation called test is created. This reservation is only accessible by interactive jobs and jobs that will take less than one hour to complete. Specifying MAXJOB=2 means that each user on the system is only allowed to run two jobs simultaneously.

The job template configuration is signified by the JOBCFG parameter. The inter.min template (JMIN) determines that if a job is marked as interactive it is eligible to have inter.set template applied to it. The JOBMATCHCFG parameter establishes the relationship between inter.min and inter.set. The inter.set template forces the ignpolicies flag on the job, which allows the job to ignore the MAXJOB policy. Also, the job automatically requests the reservation named test. Any user submitting a job may have as many interactive jobs running as resources permit, and the job runs inside the test reservation, which constrains the jobs to run on nodes atl[16-63]. With this template, users do not need to know to which nodes to send interactive jobs; Moab automatically sends interactive jobs to the nodes specified in the reservation.

### 11.10.2.2 Selecting

Rather than matching a job automatically based on job attributes, a specific template can be directly requested by setting the SELECT attribute to TRUE. Attributes specified in the job template are applied to the job.

```bash
JOBCFG[inter.set] FLAGS=ignpolicies REQRSV=test SELECT=true
```

In the preceding example, instead of the interactive jobs automatically going to the reservation named test (as in the Matching example), users can request the inter.set template for the ignpolicies flag and the requested reservation test to be applied to the job.

Use the following to request the job template:

```bash
msub -l template=inter.set myjob.sh
```

Selecting a specific reservation is often used in creating workflows. See 11.10.7 Creating Workflows with Job Templates for more information.

### 11.10.3 Job Template Extension Attributes

When creating a job template, any attribute acceptable within the WIKI workload query data format can be used. In addition, job templates can use any of the extension attributes in the following table. Note that a checkmark in the Template Type (JMIN, JMAX, JDEF, JSET) row indicates compatibility with the associated attribute.

#### ACCOUNT

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;ACCOUNT&gt;[,&lt;ACCOUNT&gt;]</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template Type:</td>
<td>JMIN JMAX JDEF JSET</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Account credentials associated with job. Used in job template matching.</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>JOBCFG[public] FLAGS=preemptee</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JOBCFG[public.min] ACCOUNT=public_acct</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JOBMATCHCFG[public] JMIN=public.min JSET=public</td>
<td></td>
</tr>
</tbody>
</table>

#### ACTION

<table>
<thead>
<tr>
<th>Format:</th>
<th>HOLD or CANCEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template Type:</td>
<td>JMIN JMAX JDEF JSET</td>
</tr>
</tbody>
</table>
### Description
Describes an action that is done to the job if the set template is applied.

#### Example
```
JOBCFG[test.min] QOS=high
JOBCFG[test.set] ACTION=CANCEL
JOBMATCHCFG[test] JMIN=test.min JSET=test.set
```

### ALLOCSIZE
#### Format:
```
<ALLOCSIZE>,<DEALLOCSIZE>
```
#### Type:
- JMIN
- JMAX
- JDEF
- JSET

#### Description:
Number of application tasks to allocate/deallocate at each allocation adjustment.

#### Example
```
JOBCFG[webdb] ALLOCSIZE=4
```

### ALLOCDELAY
#### Format:
```
[[[DD]:HH]:MM]:SS
```
#### Description:
The amount of time to wait before trying to allocate a new resource. When you have a dynamic service job—ones that grow and shrink—ALLOCDELAY is the minimum amount of time Moab waits before allowing a change to the size of the service job. The size of the job should not change too quickly, so ALLOCDELAY specifies an amount of time that must elapse before another size change can occur.

#### Example
```
JOBCFG[webdb] ALLOCDELAY=1:00
```

### ALLOCSYNCTIME
#### Format:
```
[[[DD]:HH]:MM]:SS
```
#### Description:
Time indicating maximum duration Moab waits before forcing allocation decisions via provisioning or other steps.

- By default, when deallocating resources, Moab does not reassign resources from one service to another until the service/job initially assigned the resources reports successful de-allocation via the workload query response.

#### Example
```
JOBCFG[webdb] ALLOCSYNCTIME=00:04:00
JOBCFG[sirius] ALLOCSYNCTIME=00:00:20
```

Moab forces the deallocation of resources assigned to webdb if the application does not release them within four minutes of a deallocation request. For the sirius application, Moab forces deallocation if not released by the application within twenty seconds of a deallocation request.
### CLASS

**Format:** `<CLASS>[,<CLASS>]...`

**Template Type:**

<table>
<thead>
<tr>
<th>JMIN</th>
<th>JMAX</th>
<th>JDEF</th>
<th>JSET</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description:** Class credentials associated with job. Used in job template matching.

**Example:**

```
JOBCFG[night]  FLAGS=preemptor
JOBCFG[night.min]  CLASS=night_class
JOBCFG[night.max]  JMIN=night.min  JSET=night
```

### CPU LIMIT

**Format:** `[[[DD]:HH]:MM]:SS`

**Template Type:**

<table>
<thead>
<tr>
<th>JMIN</th>
<th>JMAX</th>
<th>JDEF</th>
<th>JSET</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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:**

<table>
<thead>
<tr>
<th>JMIN</th>
<th>JMAX</th>
<th>JDEF</th>
<th>JSET</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description:** Description of the job.

**Example:**

```
JOBCFG[webdb]  DESCRIPTION="Template job"
```

### DPROCS

**Format:** `<INTEGER>`

**Template Type:**

<table>
<thead>
<tr>
<th>JMIN</th>
<th>JMAX</th>
<th>JDEF</th>
<th>JSET</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description:** Number of processors dedicated per task. Default is 1.

**Example:**

```
JOBCFG[job.min]  DPROCS=2
JOBCFG[job.max]  DPROCS=4
```

### EUSER

**Format:** `<STRING>`

**Template Type:**

<table>
<thead>
<tr>
<th>JMIN</th>
<th>JMAX</th>
<th>JDEF</th>
<th>JSET</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---
**Description:** In a job template, **EUSER** is the effective user, meaning that when a job starts it will change the user running the job to the one on the template.

**Example:**

```
JOBCFG[batch]    EUSER=render
```

### EXEC

**Format:** `<STRING>`

**Template Type:**

```
 JMIN  JMAX  JDEF  JSET
```

**Description:** Sets what the job runs, regardless of what the user sets.

**Example:**

```
JOBCFG[setup.pre]    EXEC=nfs/tools/setup.pre.sh
```

### FLAGS

**Format:** `<JOBFLAG>[,<JOBFLAG>]...`

**Template Type:**

```
 JMIN  JMAX  JDEF  JSET
```

**Description:** One or more legal job flag values.

**Example:**

```
JOBCFG[webdb]    FLAGS=NORMSTART
```

### GNAME

**Format:** `<STRING>`

**Template Type:**

```
 JMIN  JMAX  JDEF  JSET
```

**Description:** Group credential associated with job.

**Example:**

```
JOBCFG[webserv]    GNAME=service
```

*For matching the group, see the **GROUP** attribute.*

### GRES

**Format:** `<generic resource>[:<COUNT>]`

**Template Type:**

```
 JMIN  JMAX  JDEF  JSET
```

**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.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>Format:</th>
<th><code>&lt;GROUP&gt;[&lt;GROUP&gt;]</code>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template Type:</td>
<td>JMIN JMAX JDEF JSET</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Group credentials associated with job. Used in job template matching.</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>JOBCFG[webserv] GROUP=service</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For setting the group, see the GNAME attribute.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MEM</th>
<th>Format:</th>
<th><code>&lt;INTEGER&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Template Type:</td>
<td>JMIN JMAX JDEF JSET</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Maximum amount of physical memory per task used by the job. See Requesting Resources for more information.</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>JOBCFG[smalljobs] MEM=25</td>
<td></td>
</tr>
</tbody>
</table>

| NODEACCESSPOLICY | Format: | One of the following: SHARED, SHAREDONLY, SINGLEJOB, SINGLETASK, SINGLEUSER, or UNIQUEUSER |
|                  | Template Type: | JMIN JMAX JDEF JSET |
|                  | Description: | Specifies how node resources will be shared by a job. See the Node Access Overview for more information. |
|                  | Example: | JOBCFG[serverapp] NODEACCESSPOLICY=SINGLEJOB |

<table>
<thead>
<tr>
<th>NODERANGE</th>
<th>Format:</th>
<th><code>&lt;MIN&gt;[&lt;MAX&gt;]</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Template Type:</td>
<td>JMIN JMAX JDEF JSET</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Minimum and maximum nodes allowed to be allocated to job.</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>JOBCFG[vizserver] NODERANGE=1,16</td>
<td></td>
</tr>
</tbody>
</table>

<p>| NODES |</p>
<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| <INTEGER> | Number of nodes required by job. Default is 1. See Node Definition for more information. | JOBCFG[job.min] NODES=2  
JOBCFG[job.max] NODES=4 |

**PARTITION**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;PARTITION&gt;[:&lt;PARTITION&gt;]...</td>
<td>Specifies the partition (or partitions) in which a job must run.</td>
<td>JOBCFG[meis] PARTITION=math:geology</td>
</tr>
</tbody>
</table>

**PREF**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;STRING&gt;</td>
<td>Specifies which node features are preferred by the job and should be allocated if available. See PREF for more information.</td>
<td>JOBCFG[meis] PREF=bigmem</td>
</tr>
</tbody>
</table>

**PRIORITY**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;INTEGER&gt;</td>
<td>Relative job priority.</td>
<td>JOBCFG[meis] PRIORITY=25000</td>
</tr>
</tbody>
</table>

**PROCRANGE**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;MIN&gt;[,&lt;MAX&gt;]</td>
<td>Minimum and maximum processors allowed to be allocated to job.</td>
<td>JOBCFG[meis] PROCRANGE=2,64</td>
</tr>
</tbody>
</table>
### QOS

**Format:** `<QOS>[,<QOS>]...`

**Template Type:**

<table>
<thead>
<tr>
<th>JMIN</th>
<th>JMAX</th>
<th>JDEF</th>
<th>JSET</th>
</tr>
</thead>
</table>

**Description:** QoS credentials associated with job. Used in 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:**

<table>
<thead>
<tr>
<th>JMIN</th>
<th>JMAX</th>
<th>JDEF</th>
<th>JSET</th>
</tr>
</thead>
</table>

**Description:** Architecture required by job.

**Example:**

```
JOBCFG[servapp] RARCH=1386
```

### RFEATURES

**Format:** `<FEATURE>[,<FEATURE>]...`

**Template Type:**

<table>
<thead>
<tr>
<th>JMIN</th>
<th>JMAX</th>
<th>JDEF</th>
<th>JSET</th>
</tr>
</thead>
</table>

**Description:** List of features required by job.

**Example:**

```
JOBCFG[servapp] RFEATURES=fast,bigmem
```

### RM

**Format:** `<STRING>`

**Template Type:**

<table>
<thead>
<tr>
<th>JMIN</th>
<th>JMAX</th>
<th>JDEF</th>
<th>JSET</th>
</tr>
</thead>
</table>

**Description:** Destination resource manager to be associated with job.

**Example:**

```
JOBCFG[webdb] RM=slurm
```

### ROPSYS

**Format:** `<STRING>`

**Template Type:**

<table>
<thead>
<tr>
<th>JMIN</th>
<th>JMAX</th>
<th>JDEF</th>
<th>JSET</th>
</tr>
</thead>
</table>

**Description:** Operating system required by job.
### Example:
```
JOBCFG[test.set] ROPSYS=windows
```

#### SELECT

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;BOOLEAN&gt; : TRUE</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template Type</td>
<td>JMIN JMAX JDEF JSET</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Job template can be directly requested by job at submission.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>JOBCFG[servapp] SELECT=TRUE</td>
<td></td>
</tr>
</tbody>
</table>

#### SOFTWARE

| Format         | <RESTYPE>[(+|:)<COUNT>][@<TIMEFRAME>] |
|----------------|--------------------------------------|
| Template Type  | JMIN JMAX JDEF JSET |
| Description    | Indicates generic resources required by the job. See SOFTWARE for more information. |
| Example        | JOBCFG[servapp] SOFTWARE=matlab:2 |

#### SYSTEMJOBTYPE

<table>
<thead>
<tr>
<th>Format</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Template Type</td>
<td>JMIN JMAX JDEF JSET</td>
</tr>
<tr>
<td>Description</td>
<td>System job type (ex. vmcreate).</td>
</tr>
<tr>
<td>Example</td>
<td>JOBCFG[vmcreate.min] SYSTEMJOBTYPE=vmcreate JOBCFG[vmcreate.set] TRIGGER=atype=reserve,action=&quot;00:05:00&quot;,etype=end JOBMATCHCFG[vmcreate] JMIN=vmcreate.min JSET=vmcreate.set</td>
</tr>
</tbody>
</table>

#### TARGETBACKLOG

<table>
<thead>
<tr>
<th>Format</th>
<th>[&lt;MIN&gt;,]&lt;MAX&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template Type</td>
<td>JMIN JMAX JDEF JSET</td>
</tr>
<tr>
<td>Description</td>
<td>Minimum and maximum backlog for application within job. In the case of dynamic jobs, Moab allocates/deallocates resources as needed to keep the job within the target range.</td>
</tr>
<tr>
<td>Example</td>
<td>JOBCFG[pdb] TARGETBACKLOG=0.5,2.0</td>
</tr>
</tbody>
</table>

#### TARGETLOAD

<table>
<thead>
<tr>
<th>Format</th>
<th>[&lt;MIN&gt;,]&lt;MAX&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template Type</td>
<td>JMIN JMAX JDEF JSET</td>
</tr>
</tbody>
</table>

### TARGETLOAD

**Description:** Minimum and maximum load for application within job. In the case of dynamic jobs, Moab allocates/deallocates resources as needed to keep the job within the target range.

**Example:**
```
JOBCFG[pdb]  TARGETLOAD=0.5,2.0
```

### TARGETRESPONSETIME

**Format:** \([<\text{MIN}>,<\text{MAX}>]\)

**Template Type:** JMIN JMAX JDEF JSET

**Description:** Minimum and maximum response time for application within job. In the case of dynamic jobs, Moab allocates/deallocates resources as needed to keep the job within the target range.

**Example:**
```
JOBCFG[pdb]  TARGETRESPONSETIME=0.5,2.0
```

### TARGETTHROUGHPUT

**Format:** \([<\text{MIN}>,<\text{MAX}>]\)

**Template Type:** JMIN JMAX JDEF JSET

**Description:** Minimum and maximum throughput for application within job. In the case of dynamic jobs, Moab allocates/deallocates resources as needed to keep the job within the target range.

**Example:**
```
JOBCFG[pdb]  TARGETTHROUGHPUT=0.5,2.0
```

### TASKS

**Format:** \(<\text{INTEGER}>\)

**Template Type:** JMIN JMAX JDEF JSET

**Description:** Number of tasks required by job. Default is 1. See Task Definition for more information.

**Example:**
```
JOBCFG[job.min]  TASKS=4
JOBCFG[job.max]  TASKS=8
```

### TASKPERNODE

**Format:** \(<\text{INTEGER}>\)

**Template Type:** JMIN JMAX JDEF JSET

**Description:** Exact number of tasks required per node. Default is 0.

**Example:**
```
JOBCFG[job.min]  TASKPERNODE=2
JOBCFG[job.max]  TASKPERNODE=4
```
<table>
<thead>
<tr>
<th><strong>TEMPLATEDEPEND</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
<td><code>&lt;TYPE&gt;:&lt;TEMPLATE_NAME&gt;</code></td>
</tr>
<tr>
<td><strong>Template Type:</strong></td>
<td>JMIN JMAX JDEF JSET</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
| **Example:** | JOBCFG[engineering] TEMPLATEDEPEND=AFTER:setup.pre
JOBCFG[setup.pre] SELECT=TRUE EXEC=/tools/setup.pre.sh |

<table>
<thead>
<tr>
<th><strong>UNAME</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
<td><code>&lt;STRING&gt;</code></td>
</tr>
<tr>
<td><strong>Template Type:</strong></td>
<td>JMIN JMAX JDEF JSET</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>User credential associated with job.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>JOBCFG[webserv] UNAME=service</td>
</tr>
</tbody>
</table>

For matching the user, see the USER attribute.

<table>
<thead>
<tr>
<th><strong>USER</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
<td><code>&lt;USER&gt;[,&lt;USER&gt;]...</code></td>
</tr>
<tr>
<td><strong>Template Type:</strong></td>
<td>JMIN JMAX JDEF JSET</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>User credentials associated with job.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>JOBCFG[webserv] USER=service</td>
</tr>
</tbody>
</table>

For setting the user, see the UNAME attribute.

<table>
<thead>
<tr>
<th><strong>WCLIMIT</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
<td><code>[[HH:]MM:]SS</code></td>
</tr>
<tr>
<td><strong>Template Type:</strong></td>
<td>JMIN JMAX JDEF JSET</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Walltime required by job. Default is 864000 (100 hours).</td>
</tr>
</tbody>
</table>
| **Example:** | JOBCFG[job.min] WCLIMIT=2:00:00
JOBCFG[job.max] WCLIMIT=12:00:00 |
11.10.4 Resource Manager Templates

Job templates can also be applied to resource managers modifying attributes of jobs submitted to, migrated to, or routed through specified resource manager interfaces. In particular the RMCFG attributes MAX.JOB, MIN.JOB, SET.JOB, and DEFAULT.JOB can be used. However, the meanings of resource manager job templates are slightly different than their JOBMATCHCFG counterparts as described in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX.JOB</td>
<td>A potential job is not allowed to access the resource manager if it has matching attributes set or has resource requests that exceed those specified in this template.</td>
</tr>
<tr>
<td>MIN.JOB</td>
<td>A potential job is not allowed to access the resource manager if it does not have matching attributes set or has resource requests that do not meet or exceed those specified in this template.</td>
</tr>
<tr>
<td>DEFAULT.JOB</td>
<td>A job associated with this resource manager 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.</td>
</tr>
<tr>
<td>SET.JOB</td>
<td>A job associated with this resource manager has the specified attributes forced to these values and these values override any values specified by the submittor at job submission time.</td>
</tr>
</tbody>
</table>

11.10.5 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:

11.10.5.1 Example 1: Setting Up Application-Specific Environments

```
JOBCFG[xxx] EXEC=*app* JOBPROLOG=/usr/local/appprolog.x
```

11.10.5.2 Example 2: Applying Job Preferences and Defaults

```
JOBCFG[xxx] CLASS=appq EXEC=*app* PREF=clearspeed
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT] PRIOR=5.0*PREF
```

11.10.5.3 Example 3: Applying Resource Constraints to Fuzzy Collections

In the following example, a job template match is set up. Using the JOBMATCHCFG 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.
11.10.5.4 Example 4: 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.

```
# 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  REQRSV=interactive

JOBMATCHCFG[interactive]  JMIN=inter.min  JSET=inter.set
```

11.10.6 Managing Job Templates

11.10.6.1 Dynamically Creating Job Templates

Job templates can be dynamically created while Moab is running using the mschedctl -m command.

Dynamically created job templates will only survive a Moab restart if the --flags=pers[istent] flag is used.

```
> mschedctl -m --flags=pers "JOBCFG[preempt]  FLAGS=PREEMPTOR"
> mschedctl -m --flags=pers "JOBCFG[preempt] SELECT=TRUE"
> msub -l template=preempt mysjob.sh
```

11.10.6.2 Removing Job Templates

Job templates cannot be dynamically removed. You can only remove job templates by commenting out (or removing) the template definition (lines) in the moab.cfg file; then you must restart Moab.

11.10.6.3 Dynamically Modifying Job Templates

Job templates can be dynamically modified by using the mschedctl -m command.

Dynamic changes do not persist after a shutdown of the Moab server unless the --flags=pers[istent] flag is used.

```
> mschedctl -m "JOBCFG[single] NODEACCESSPOLICY=SINGLEJOB"
```

11.10.6.4 Viewing Job Templates

Job templates can be viewed by specifying "template:" before the template name in checkjob and also by using the command mdiag -j as in the following example:

```
> mdiag -j --flags=policy
> checkjob -v template:match.min
```
See Also

- Moab Workload Manager for Grids
- Job Dependencies
12.0 General Node Administration

- 12.1 Node Location (Partitions, Frames, Queues, etc.)
- 12.2 Node Attributes (Node Features, Speed, etc.)
- 12.3 Node-Specific Policies (MaxJobPerNode, etc.)
- 12.4 Managing Shared Cluster Resources
- 12.5 Node State Management
- 12.6 Managing Consumable Generic Resources
- 12.7 Enabling Generic Metrics
- 12.8 Enabling Generic Events

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. To accommodate all systems, Moab determines a node's configuration through the following approaches:

Resource Manager Direct Specification

Some node attributes may be directly specified through the resource manager. For example, Loadleveler allows a site to assign a MachineSpeed value to each node. If the site chooses to specify this value within the Loadleveler configuration, Moab obtains this information via the Loadleveler scheduling API and uses it in scheduling decisions. The list of node attributes supported in this manner varies from resource manager to resource manager and should be determined by consulting resource manager documentation.

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.
12.1 Node Location

Nodes can be assigned three types of location information based on partitions, racks, and queues.

- **12.1.1 Partitions**
- **12.1.2 Racks**
- **12.1.3 Queues**
  - **12.1.3.1 TORQUE/OpenPBS Queue to Node Mapping**
- **12.1.4 Node Selection/Specification**

### 12.1.1 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 a 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.

### 12.1.2 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 64. 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/Laodeleveler 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 and their use is simply for convenience purposes in displaying and analyzing compute resources.

Example:

```
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 64, and racks must be in the range of 1 to 400.

### 12.1.3 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.

#### 12.1.3.1 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 host17.

> 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.

12.1.4 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=c137.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 1: can be used as in the following example:

> setres 1: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

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 SRCFG'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 **OBJECTLIST** parameter to one of the following options: exact, range, regex, rangere, or rerange.
## 12.2 Node Attributes

- **12.2.1 Configurable Node Attributes**
- **12.2.2 Node Features/Node Properties**

### 12.2.1 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS</td>
<td>Specifies the node access policy that can be one of <strong>SHARED, SHAREDONLY, SINGLEJOB, SINGLETASK, or SINGLEUSER.</strong> See Node Access Policies for more details.</td>
</tr>
<tr>
<td>ARCH</td>
<td>Specifies the node's processor architecture.</td>
</tr>
<tr>
<td>CHARGERATE</td>
<td>Allows a site to assign specific charging rates to the usage of particular resources. The <strong>CHARGERATE</strong> value may be specified as a floating point value and is integrated into a job's total charge (as documented in the Charging and Allocation Management section).</td>
</tr>
<tr>
<td>COMMENT</td>
<td>Allows an organization to annotate a node via the configuration file to indicate special information regarding this node to both users and administrators. The</td>
</tr>
<tr>
<td>MAXPROC</td>
<td></td>
</tr>
<tr>
<td>MAXPROCPERCLASS</td>
<td></td>
</tr>
<tr>
<td>MAXPE</td>
<td></td>
</tr>
<tr>
<td>MAXPROC</td>
<td></td>
</tr>
<tr>
<td>PRIORITY</td>
<td></td>
</tr>
<tr>
<td>PRIORITYF</td>
<td></td>
</tr>
<tr>
<td>POWERPOLICY</td>
<td></td>
</tr>
<tr>
<td>PREEMPTMAXCPULOAD</td>
<td></td>
</tr>
<tr>
<td>PREEMPTMINMEMAVAIL</td>
<td></td>
</tr>
<tr>
<td>PREEMPTPOLICY</td>
<td></td>
</tr>
<tr>
<td>PREVRM</td>
<td></td>
</tr>
<tr>
<td>PROCSPEED</td>
<td></td>
</tr>
<tr>
<td>PROCSCORE</td>
<td></td>
</tr>
<tr>
<td>RACK</td>
<td></td>
</tr>
<tr>
<td>RADISK</td>
<td></td>
</tr>
<tr>
<td>RCDISK</td>
<td></td>
</tr>
<tr>
<td>RCPROCMEM</td>
<td></td>
</tr>
<tr>
<td>RCSDISK</td>
<td></td>
</tr>
<tr>
<td>RCSDRIVE</td>
<td></td>
</tr>
<tr>
<td>RCMEM</td>
<td></td>
</tr>
<tr>
<td>RCSWAP</td>
<td></td>
</tr>
<tr>
<td>SIZE</td>
<td></td>
</tr>
<tr>
<td>SLOT</td>
<td></td>
</tr>
<tr>
<td>SPEED</td>
<td></td>
</tr>
<tr>
<td>SPEEDPERCPUS</td>
<td></td>
</tr>
<tr>
<td>TRIGGER</td>
<td></td>
</tr>
<tr>
<td>VARIABLE</td>
<td></td>
</tr>
<tr>
<td>NODECFG[node013] ACCESS=singlejob</td>
<td></td>
</tr>
<tr>
<td>NODECFG[node013] ARCH=opteron</td>
<td></td>
</tr>
<tr>
<td>NODECFG[DEFAULT] CHARGERATE=1.0</td>
<td></td>
</tr>
<tr>
<td>NODECFG[node003] CHARGERATE=1.5</td>
<td></td>
</tr>
<tr>
<td>NODECFG[node022] CHARGERATE=2.5</td>
<td></td>
</tr>
</tbody>
</table>
**COMMENT** value may be specified as a **quote** delimited string as shown in the example that follows. Comment information is visible using `checknode`, `mdiaq`, Moab Cluster Manager, and Moab Access Portal.

<table>
<thead>
<tr>
<th>NODECFG[node013]</th>
<th>COMMENT=&quot;Login Node&quot;</th>
</tr>
</thead>
</table>

**ENABLEPROFILING**

Allows an organization to track node state over time. This information is available using `showstats -n`.

<table>
<thead>
<tr>
<th>NODECFG[DEFAULT]</th>
<th>ENABLEPROFILING=TRUE</th>
</tr>
</thead>
</table>

**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 set/overwrite a node's features, use `FEATURES=<X>`; to append node features, use `FEATURES+=<X>`.

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.

| NODECFG[node013] | FEATURES+=gpfs,fastio |

**FLAGS**

Specifies various attributes of the **NODECFG** parameter.

The NoVMMigrations flag excludes VMs from migrations.

| NODECFG[node1] | FLAGS=NoVMMigrations |

**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 |

**LOGLEVEL**

Node specific loglevel allowing targetted log facility verbosity.

**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
...
```

**MAXPROCPERCLASS**
Maximum dedicated processors allowed per class on this node. No jobs are scheduled on this node when this number is reached. See Node Policies for more information.

```
NODECFG[node024] MAXPROCPERCLASS=2
...
```

**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
...
```

**NODETYPE**
The `NODETYPE` attribute is most commonly used in conjunction with an allocation management system such as Gold. In these cases, each node is assigned a node type and within the allocation management system, each node type is assigned a charge rate. For example, a site administrator may want to charge users more for using large memory nodes and may assign a node type of `BIGMEM` to these nodes. The allocation management system would then charge a premium rate for jobs using `BIGMEM` nodes. (See the Allocation Manager Overview for more information.)

Node types are specified as simple strings. If no node type is explicitly set, the node will possess the default node type of `[DEFAULT]`. Node type information can be specified directly using `NODECFG` or through use of the `FEATURENODETYPEHEADER` parameter.

```
NODECFG[node024] NODETYPE=BIGMEM
```

**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` instead of `OS` if you are using TORQUE.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OSLIST</strong></td>
<td>This attribute specifies the list of operating systems the node can run.</td>
<td><code>NODECFG[compute002] OSLIST=linux,windows</code></td>
</tr>
<tr>
<td><strong>OVERCOMMIT</strong></td>
<td>Allows overcommitting of specified attributes of a node. Possible attributes include, DISK, MEM, PROCS, and SWAP. Usage is <code>&lt;attr&gt;:&lt;integer&gt;</code>.</td>
<td><code>NODECFG[node012] OVERCOMMIT=PROCS:2 MEM:4</code></td>
</tr>
<tr>
<td><strong>PARTITION</strong></td>
<td>See Node Location for details.</td>
<td></td>
</tr>
<tr>
<td><strong>POOL</strong></td>
<td>Specifies the associated node pool.</td>
<td></td>
</tr>
<tr>
<td><strong>POWERPOLICY</strong></td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td><strong>PREEMPTMAXCPULOAD</strong></td>
<td>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 <code>PREEMPTPOLICY</code> attribute. If this attribute is not specified, the global default policy specified with <code>PREEMPTPOLICY</code> parameter is used. See Sharing Server Resources for further details.</td>
<td><code>NODECFG[node024] PRIORITY=-150 COMMENT=&quot;NFS Server Node&quot; NODECFG[node024] PREEMPTPOLICY=CANCEL PREEMPTMAXCPULOAD=1.2 ...</code></td>
</tr>
<tr>
<td><strong>PREEMPTMINMEMAVAIL</strong></td>
<td>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 <code>PREEMPTPOLICY</code> attribute. If this attribute is not specified, the global default policy specified with <code>PREEMPTPOLICY</code> parameter is used. See Sharing Server Resources for further details.</td>
<td><code>NODECFG[node024] PRIORITY=-150 COMMENT=&quot;NFS Server Node&quot; NODECFG[node024] PREEMPTPOLICY=CANCEL PREEMPTMINMEMAVAIL=1.2 ...</code></td>
</tr>
<tr>
<td><strong>PREEMPTPOLICY</strong></td>
<td>If any node preemption policies are triggered (such as <code>PREEMPTMAXCPULOAD</code> or <code>PREEMPTMINMEMAVAIL</code>) 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 <code>PREEMPTPOLICY</code> parameter is used. See Sharing Server Resources for further details.</td>
<td><code>NODECFG[node024] PRIORITY=-150 COMMENT=&quot;NFS Server Node&quot; NODECFG[node024] PREEMPTPOLICY=CANCEL</code></td>
</tr>
</tbody>
</table>
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.

```moab
NODEALLOCATIONPOLICY  PRIORITY
NODECFG[node024]  PRIORITY=120
```

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:

```moab
<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.

```moab
NODEALLOCATIONPOLICY  PRIORITY
NODECFG[node024]  PRIORITYF='SPEED + .01 * AMEM - 10 * JOBCOUNT'
```

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.

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.

```moab
RMCFG[prov]  TYPE=NATIVE RESOURCETYPE=PROV
RMCFG[prov]  PROVDURATION=10:00
RMCFG[prov]  NODEMODIFYURL=exec://$HOME/tools/os.switch.pl
...  
NODECFG[node024]  PROVRM=prov
```
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RACK</strong></td>
<td>The rack associated with the node's physical location. Valid values range from 1 to 400. See Node Location for details.</td>
</tr>
<tr>
<td><strong>RADISK</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>RCDISK</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>RCMEM</strong></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>NODECFG[node024] RCMEM=2048</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>If the resource manager does not report any memory, then Moab will assign node024 2048 MB of memory.</td>
</tr>
<tr>
<td></td>
<td>NODECFG[node024] RCMEM=2048^</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>Moab will assign 2048 MB of memory to node024 regardless of what the resource manager reports.</td>
</tr>
<tr>
<td><strong>RCPROC</strong></td>
<td>The RCPROC specifies the number of processors available on a compute node.</td>
</tr>
<tr>
<td></td>
<td>NODECFG[node024] RCPROC=8</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td><strong>RCSWAP</strong></td>
<td>Jobs can request a certain amount of swap space in MB. RCSWP 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.</td>
</tr>
<tr>
<td></td>
<td>NODECFG[node024] RCSWAP=2048</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
If the resource manager does not report any memory, Moab will assign node024 2048 MB of swap.

```bash
NODECFG[node024] RCSWAP=2048
...
```

Moab will assign 2048 MB of swap to node024 regardless of what the resource manager reports.

<table>
<thead>
<tr>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

```bash
NODECFG[node024] SIZE=2
...
```

<table>
<thead>
<tr>
<th>SLOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>The first slot in the rack associated with the node's physical location. Valid values range from 1 to <code>MMAX_RACKSIZE</code> (default=64). See Node Location for details.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPEED</th>
</tr>
</thead>
<tbody>
<tr>
<td>A node's speed is very similar to its processor speed but is specified as a relative value. In general use, the speed of a base node is determined and assigned a speed of 1.0. A node that is 50% faster would be assigned a value of 1.5 while a slower node may receive a value that is proportionally less than 1.0. Node speeds do not have to be directly proportional to processor speeds and may take into account factors such as memory size or networking interface. Generally, node speed information is used to determine proper wallclock limit and CPU time scaling adjustments. Node speed information is specified as a unitless floating point ratio and can be specified through the resource manager or with the <code>NODECFG</code> parameter.</td>
</tr>
</tbody>
</table>

- The `SPEED` specification must be in the range of 0.01 to 100.0.

<table>
<thead>
<tr>
<th>TRIGGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>See Object Triggers for details</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VARIABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables associated with the given node, which can be used in job scheduling. See <code>-l PREF</code>.</td>
</tr>
</tbody>
</table>

```bash
NODECFG[node024] VARIABLE=var1
...
```

12.2.2 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. As a convenience feature, certain node attributes can be specified via node features using the parameters listed in the following table:
<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEATURENODETYPEHEADER</td>
<td>Set Node Type</td>
</tr>
<tr>
<td>FEATUREPARTITIONHEADER</td>
<td>Set Partition</td>
</tr>
<tr>
<td>FEATUREPROCSPEEDHEADER</td>
<td>Set Processor Speed</td>
</tr>
<tr>
<td>FEATURERACKHEADER</td>
<td>Set Rack</td>
</tr>
<tr>
<td>FEATURESLOTHEADER</td>
<td>Set Slot</td>
</tr>
</tbody>
</table>

Example

```
FEATUREPARTITIONHEADER par
FEATUREPROCSPEEDHEADER cpu
```

See Also

- Job Preferences
- Configuring Node Features 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
12.3 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] ...

12.3.1 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.

If node pools are specified via the node's POOL attribute, the MAXJOB limit is applied across the aggregate pool as in the following example:

<table>
<thead>
<tr>
<th>NODECFG[node01-a]</th>
<th>POOL=node01</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODECFG[node01-b]</td>
<td>POOL=node01</td>
</tr>
<tr>
<td>NODECFG[node01-c]</td>
<td>POOL=node01</td>
</tr>
<tr>
<td>NODECFG[node01-d]</td>
<td>POOL=node01</td>
</tr>
<tr>
<td>NODECFG[node02-a]</td>
<td>POOL=node02</td>
</tr>
<tr>
<td>NODECFG[node02-b]</td>
<td>POOL=node02</td>
</tr>
<tr>
<td>NODECFG[node02-c]</td>
<td>POOL=node02</td>
</tr>
<tr>
<td>NODECFG[node02-d]</td>
<td>POOL=node02</td>
</tr>
</tbody>
</table>

# only allow one job to run per node pool
NODECFG[DEFAULT] MAXJOB=1

MAXJOBPERUSER

This policy 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

This policy 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

This policy constrains the CPU load the node will support as opposed to the number of jobs. If the node's load exceeds the MAXLOAD limit and the NODELOADPOLICY parameter is set to ADJUSTSTATE, the node is marked busy. 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:

```
NODECFG[node024] MAXJOB=4 MAXJOBPERUSER=2
NODECFG[node025] MAXJOB=2
NODECFG[node026] MAXJOBPERUSER=1
NODECFG[DEFAULT] MAXLOAD=2.5
...
```

12.3.2 Desktop Management Policies (Desktop Harvesting)

KBDDETECTPOLICY, MINRESUMEKBDIDLETIME, and MINPREEMPTLOAD

Sometimes a site administrator would like to add normal desktops to their cluster but are afraid that jobs might overrun normal desktop usage. Using these three parameters desktop support can be enabled and configured based on a particular site administrator's needs. KBDDETECTPOLICY tells the scheduler what to do when local keyboard usage is detected. When it is set to "DRAIN" any jobs currently running on the node are allowed to finish but no new jobs will run on the node until MINRESUMEKBDIDLETIME is reached. When KBDDETECTPOLICY is set to "PREEMPT" any jobs running on the system are preempted and the full system reverts to desktop usage until no keyboard activity is detected for MINRESUMEKBDIDLETIME.

Desktop support can be further configured using MINPREEMPTLOAD. This parameter tells Moab to continue scheduling jobs on a node even when local keyboard usage is detected as long as the total system load remains below the specified value. These three parameters can be configured globally or for each particular node.

Example:

```
NODECFG[node024] KBDDETECTPOLICY=DRAIN  # when local keyboard is active jobs will be allowed to finish
NODECFG[node024] MINRESUMEKBDIDLETIME=600 # no jobs will be scheduled on node until keyboard has been idle for 10 minutes
NODECFG[node024] MINPREEMPTLOAD=0.5    # as long as system load remains below 0.5 jobs will be scheduled, # even if keyboard is active
...
```

OR:

```
NODECFG[DEFAULT] KBDDETECTPOLICY=PREEMPT  # any jobs running on any nodes will be preempted if local keyboard usage is detected
NODECFG[DEFAULT] MINRESUMEKBDIDLETIME=600 # no jobs will be scheduled on any node until local keyboard has been idle for 10 minutes
NODECFG[DEFAULT] MINPREEMPTLOAD=0.5      # as long as the local system load remains below 0.5, jobs will be scheduled on the node, even if keyboard is active
...
```
12.3.3 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

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
```

See Also

- mnodectl
12.4 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.

12.4.1 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
```

If defining a pseudo-node other than GLOBAL, the node name will need to be added to the RESOURCETEMP list.

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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Pseudo-Node</th>
<th>Generic Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node-Locked</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Resources can be encapsulated as an independent node.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requires exclusive batch system control over resource</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Resources (such as file systems and licenses) may be consumed both inside and outside of batch system workload.</td>
<td></td>
<td>Resources must only be consumed by batch workload. Use outside of batch control results in loss of resource synchronization.</td>
</tr>
<tr>
<td>Allows scheduler level allocation of resources</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>If required, the scheduler can take external administrative action to allocate the resource to the job.</td>
<td></td>
<td>The scheduler can only maintain logical allocation information and cannot take any external action to allocate resources to the job.</td>
</tr>
</tbody>
</table>

12.4.2 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.

12.4.2.1 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.

### 12.4.3 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
```

### 12.4.4 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**

Configure Moab to support four floating **quickcalc** and two floating **matlab** licenses.

```
NODECFG[GLOBAL]  GRES=quickcalc:4,matlab:2
... 
```

Submit a **TORQUE** job requesting a node-locked or floating **quickcalc** license.

```
> qsub -l nodes=1,software=quickcalc,walltime=72000 testjob.cmd
```

### See Also

- Managing Resources Directly with the Native Interface
- License Management
12.5 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

12.5.1 Node State Definitions

<table>
<thead>
<tr>
<th>State</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Down</td>
<td>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.</td>
</tr>
<tr>
<td>Idle</td>
<td>Node is reporting status, currently is not executing any workload, and is ready to accept additional workload.</td>
</tr>
<tr>
<td>Busy</td>
<td>Node is reporting status, currently is executing workload, and cannot accept additional workload due to load.</td>
</tr>
<tr>
<td>Running</td>
<td>Node is reporting status, currently is executing workload, and can accept additional workload.</td>
</tr>
<tr>
<td>Drained</td>
<td>Node is reporting status, currently is not executing workload, and cannot accept additional workload due to administrative action.</td>
</tr>
<tr>
<td>Draining</td>
<td>Node is reporting status, currently is executing workload, and cannot accept additional workload due to administrative action.</td>
</tr>
</tbody>
</table>

12.5.2 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.

12.5.3 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).

12.5.4 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.

12.5.4.1 Down State

- JOBACTIIONONNODEFAILURE parameter (cancel/requeue jobs if allocated nodes fail).
- Triggers (take specified action if failure is detected).

See Also

- Managing Resources Directly with the Native Interface
- License Management
- Adjusting Node Availability
- NODEMAXLOAD parameter
12.6 Managing Consumable Generic Resources

- 12.6.1 Configuring Node-Locked Consumable Generic Resources
  - 12.6.1.1 Requesting Consumable Generic Resources
- 12.6.2 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.

12.6.1 Configuring Node-locked Consumable Generic Resources

Consumable generic resources are supported within Moab using either direct configuration or resource manager auto-detect. 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
...

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
```

*By default, Moab supports up to 128 independent generic resource types.*

12.6.1.1 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 1: 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
```
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 2: Per Job Requests

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 Example 1: Per Task Requests, 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:

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:

For each node requested by a viz job, also request two graphics cards.

12.6.2 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.

**See Also**

- Consumable Resource Handling
- 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
12.7 Enabling Generic Metrics

- 12.7.1 Configuring Generic Metrics
- 12.7.2 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

12.7.1 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 outs even 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
12.7.2 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:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigBrother</td>
<td><a href="http://www.bb4.org">http://www.bb4.org</a></td>
</tr>
<tr>
<td>Ganglia</td>
<td><a href="http://ganglia.sourceforge.net">http://ganglia.sourceforge.net</a></td>
</tr>
<tr>
<td>Monit</td>
<td><a href="http://www.tildeslash.com/monit">http://www.tildeslash.com/monit</a></td>
</tr>
<tr>
<td>Nagios</td>
<td><a href="http://www.nagios.org">http://www.nagios.org</a></td>
</tr>
</tbody>
</table>

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:

```perl
#!/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:

```plaintext
RMCFG[local] TYPE=NATIVE
CLUSTERQUERYURL=exec://$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:

```plaintext
> 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='node $OID temp high'
...
```

See Also

- Simulation Overview
- Generic Consumable Resources
- Object Variables
- Generic Event Counters
12.8 Enabling Generic Events

- 12.8.1 Configuring Generic Events
  - 12.8.1.1 Action Types
  - 12.8.1.2 Named Events
  - 12.8.1.3 Generic Metric (GMetric) Events
- 12.8.2 Reporting Generic Events
- 12.8.3 Generic Events Attributes
- 12.8.4 Manually Creating Generic 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

12.8.1 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 are listed in the following table:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION</td>
<td>Comma-delimited list of actions to be processed when a new event is received.</td>
</tr>
<tr>
<td>ECOUNT</td>
<td>Number of events that must occur before launching action.</td>
</tr>
<tr>
<td>REARM</td>
<td>Minimum time between events specified in [[[DD:]HH:]MM:SS] format.</td>
</tr>
<tr>
<td>SEVERITY</td>
<td>An arbitrary severity level from 1 through 4, inclusive. SEVERITY appears in the output of <code>mdia</code>g <code>n v v --xml</code>.</td>
</tr>
</tbody>
</table>

12.8.1.1 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE[:&lt;OTYPE&gt;:&lt;OID&gt;]</td>
<td>Marks event object (or specified object) down until event report is cleared.</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>Executes a script at the provided path. Arguments are allowed at the end of the path and are separated by question marks (?). Trigger variables (such as $OID) are allowed.</td>
</tr>
<tr>
<td>NOTIFY</td>
<td>Notifies administrators of the event occurrence.</td>
</tr>
<tr>
<td>OFF</td>
<td>Powers off node or resource.</td>
</tr>
<tr>
<td>ON</td>
<td>Powers on node or resource.</td>
</tr>
<tr>
<td>PREEMPT[:&lt;POLICY&gt;]</td>
<td>Preempts workload associated with object (valid for node, job, reservation, partition, resource manager, user, group, account, class, QoS, and cluster objects).</td>
</tr>
<tr>
<td>RECORD</td>
<td>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.</td>
</tr>
<tr>
<td>RESERVE[:&lt;DURATION&gt;]</td>
<td>Reserves node for specified duration (default: 24 hours).</td>
</tr>
<tr>
<td>RESET</td>
<td>Resets object (valid for nodes - causes reboot).</td>
</tr>
<tr>
<td>SIGNAL[:&lt;SIGNO&gt;]</td>
<td>Sends signal to associated jobs or services (valid for node, job, reservation, partition, resource manager, user, group, account, class, QoS, and cluster objects).</td>
</tr>
</tbody>
</table>

### 12.8.1.2 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**

```bash
# 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
```

### 12.8.1.3 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:

```bash
GEVENTCFG[GMETRIC<COMPARISON>VALUE] ACTION=...
```
Example

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 shown in the following table:

<table>
<thead>
<tr>
<th>Type</th>
<th>Comparison</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>Numeric values only</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>Numeric values only</td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
<td>Numeric values only</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>Numeric values only</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>Numeric values only</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>not equal</td>
<td>Numeric values only</td>
</tr>
</tbody>
</table>

### 12.8.2 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.

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 messages specified after GEVENT are routed to Moab Cluster Manager for graphical display and can be used to dynamically adjust scheduling behavior.

### 12.8.3 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.

### 12.8.4 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:

```bash
> mnodectl -m gevent=event:"hello" node02
```

To add GEVENT "event" with message "hello" to myvm, do the following:

```bash
> mvmctl -m gevent=event:"hello" myvm
```

**See Also**

- Simulation Overview
- Generic Consumable Resources
- Object Variables
- Generic Event Counters
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:

- **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.)
13.1 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.

13.1.1 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.

13.1.1.1 Resource Manager Commands

For many environments, Moab interaction with the resource manager is limited to the following objects and functions:

<table>
<thead>
<tr>
<th>Object</th>
<th>Function</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job</td>
<td>Query</td>
<td>Collect detailed state, requirement, and utilization information about jobs</td>
</tr>
<tr>
<td>Job</td>
<td>Modify</td>
<td>Change job state and/or attributes</td>
</tr>
<tr>
<td>Job</td>
<td>Start</td>
<td>Execute a job on a specified set of resource</td>
</tr>
<tr>
<td>Job</td>
<td>Cancel</td>
<td>Cancel an existing job</td>
</tr>
<tr>
<td>Job</td>
<td>Preempt/Resume</td>
<td>Suspend, resume, checkpoint, restart, or requeue a job</td>
</tr>
<tr>
<td>Node</td>
<td>Query</td>
<td>Collect detailed state, configuration, and utilization information about compute resources</td>
</tr>
<tr>
<td>Node</td>
<td>Modify</td>
<td>Change node state and/or attributes</td>
</tr>
<tr>
<td>Queue</td>
<td>Query</td>
<td>Collect detailed policy and configuration information from the resource manager</td>
</tr>
</tbody>
</table>

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 dynamic job support, provisioning, and cluster level resource management.

13.1.1.2 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.

### 13.1.2 Resource Manager Specific Details (Limitations/Special Features)

- LoadLeveler
  - [Loadleveler Integration Guide](#)
- TORQUE/OpenPBS
  - [TORQUE Homepage](#)
  - [PBS Integration Guide](#)
- PBSPro
  - [PBS Integration Guide](#)
- SGE 6.0+
  - [SGE Integration Guide](#)
- SLURM/Wiki
  - [SLURM Integration Guide](#)
  - [Wiki Overview](#)
- LSF
  - [LSF Integration Guide](#)

### 13.1.3 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)
- Job State Synchronization Policies (see JOBSYNCTIME)
- Stale Data Purging (see JOBPURGETIME)
- Thread Management (preventing resource manager failures from affecting scheduler operation)
- Resource Manager Poll Interval (see RMPOLLINTERVAL)
- Node Query Refresh Rate (see NODEPOLLFREQUENCY)

### 13.1.4 Evaluating Resource Manager Availability and Performance

Each resource manager is individually tracked and evaluated by Moab. Using the `mdiaq -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.

**See Also**

- Resource Manager Configuration
- Resource Manager Extensions
13.2 Resource Manager Configuration

- 13.2.1 Defining and Configuring Resource Manager Interfaces
  - 13.2.1.1 Resource Manager Attributes
- 13.2.2 Resource Manager Configuration Details
  - 13.2.2.1 Resource Manager Types
  - 13.2.2.2 Resource Manager Name
  - 13.2.2.3 Resource Manager Location
  - 13.2.2.4 Resource Manager Flags
  - 13.2.2.5 Other Attributes
- 13.2.3 Scheduler/Resource Manager Interactions

13.2.1 Defining and Configuring Resource Manager Interfaces

Moab resource manager interfaces are defined using the RMCFG 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.

13.2.1.1 Resource Manager Attributes

The following table lists the possible resource manager attributes that can be configured.

<table>
<thead>
<tr>
<th>ADMINEXEC</th>
<th>JOBRSVRECREATE</th>
<th>SLURMFLAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTHTYPE</td>
<td>JOBSTARTURL</td>
<td>SOFTTERMSIG</td>
</tr>
<tr>
<td>BANDWIDTH</td>
<td>JOBSUBMITURL</td>
<td>STAGETHRESHOLD</td>
</tr>
<tr>
<td>CHECKPOINTSIG</td>
<td>JOBSUSPENDURL</td>
<td>STARTCMD</td>
</tr>
<tr>
<td>CHECKPOINTTIMEOUT</td>
<td>JOBVOLUMEURL</td>
<td>SUBMITCMD</td>
</tr>
<tr>
<td>CLIENT</td>
<td>MAXDSOP</td>
<td>SUBMITPOLICY</td>
</tr>
<tr>
<td>CLUSTERQUERYURL</td>
<td>MAXJOB</td>
<td>SUSPENDSIG</td>
</tr>
<tr>
<td>CONFIGFILE</td>
<td>MAXJOBPERMINUTE</td>
<td>SYNCJOBID</td>
</tr>
<tr>
<td>DATARM</td>
<td>MAXJOBS</td>
<td>SYSTEMMODIFYURL</td>
</tr>
<tr>
<td>DEFAULTCLASS</td>
<td>MINETIME</td>
<td>SYSTEMQUERYURL</td>
</tr>
<tr>
<td>DEFAULTJOB</td>
<td>MINJOB</td>
<td>TARGETUSAGE</td>
</tr>
<tr>
<td>DEFAULTTHIGHSPEEDADAPTER</td>
<td></td>
<td>TIMEOUT</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>NIMPORT</td>
<td>TRANSLATIONSCRIPT</td>
</tr>
<tr>
<td>ENV</td>
<td>NODEDEFAULTERSVPOLICY</td>
<td>TRIGGER</td>
</tr>
<tr>
<td>EPORT</td>
<td>NODESTATEPOLICY</td>
<td>TYPE</td>
</tr>
<tr>
<td>FAILTIME</td>
<td>OMAP</td>
<td>UCALLOCDURATION</td>
</tr>
<tr>
<td>FLAGS</td>
<td>POLLINTERVAL</td>
<td>UCALLOCOSIZE</td>
</tr>
<tr>
<td>FLOWINTERVAL</td>
<td>POLLTIMEISRIGID</td>
<td>UCMAXSIZE</td>
</tr>
<tr>
<td>FLOWLIMIT</td>
<td>PORT</td>
<td>UCTHRESHOLD</td>
</tr>
<tr>
<td>FLOWMETRIC</td>
<td>PYSTRING</td>
<td>UCTHRESHOLDDURATION</td>
</tr>
<tr>
<td>FNLIST</td>
<td>RESOURCECREATEURL</td>
<td>USEVNODES</td>
</tr>
<tr>
<td>HOST</td>
<td>RESOURCETYPE</td>
<td>VARIABLES</td>
</tr>
<tr>
<td>IGNHNODES</td>
<td>RMSTARTURL</td>
<td>VERSION</td>
</tr>
<tr>
<td>JOBCANCELURL</td>
<td>RMSTOPURL</td>
<td>VMOWNERRM</td>
</tr>
<tr>
<td>JOBEXTENDDURATION</td>
<td>SBINDIR</td>
<td>WORKLOADQUERYURL</td>
</tr>
<tr>
<td>JOBMODIFYURL</td>
<td>SERVER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SETJOB</td>
<td></td>
</tr>
</tbody>
</table>

**Format:** "jobsubmit"
### Default:
NONE

### 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:
```plaintext
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.

- **CHECKSUM**
  - Only valid with WIKI based interfaces.

### Example:
```plaintext
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).

### Example:
```plaintext
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>**

**Default:** ---

**Description:** Specifies what signal to send the resource manager when a job is checkpointed. (See Checkpoint Overview.)

- **SIGKILL**
  - Moab routes the signal SIGHUP through the resource manager to the job when a job is checkpointed.

### CHECKPOINTTIMEOUT

- **RMCFG[base]**
  - **ADMINEXEC=jobsubmit**
  - **AUTHTYPE=CHECKSUM**
  - **BANDWIDTH=340G**
  - **CHECKPOINTSIG=SIGKILL**
<table>
<thead>
<tr>
<th><strong>Format:</strong></th>
<th>[[[DD:]HH:]MM:]SS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default:</strong></td>
<td>0 (no timeout)</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>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.)</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>RMCFG[base] CHECKPOINTTIMEOUT=5:00</td>
</tr>
</tbody>
</table>

Moab cancels any job that has not exited 5 minutes after receiving a checkpoint request.

---

<table>
<thead>
<tr>
<th><strong>CLIENT</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
</tbody>
</table>

Moab will look up and use information for peer clusterB when authenticating the clusterBI resource manager.

---

<table>
<thead>
<tr>
<th><strong>CLUSTERQUERYURL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
</tbody>
</table>

Moab reads /tmp/cluster.config when it queries base resource manager.

---

<table>
<thead>
<tr>
<th><strong>CONFIGFILE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
</tbody>
</table>

The scheduler uses the specified file when establishing the resource manager/scheduler
### DATARM

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;RM NAME&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>If specified, the resource manager uses the given storage resource manager to handle staging data in and out.</td>
</tr>
<tr>
<td>Example</td>
<td><code>RMCFG[clusterB] DATARM=clusterB_storage</code></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;STRING&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>Specifies the class to use if jobs submitted via this resource manager interface do not have an associated class.</td>
</tr>
<tr>
<td>Example</td>
<td><code>RMCFG[internal] DEFAULTCLASS=batch</code></td>
</tr>
</tbody>
</table>

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.

### DEFAULT.JOB

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;STRING&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>Specifies the job template to use to set various job attributes that are not specified by the submitter.</td>
</tr>
<tr>
<td>Example</td>
<td><code>RMCFG[base] DEFAULT.JOB=defjob</code></td>
</tr>
</tbody>
</table>

Moab uses the defjob job template to identify and apply job attribute defaults.

### DEFAULTHIGHSPEEDADAPTER

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;STRING&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>sn0</td>
</tr>
<tr>
<td>Description</td>
<td>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).</td>
</tr>
<tr>
<td>Example</td>
<td><code>RMCFG[base] DEFAULTHIGHSPEEDADAPTER=sn1</code></td>
</tr>
</tbody>
</table>

The scheduler will start jobs requesting a high speed adapter on sn1.
### DESCRIPTION

**Format:** `<STRING>`

**Default:** N/A

**Description:** Specifies the human-readable description for the resource manager interface. If white space is used, the description should be quoted.

**Example:**

```plaintext
RMCFG[ganglia] TYPE=NATIVE:ganglia DESCRIPTION='resource monitor providing extended resource utilization stats'
```

Moab annotates the `ganglia` 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 URL's of type 'exec://' for that resource manager.

**Example:**

```plaintext
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>`

**Default:** N/A

**Description:** Specifies the event port to use to receive resource manager based scheduling events.

**Example:**

```plaintext
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`

**Default:** N/A

**Description:** Specifies how long a resource manager must be down before any failure triggers associated with the resource manager fire.

**Example:**

```plaintext
RMCFG[base] FAILTIME=3:00
```

If the `base` resource manager is down for three minutes, any resource manager failure triggers fire.
### RMCFG Flags

<table>
<thead>
<tr>
<th>Format</th>
<th>comma delimited list of zero or more of the following: <strong>asyncstart, autostart, autosync, client, fullcp, executionServer, grid, hostingCenter, ignqueueestate, private, pushslavejobupdates, report, shared, slavepeer</strong> or static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>Specifies various attributes of the resource manager. See Flag Details for more information.</td>
</tr>
<tr>
<td>Example</td>
<td>RMCFG[base] FLAGS=static,slavepeer</td>
</tr>
</tbody>
</table>

Moab uses this resource manager to perform a single update of node and job objects reported elsewhere.

### FLOWINTERVAL

<table>
<thead>
<tr>
<th>Format</th>
<th><code>[[[DD:]HH:]MM:]SS</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td><code>01:00:00</code> (one hour)</td>
</tr>
<tr>
<td>Description</td>
<td>Specifies the duration of the flow control sliding window.</td>
</tr>
<tr>
<td>Example</td>
<td>RMCFG[base] FLOWINTERVAL=00:30:00 FLOWMETRIC=jobs FLOWLIMIT=30</td>
</tr>
</tbody>
</table>

The scheduler limits jobs running on this resource manager to no more than 30 jobs every 30 minutes.

### FLOWLIMIT

<table>
<thead>
<tr>
<th>Format</th>
<th><code>&lt;INTEGER&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>Specifies the limit of flow metric consumption allowed within the sliding window.</td>
</tr>
<tr>
<td>Example</td>
<td>RMCFG[base] FLOWINTERVAL=00:30:00 FLOWMETRIC=jobs FLOWLIMIT=30</td>
</tr>
</tbody>
</table>

The scheduler limits jobs running on this resource manager to no more than 30 jobs every 30 minutes.

### FLOWMETRIC

<table>
<thead>
<tr>
<th>Format</th>
<th>one of <strong>jobs, procs, nodes, procseconds</strong>, or <strong>peseconds</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>jobs</td>
</tr>
<tr>
<td>Description</td>
<td>Specifies the metric of consumption of the flow control sliding window.</td>
</tr>
<tr>
<td>Example</td>
<td>RMCFG[base] FLOWINTERVAL=00:30:00 FLOWMETRIC=jobs FLOWLIMIT=30</td>
</tr>
</tbody>
</table>

The scheduler limits jobs running on this resource manager to no more than 30 jobs every 30 minutes.

### FNLIST

<table>
<thead>
<tr>
<th>Format</th>
<th>comma delimited list of zero or more of the following: <strong>clusterquery, jobcancel, jobrequeue, jobresume, jobstart, jobsuspend, queuequery, resourcequery</strong> or <strong>workloadquery</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Example:

```
RMCFG[base]  FNLIST=queuequery
```
Moab only uses this resource manager interface to load queue configuration information.

### HOST

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;STRING&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>localhost</td>
</tr>
<tr>
<td>Description:</td>
<td>The host name of the machine on which the resource manager server is running.</td>
</tr>
<tr>
<td>Example:</td>
<td>RMCFG[base] host=server1</td>
</tr>
</tbody>
</table>

### IGNHNODES

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>FALSE</td>
</tr>
<tr>
<td>Description:</td>
<td>Specifies whether to read in the PBSPro host nodes. This parameter is used in conjunction with USEVNODES. When both are set to TRUE, the host nodes are not queried.</td>
</tr>
<tr>
<td>Example:</td>
<td>RMCFG[pbs] IGNHNODES=TRUE</td>
</tr>
</tbody>
</table>

### JOBCANCELURL

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;protocol&gt;://[&lt;host&gt;[:&lt;port&gt;]][&lt;path&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>Specifies how Moab cancels jobs via the resource manager. (See URL Notes below.)</td>
</tr>
<tr>
<td>Example:</td>
<td>RMCFG[base] JOBCANCELURL=exec:///opt/moab/job.cancel.lsf.pl</td>
</tr>
</tbody>
</table>

Moab executes /opt/moab/job.cancel.lsf.pl to cancel specific jobs.

### JOBEXTENDDURATION

<table>
<thead>
<tr>
<th>Format:</th>
<th>[[[DD:]HH:]MM:]SS,[[[DD:]HH:]MM:]SS[!]&lt;&gt; (or &lt;MIN TIME&gt;[,.&lt;MAX TIME&gt;][])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>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 wallclock) 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 ‘&lt;’ is at the end of the string, however, the job is extended the maximum amount possible.</td>
</tr>
<tr>
<td>Example:</td>
<td>RMCFG[base] JOBEXTENDDURATION=30,1:00:00</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Format</th>
<th>Default</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| **JOBMODIFYURL**| `<protocol>://[<host>[:<port>]][<path>]` |         | Specifies how Moab modifies jobs via the resource manager. (See URL Notes and interface details.) | RMCFG[base] JOBMODIFYURL=exec://$TOOLSDIR/job.modify.dyn.pl  
Moab executes /opt/moab/job.modify.dyn.pl to modify specific jobs. |
| **JOBRSVRECREATE** | Boolean                           | TRUE    | Specifies whether Moab will re-create a job reservation each time job information is updated by a resource manager. (See Considerations for Large Clusters for more information.) | RMCFG[base] JOBRSVRECREATE=FALSE  
Moab only creates a job reservation once when the job first starts. |
| **JOBSTARTURL**  | `<protocol>://[<host>[:<port>]][<path>]` |         | Specifies how Moab starts jobs via the resource manager. (See URL Notes below.) | RMCFG[base] JOBSTARTURL=http://orion.bsu.edu:1322/moab/jobstart.cgi  
Moab triggers the jobstart.cgi script via http to start specific jobs. |
| **JOBSUBMITURL** | `<protocol>://[<host>[:<port>]][<path>]` |         | Specifies how Moab submits jobs to the resource manager. (See URL Notes below.) | RMCFG[base] JOBSUBMITURL=exec://$TOOLSDIR/job.submit.dyn.pl  
Moab submits jobs directly to the database located on host dbserver.flc.com. |
| **JOBSUSPENDURL** | `<protocol>://[<host>[:<port>]][<path>]` |         | 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>]
```

**Default:**
---

**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)

**Description:**
Specifies the maximum number of data staging operations that may be simultaneously active.

**Example:**
```
RMCFG[ds] MAXDSOP=16
```

### MAX.JOB

**Format:**
```
<STRING>
```

**Default:**
---

**Description:**
Specifies the job template to use to check various maximum/excluded job attributes that are specified by the submitter.

**Example:**
```
RMCFG[base] MAX.JOB=maxjob
```
Moab will use the `maxjob` job template to identify and enforce maximum/excluded job attributes.

### 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

---
## MAXJOBS

**Format:** `<INTEGER>`

**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:**

```shell
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:**

```shell
RMCFG[base] MINETIME=5
```

The scheduler batch-processes scheduling events that occur less than five seconds apart.

## MIN.JOB

**Format:** `<STRING>`

**Default:** `---`

**Description:** Specifies the job template to use to check various minimum/required job attributes that are specified by the submitter.

**Example:**

```shell
RMCFG[base] MIN.JOB=minjob
```

Moab uses the `minjob` job template to identify and enforce minimum/required job attributes.

## 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:**

```shell
RMCFG[base] NMPORT=13001
```

The scheduler contacts the node manager located on each compute node at port 13001.

## NODEFAILURESVPROFILE

**Format:** `<STRING>`

**Default:** `N/A`

**Description:** Specifies the rsv template to use when placing a reservation onto failed nodes. (See also...
**NODEFAILURERESERVETIME**

Example:

```
moab.cfg
RMCFG[base] NODEFAILURERSVPROFILE=long
RSVPROFILE[long] DURATION=25:00
RSVPROFILE[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[ganglia] TYPE=NATIVE CLUSTERQUERYURL=ganglia://
RMCFG[ganglia] FLAGS=SLAVEPEER NODESTATEPOLICY=OPTIMISTIC
```

**OMAP**

Format: `<protocol>://[<host>[:<port>]][<path>]`

Default: ---

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 file /opt/moab/omap.dat.

**POLLINTERVAL**

Format: `[[[DD::]HH:]MM:]SS`

Default: 30

Description: Specifies how often the scheduler will poll the resource manager for information.

Example:

```
RMCFG[base] POLLINTERVAL=1:00
```

Moab contacts resource manager base every minute for updates.

**POLLTIMEISRIGID**

Format: `<BOOLEAN>`

Default: **FALSE**
<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>Determines whether the <strong>POLLINTERVAL</strong> parameter is interpreted as an interval or a set time for contacting.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td><strong>RMCFG[base]</strong> POLLTIMEISRIGID=TRUE POLLINTERVAL=1:00:00&lt;br&gt;Moab polls the resource manager at startup and on the hour.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>PORT</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>PTYSTRING</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>RESOURCECREATEURL</strong></th>
</tr>
</thead>
</table>
| **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. |
| **Default:** | **RESOURCECREATEURL** has a default only when the RM type is declared as MSM (RMCFG[] TYPE=Native:MSM). If this is the case, the default is: `exec:///STOOLSDIR/msm/contrib/node.create.pl` |
| **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. |
Moab invokes the `vm.provision.py` script, passing in data as command line arguments, to request a creation of new resources.

<table>
<thead>
<tr>
<th><strong>RESOURCETYPE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Format: `{COMPUTE</td>
</tr>
<tr>
<td>Default: <code>---</code></td>
</tr>
<tr>
<td>Description: Specifies which type of resource this resource manager is configured to control. See Native Resource Managers for more information.</td>
</tr>
<tr>
<td>Example: <code>RMCFG[base] TYPE=NATIVE RESOURCETYPE=FS</code></td>
</tr>
<tr>
<td><strong>RMSTARTURL</strong></td>
</tr>
<tr>
<td>Format: `[exec://&lt;path&gt;</td>
</tr>
<tr>
<td>If <code>exec://</code> is specified, Moab treats the destination as an executable file; if <code>http://</code> is specified, Moab treats the destination as a hypertext transfer protocol file.</td>
</tr>
<tr>
<td>Default: <code>---</code></td>
</tr>
<tr>
<td>Description: Specifies how Moab starts the resource manager.</td>
</tr>
<tr>
<td>Example: <code>RMCFG[base] RMSTARTURL=exec:///tmp/nat.start.pl</code></td>
</tr>
<tr>
<td><strong>RMSTOPURL</strong></td>
</tr>
<tr>
<td>Format: `[exec://&lt;path&gt;</td>
</tr>
<tr>
<td>If <code>exec://</code> is specified, Moab treats the destination as an executable file; if <code>http://</code> is specified, Moab treats the destination as a hypertext transfer protocol file.</td>
</tr>
<tr>
<td>Default: <code>---</code></td>
</tr>
<tr>
<td>Description: Specifies how Moab stops the resource manager.</td>
</tr>
<tr>
<td>Example: <code>RMCFG[base] RMSTOPURL=exec:///tmp/nat.stop.pl</code></td>
</tr>
<tr>
<td><strong>SBINDIR</strong></td>
</tr>
<tr>
<td>Format: <code>&lt;PATH&gt;</code></td>
</tr>
<tr>
<td>Default: <code>N/A</code></td>
</tr>
<tr>
<td>Description: For use with TORQUE; specifies the location of the TORQUE system binaries (supported in TORQUE 1.2.0p4 and higher).</td>
</tr>
<tr>
<td>Example: <code>RMCFG[base] TYPE=pbs SBINDIR=/usr/local/torque/sbin</code></td>
</tr>
</tbody>
</table>
Moab tells TORQUE that its system binaries are located in `/usr/local/torque/sbin`.

### SERVER

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;URL&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td><code>RMCFG[base] server=ll://supercluster.org:9705</code></td>
</tr>
</tbody>
</table>

Moab attempts to use the Loadleveler scheduling API at the specified location.

### SET.JOB

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;STRING&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Specifies the job template to use to force various job attributes regardless of whether or not they are specified by the submitter.</td>
</tr>
</tbody>
</table>
| **Example:**   | `RMCFG[internal] SET.JOB=test
RMCFG[pbs] SET.JOB=test
JOBCFG[test] CLASS=class1` |

Moab uses the `test` job template to identify and enforce mandatory job attributes.

### SLURMFLAGS

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;STRING&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Specifies characteristics of the SLURM resource manager interface.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td><code>RMCFG[slurm] SLURMFLAGS=COMPRESSOUTPUT</code></td>
</tr>
</tbody>
</table>

Moab uses the specified flag to determine interface characteristics with SLURM. The COMPRESSOUTPUT flag instructs Moab to use the compact hostlist format for job submissions to SLURM. The flag NODEDELT AQ UERY instructs Moab to request delta node updates when it queries SLURM for node configuration.

### SOFTTERMSIG

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;INTEGER&gt;</code> or <code>SIG&lt;X&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Specifies what signal to send the resource manager when a job reaches its soft wallclock limit. (See <code>JOBMAXOVERRUN</code>.)</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td><code>RMCFG[base] SOFTTERMSIG=SIGUSR1</code></td>
</tr>
</tbody>
</table>

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`

**Default:** `N/A`

**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.

Note: If this attribute is used, backfill is disabled for the associated resource manager.

**Example:**

```plaintext
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>`

**Default:** `N/A`

**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.

Note: 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:**

```plaintext
RMCFG[base] STARTCMD=/usr/local/bin/qrun
```

Moab uses the specified start command if API failures occur when launching jobs.

### SUBMITCMD

**Format:** `<STRING>`

**Default:** `N/A`

**Description:** Specifies the full path to the resource manager job submission client.

**Example:**

```plaintext
RMCFG[base] SUBMITCMD=/usr/local/bin/qsub
```

Moab uses the specified submit command when migrating jobs.

### SUBMITPOLICY

**Format:** one of **NODECENTRIC** or **PROCENTRIC**

**Default:** **PROCENTRIC**

**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:**

```plaintext
RMCFG[base] SUBMITPOLICY=NODECENTRIC
```

Moab uses the specified submit policy when migrating jobs.
### SUSPENDSIG

**Format:** `<INTEGER>` (valid Unix signal between 1 and 64)

**Default:** `---` *(resource manager specific default)*

**Description:** If set, Moab sends the specified signal to a job when a job suspend request is issued.

**Example:**

```bash
RMCFG[base] SUSPENDSIG=19
```

Moab uses the specified suspend signal when suspending jobs within the base resource manager.

**Note:** `SUSPENDSIG` should not be used with TORQUE or other PBS-based resource managers.

### SYNCJOBID

**Format:** `<BOOLEAN>`

**Default:** `---`

**Description:** Specifies that Moab should migrate jobs to the local resource manager queue with a job ID matching the job's Moab-assigned job ID *(only available with SLURM)*.

**Example:**

```bash
RMCFG[base] TYPE=wiki:slurm SYNCJOBID=TRUE
```

Moab migrates jobs to the SLURM queue with a jobid matching the Moab-assigned job ID.

### 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.

**Default:** `---`

**Description:** Specifies how Moab modifies attributes of the system. This interface is used in data staging.

**Example:**

```bash
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:** `[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.

**Default:** `---`

**Description:** Specifies how Moab queries attributes of the system. This interface is used in data staging.

**Example:**

```bash
RMCFG[base] SYSTEMQUERYURL=file:///tmp/system.query
```
Moab reads `/tmp/system.query` when it queries the system in conjunction with **base** resource manager.

### TARGETUSAGE

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;INTEGER&gt;[%]</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>90%</td>
</tr>
<tr>
<td>Description:</td>
<td>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).</td>
</tr>
<tr>
<td>Example:</td>
<td>RMCFG[storage] TYPE=NATIVE RESOURCETYPE=storage RMCFG[storage] TARGETUSAGE=80%</td>
</tr>
</tbody>
</table>

Moab schedules data migration requests to never exceed 80% usage of the storage resource manager's disk cache and network resources.

### TIMEOUT

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;INTEGER&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>30</td>
</tr>
<tr>
<td>Description:</td>
<td>Time (in seconds) the scheduler waits for a response from the resource manager.</td>
</tr>
<tr>
<td>Example:</td>
<td>RMCFG[base] TIMEOUT=40</td>
</tr>
</tbody>
</table>

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.

### TRANSLATIONSCRIPT

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;STRING&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>Script to be inserted into user command file if migration involves translation from one resource manager type to another. The script takes two arguments where the first is the source resource manager type and the second is the destination resource manager type. Types can be any of the following: PBS, SLURM, LSF, SGE, LOADLEVELER, or BPROC.</td>
</tr>
<tr>
<td>Example:</td>
<td>RMCFG[base] TRANSLATIONSCRIPT=/opt/moab/tools/tscript.sh</td>
</tr>
</tbody>
</table>

Moab inserts a line that will source the specified script into the start of each translated job command file.

### TRIGGER

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;TRIG_SPEC&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>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.</td>
</tr>
<tr>
<td>Example:</td>
<td>RMCFG[base] TRIGGER=&lt;X&gt;</td>
</tr>
</tbody>
</table>
### TYPE

| Format: | <RMTYPE>[;<RMSUBTYPE>] where <RMTYPE> is one of the following: TORQUE, LL, LSF, NATIVE, PBS, RMS, SGE, 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.

### UCALLOCDURATION

| Format: | [[[DD:]HH:]MM:SS] |
| Default: | -- (no allocation duration) |
| Description: | Specifies the minimum duration utility computing resources are allocated when the resource manager's UCTHRESHOLD is satisfied. This feature is used when allocating automatic utility computing resources. |

**Example:**

```
RMCFG[base] UCTHRESHOLD=00:20:00
RMCFG[base] UCALLOCDURATION=00:30:00
RMCFG[base] UCALLOCSIZE=2
```

The resource manager allocates two processors at a time for a period of at least 30 minutes if the cluster maintains a 20-minute backlog for more than three minutes.

### UCALLOCSIZE

| Format: | <INTEGER> (processors) |
| Default: | 1 |
| Description: | Specifies the number of additional nodes to allocate each time a dynamic utility computing threshold is reached. This feature is used primarily with utility computing resources. |

> Either **UCMAXSIZE** or **UCALLOCSIZE** must be specified to enable performance or threshold based automatic utility computing usage.

**Example:**

```
RMCFG[base] UCTHRESHOLD=00:20:00
RMCFG[base] UCALLOCSIZE=4
```

The resource manager allocates four additional nodes each time the utility computing threshold is reached.

### UCMAXSIZE

| Format: | <INTEGER> (processors) |
**Default:** 1

**Description:** Specifies the maximum number of nodes the local cluster can allocate in response to utility computing thresholds. This feature is used primarily with utility computing resources.

*Either UCMAXSIZE or UCALLOCSIZE must be specified to enable performance or threshold based automatic utility computing usage.*

**Example:**

<table>
<thead>
<tr>
<th>RMCFG[base] UCMAXSIZE=256</th>
</tr>
</thead>
</table>

Moab may not allocate more than a total of 256 processors from the utility computing resource even if the utility computing threshold is in violation.

---

### UCTHRESHOLD

**Format:** [[[DD:]HH:]MM:SS

**Default:** -- (no activation threshold)

**Description:** Specifies the cluster backlog duration required before the resource manager automatically activates. This feature is used primarily to activate utility computing resources.

*This parameter is required to enable performance or threshold based automatic utility computing usage.*

**Example:**

<table>
<thead>
<tr>
<th>RMCFG[base] UCTHRESHOLD=00:20:00</th>
</tr>
</thead>
</table>

The resource manager should be activated if the cluster obtains a 20 minute backlog.

---

### UCTHRESHOLDDURATION

**Format:** [[[DD:]HH:]MM:SS

**Default:** -- (no threshold duration)

**Description:** Specifies how long the resource manager's UCTHRESHOLD must be satisfied before resource manager activation is allowed. This parameter prevents statistical spikes from causing unnecessary utility computing allocations. This feature is used primarily to activate utility computing resources.

**Example:**

<table>
<thead>
<tr>
<th>RMCFG[base] UCTHRESHOLD=00:20:00</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMCFG[base] UCTHRESHOLDDURATION=00:03:00</td>
</tr>
</tbody>
</table>

Utility computing resources should be allocated if the cluster maintains a 20 minute backlog for more than three minutes.

---

### 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 parameter.

---
**VARIABLES**

**Format:** `<VAR>=<VAL>[,VAR]=<VAL>`

**Default:** --

**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>`

**Default:** --

**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.

**Default:** --

**Description:** Specifies how Moab queries the resource manager for workload information. (See Native RM, URL Notes, and interface details.)

**Example:**

```
RMCFG[dynamic_jobs]
WORKLOADQUERYURL=exec://$TOOLSDIR/job.query.dyn.pl
```

Moab executes `/opt/moab/tools/job.query.dyn.pl` to obtain updated workload information from resource manager `dynamic_jobs`.

**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
```

For the protocol http, Moab loads the data from the web based HTTP (hypertext transfer protocol) destination.

```
```

### 13.2.2 Resource Manager Configuration Details

As with all scheduler parameters, RMCFG follows the syntax described within the Parameters Overview.

### 13.2.2.1 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. To further assist in configuration, Integration Guides are provided for PBS, SGE, and Loadleveler systems.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>Resource Managers</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL</td>
<td>Loadleveler version 2.x and 3.x</td>
<td>N/A</td>
</tr>
<tr>
<td>LSF</td>
<td>Platform's Load Sharing Facility, version 5.1 and higher</td>
<td>N/A</td>
</tr>
<tr>
<td>Moab</td>
<td>Moab Workload Manager</td>
<td>Use the Moab peer-to-peer (grid) capabilities to enable grids and other configurations. (See Grid Configuration.)</td>
</tr>
<tr>
<td>Native</td>
<td>Moab Native Interface</td>
<td>Used for connecting directly to scripts, files, databases, and web services. (See Managing Resources Directly with the Native Interface.)</td>
</tr>
<tr>
<td>PBS</td>
<td>TORQUE (all versions), OpenPBS (all versions), PBSPro (all versions)</td>
<td>N/A</td>
</tr>
<tr>
<td>RMS</td>
<td>RMS (for Quadrics based systems)</td>
<td>For development use only; not production quality.</td>
</tr>
<tr>
<td>SGE</td>
<td>Sun Grid Engine version 5.3 and higher</td>
<td>N/A</td>
</tr>
<tr>
<td>SSS</td>
<td>Scalable Systems Software Project version 2.0 and higher</td>
<td>N/A</td>
</tr>
<tr>
<td>WIKI</td>
<td>Wiki interface specification version 1.0 and higher</td>
<td>Used for LRM, YRM, ClubMASK, BProc, SLURM, and others.</td>
</tr>
</tbody>
</table>
13.2.2.2 Resource Manager Name

Moab can support more than one resource manager simultaneously. Consequently, the RMCFG parameter takes an index value such as \texttt{RMCFG[clusterA]}. This index value essentially names the resource manager (as done by the deprecated parameter \texttt{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.

13.2.2.3 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 (such as OpenPBS, PBSPro, Loadleveler, SGE, and LSF) 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.

13.2.2.4 Resource Manager Flags

The FLAGS attribute can be used to modify many aspects of a resources manager's behavior.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASYNCSTART</td>
<td>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.)</td>
</tr>
<tr>
<td>AUTOSTART</td>
<td>Jobs staged to this resource manager do not need to be explicitly started by the scheduler. The resource manager itself handles job launch.</td>
</tr>
<tr>
<td>AUTOSYNC</td>
<td>Resource manager starts and stops together with Moab.</td>
</tr>
<tr>
<td>BECOMEMASTER</td>
<td>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.</td>
</tr>
<tr>
<td>CLIENT</td>
<td>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.</td>
</tr>
<tr>
<td>CLOCKSKEWCHECKING</td>
<td>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.</td>
</tr>
<tr>
<td>COLLAPSEDVIEW</td>
<td>The resource manager masks details about local workload and resources and presents only information relevant to the remote server.</td>
</tr>
<tr>
<td>DYNAMICCRED</td>
<td>The resource manager creates credentials within the cluster as needed to support workload. (See Identity Manager Overview.)</td>
</tr>
<tr>
<td>EXECUTIONSERVER</td>
<td>The resource manager is capable of launching and executing batch workload.</td>
</tr>
<tr>
<td>FSISREMOTE</td>
<td>Add this flag if the working file system doesn’t exist on the server to prevent Moab from validating files and directories at migration.</td>
</tr>
<tr>
<td><strong>FULLCP</strong></td>
<td>Always checkpoint full job information (useful with Native resource managers).</td>
</tr>
<tr>
<td><strong>HOSTINGCENTER</strong></td>
<td>The resource manager interface is used to negotiate an adjustment in dynamic resource access.</td>
</tr>
<tr>
<td><strong>IGNQUEUESTATE</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>IGNWORKLOADSTATE</strong></td>
<td>When this flag is applied to a native resource manager, any jobs that are reported via that resource manager's &quot;workload query URL&quot; 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 &quot;Running,&quot; 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 &quot;NATIVE&quot;.</td>
</tr>
<tr>
<td><strong>LOCALRSVEXPORT</strong></td>
<td>When using Moab-type resource managers, Moab will export local reservations when requested.</td>
</tr>
<tr>
<td><strong>MIGRATEALLJOBATTRIBUTES</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>NOAUTHORES</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>NOCREATEALL</strong></td>
<td>RMs with this flag only update resources/jobs discovered by other non-slave RMs.</td>
</tr>
<tr>
<td></td>
<td><strong>Non-Moab slaves cannot report resources not seen by non-slave RMs. Moab slaves can report resources not seen by non-slave RMs, but cannot take independent scheduling action.</strong></td>
</tr>
<tr>
<td><strong>NOCREATERESOURCE</strong></td>
<td>RMs with this flag only update resources discovered by other non-slave RMs.</td>
</tr>
<tr>
<td><strong>PRIVATE</strong></td>
<td>The resources and workload reported by the resource manager are not reported to non-administrator users.</td>
</tr>
<tr>
<td><strong>PUSHSLAVEJOBUPDATES</strong></td>
<td>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).</td>
</tr>
<tr>
<td><strong>REPORT</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>ROOTPROXYJOBSUBMISSION</strong></td>
<td>Enables Admin proxy job submission, which means administrators may submit jobs in behalf of other users.</td>
</tr>
<tr>
<td><strong>SHARED</strong></td>
<td>Resources of this resource manager may be scheduled by multiple independent sources and may not be assumed to be owned by any single source.</td>
</tr>
<tr>
<td><strong>SLAVEPEER</strong></td>
<td>Information from this resource manager may not be used to identify new jobs or nodes. Instead, this information may only be used to update jobs and nodes discovered and loaded from other non-slave resource managers.</td>
</tr>
</tbody>
</table>
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.

Example

```
# resource manager 'torque' should use asynchronous job start
# and report resources in 'grid' mode
RMCFG[torque] FLAGS=asyncstart,grid
```

### 13.2.3 Scheduler/Resource Manager Interactions

In the simplest configuration, Moab interacts with the resource manager using the following four primary functions:

**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.

**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 (LL, PBS, or WIKI) overviews.

In addition to these base commands, other commands are required to support advanced features such as dynamic job support, 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.
13.3 Resource Manager Extensions

- 13.3.1 Resource Manager Extension Specification
- 13.3.2 Resource Manager Extension Values
- 13.3.3 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.

### 13.3.1 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:

<table>
<thead>
<tr>
<th>Resource Manager</th>
<th>Specification Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>TORQUE 2.0+</td>
<td>-l</td>
</tr>
<tr>
<td></td>
<td>&gt; qsub -l nodes=3,qos=high sleepy.cmd</td>
</tr>
<tr>
<td>TORQUE 1.x/OpenPBS</td>
<td>-W x=</td>
</tr>
<tr>
<td></td>
<td>&gt; qsub -l nodes=3 -W x=qos:high sleepy.cmd</td>
</tr>
<tr>
<td>OpenPBS does not support this ability by default but can be patched as described in the PBS Resource Manager Extension Overview.</td>
<td></td>
</tr>
<tr>
<td>Loadleveler</td>
<td>#@comment</td>
</tr>
<tr>
<td></td>
<td>@nodes = 3</td>
</tr>
<tr>
<td></td>
<td>@comment = qos:high</td>
</tr>
<tr>
<td>LSF</td>
<td>-ext</td>
</tr>
<tr>
<td></td>
<td>&gt; bsub -ext advres:system.2</td>
</tr>
<tr>
<td>PBSPro</td>
<td>-l</td>
</tr>
<tr>
<td></td>
<td>&gt; qsub -l advres=system.2</td>
</tr>
<tr>
<td>Use of PBSPro resources requires configuring the server_priv/resourcedef file to define the needed extensions as in the following example:</td>
<td></td>
</tr>
<tr>
<td>advres type=string</td>
<td></td>
</tr>
<tr>
<td>qos type=string</td>
<td></td>
</tr>
<tr>
<td>sid type=string</td>
<td></td>
</tr>
<tr>
<td>sjid type=string</td>
<td></td>
</tr>
<tr>
<td>Wiki</td>
<td>comment</td>
</tr>
</tbody>
</table>
## 13.3.2 Resource Manager Extension Values

Using the resource manager specific method, the following job extensions are currently available:

<table>
<thead>
<tr>
<th>Extension</th>
<th>Format</th>
<th>Default</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVRES</td>
<td>[&lt;RSVID&gt;]</td>
<td>---</td>
<td>Specifies that reserved resources are required to run the job. If &lt;RSVID&gt; 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.</td>
<td>&gt; qsub -l advres=grid.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Resources for the job must come from grid.3</td>
<td>&gt; qsub -l !advres=grid.5</td>
</tr>
<tr>
<td>BANDWIDTH</td>
<td>&lt;DOUBLE&gt; (in MB/s)</td>
<td></td>
<td>Minimum available network bandwidth across allocated resources. (See Network Management.)</td>
<td>&gt; bsub -ext bandwidth=120 chemjob.txt</td>
</tr>
<tr>
<td>DDISK</td>
<td>&lt;INTEGER&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**ADVRES**

**Format:** [<RSVID>]

**Default:** ---

**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.

**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)

**Default:** ---

**Description:** Minimum available network bandwidth across allocated resources. (See Network Management.)

**Example:**

> bsub -ext bandwidth=120 chemjob.txt

---

**DDISK**

**Format:** <INTEGER>
<table>
<thead>
<tr>
<th><strong>Default:</strong> 0</th>
<th><strong>Description:</strong> Dedicated disk per task in MB.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example:</strong> qsub -l ddisk=2000</td>
<td></td>
</tr>
</tbody>
</table>

**DEADLINE**

<table>
<thead>
<tr>
<th><strong>Format:</strong> [[DD:]HH:]MM:SS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default:</strong> ---</td>
</tr>
<tr>
<td><strong>Description:</strong> Relative completion deadline of job (from job submission time).</td>
</tr>
<tr>
<td><strong>Example:</strong> &gt; qsub -l deadline=2:00:00,nodes=4 /tmp/bio3.cmd</td>
</tr>
</tbody>
</table>

**DEPEND**

| **Format:** [<DEPENDTYPE>:][{jobname|jobid}.]<ID>:[{jobname|jobid}.]<ID>... |
|-----------------------------------------------|
| **Default:** 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**

<table>
<thead>
<tr>
<th><strong>Format:</strong> &lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default:</strong> 0</td>
</tr>
<tr>
<td><strong>Description:</strong> Dedicated memory per task in bytes.</td>
</tr>
<tr>
<td><strong>Example:</strong> msub -l DMEM=20480</td>
</tr>
</tbody>
</table>

**EPILOGUE**

<table>
<thead>
<tr>
<th><strong>Format:</strong> &lt;STRING&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default:</strong> ---</td>
</tr>
<tr>
<td><strong>Description:</strong> 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=&lt;file&gt;. The file can be designated with an absolute or relative path.</td>
</tr>
<tr>
<td><strong>Note:</strong> This parameter works only with TORQUE.</td>
</tr>
<tr>
<td><strong>Example:</strong> msub -l epilogue=epilogue_script.sh job.sh</td>
</tr>
</tbody>
</table>
### EXCLUDENODES

**Format:** `{<nodeid>|<node_range>}[,...]`

**Default:** ---

**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>]...`

**Default:** ---

**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>`

**Default:** ---

**Description:** Generic job attribute associated with job.

**Example:**
```
> qsub -l gattr=bigjob
```

### GEOMETRY

**Format:** `{(<TASKID>[,<TASKID>[,...]][(<TASKID>[,...])...])}`

**Default:** ---

**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>]`

**Default:** ---

**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.)
### GPUs

**Format:**

```
msub -l nodes=<VALUE>:ppn=<VALUE>:gpus=<VALUE>
```

Moab does not support requesting GPUs as a GRES. Submitting `msub -l gres=gpus:x` does not work.

**Default:**
---

**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 "gpus" attribute. Moab automatically parses the "gpus" attribute and assigns them in the correct manner.

**Examples:**

```
> msub -l nodes=1:ppn=2:gpus=1
```
Submits a job that requests 1 node, 2 processors and 1 gpu per node (2 gpus total).

```
> msub -l nodes=2:ppn=2:gpus=1
```
Submits a job that requests 2 nodes, 2 processors and 1 gpu per node (4 gpus total).

```
> msub -l nodes=4:gpus=1
```
Submits a job that requests 4 tasks, 1 processor and 1 gpu per task (4 gpus total).

```
> msub -l nodes=4:gpus=2+1:ppn=2,walltime=600
```
Submits a job that requests 2 different types of tasks, the first is 1 task with 2 processors, the second is 4 tasks, each with 1 processor and 2 gpus.

### GRES and SOFTWARE

**Format:**

Percent sign (%) delimited list of generic resources where each resource is specified using the format `<RESTYPE>{+|:}<COUNT>`

**Default:**
---

**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:**

'+' delimited list of hostnames; also, ranges and regular expressions

**Default:**
---

**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.*

**Examples:**

```
> qsub -l gmetric=bioversion:ge:133244 testj.txt
```
**JGROUP**

**Format:**  
<JOBGROUPID>

**Default:**  ---

**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)

**Default:**  ---

**Description:** Associates various flags with the job.

**Example:**  
> qsub -l nodes=1,walltime=3600,jobflags=advres myjob.py

---

**LOGLEVEL**

**Format:**  <INTEGER>

**Default:**  ---

**Description:** Per job log verbosity.

**Example:**  
> qsub -l -W x=loglevel:5 bw.cmd

Job events and analysis will be logged with level 5 verbosity.

---

**MAXMEM**

**Format:**  <INTEGER> (in megabytes)

**Default:**  ---

**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>`

**Default:**

**Description:** Maximum CPU load the job may consume across all tasks before the **JOBPROC** action is taken.

**Example:**

```bash
> 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.

### MINPREEMPTTIME

**Format:** `[[DD:]HH:]MM:]SS`

**Default:**

**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:**

```bash
> qsub -l minpreempttime=900 bw.cmd
```

Job cannot be preempted until it has run for 15 minutes.

### MINPROCSPEED

**Format:** `<INTEGER>`

**Default:** 0

**Description:** Minimum processor speed (in MHz) for every node that this job will run on.

**Example:**

```bash
> 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:** 1:00:00

**Description:** Minimum wallclock limit job must run before being eligible for extension. (See **JOBEXTENDDURATION** or **JOBEXTENDSTARTWALLTIME**.)

**Example:**

```bash
> 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][|<SRCURL>...]%]<DSTURL>`

**Default:**

**Description:** Indicates a job has data staging requirements. The source URL(s) listed will be transferred to the

```bash
> qsub -W x=MAXPROC:4 bw.cmd
> qsub -l minpreempttime=900 bw.cmd
> qsub -W x=MINPROCSPEED:2000 bw.cmd
> qsub -l minwclimit=300,walltime=16000 bw.cmd
```
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 use 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='mstagein=file://$HOME/helperscript.sh|file:///home/dev/datafile.txt%script.sh

Copy datafile.txt and helperscript.sh 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>][<SRCURL>...%]<DSTURL>
Default:  ---
Description: Indicates 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:

```bash
> msub -W x='mstageout=ssh://$DEST/$HOME/resultfile1.txt|ssh://host/home/dev/resu
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**

<table>
<thead>
<tr>
<th>Format:</th>
<th>one of <code>SHARED, SINGLEJOB, SINGLETASK, SINGLEUSER,</code> or <code>UNIQUEUSER</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>--</td>
</tr>
<tr>
<td>Description:</td>
<td>Specifies how node resources should be accessed. (See Node Access Policies for more information).</td>
</tr>
</tbody>
</table>

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:

```bash
> 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.

**NALLOCOPOLICY**

<table>
<thead>
<tr>
<th>Format:</th>
<th>one of the valid settings for the parameter <code>NODEALLOCATIONPOLICY</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>--</td>
</tr>
<tr>
<td>Description:</td>
<td>Specifies how node resources should be selected and allocated to the job. (See Node Allocation Policies for more information.)</td>
</tr>
</tbody>
</table>

Example:

```bash
> qsub -l nallocpolicy=minresource bw.cmd
```

Job should use the `minresource` node allocation policy.

**NCPUS**

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;INTEGER&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>--</td>
</tr>
<tr>
<td>Description:</td>
<td>The number of processors in one task where a task cannot span nodes. If NCPUS is used, then the resource manager's <code>SUBMITPOLICY</code> should be set to <code>NODECENTRIC</code> to get correct behavior. <code>-l ncpus=&lt;#&gt;</code> is equivalent to <code>-l nodes=1:ppn=&lt;#&gt;</code> when JOBNODEMATCHPOLICY is set to <code>EXACTNODE</code>. NCPUS is used when submitting jobs to an SMP.</td>
</tr>
</tbody>
</table>

**NMATCHPOLICY**

<table>
<thead>
<tr>
<th>Format:</th>
<th>one of the valid settings for the parameter <code>JOBNODEMATCHPOLICY</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>--</td>
</tr>
<tr>
<td>Description:</td>
<td>Specifies how node resources should be selected and allocated to the job.</td>
</tr>
</tbody>
</table>
### Example:
```
> qsub -l nodes=2 -W x=nmatchpolicy:exactnode bw.cmd
```
Job should use the **EXACTNODE** **JOBNODEMATCHPOLICY**.

### NODESET
**Format:** `<SETTYPE>:<SETATTR>[[:<SETLIST>]]`
**Default:** ---
**Description:** Specifies nodeset constraints for job resource allocation. (See the NodeSet Overview for more information.)
**Example:**
```
> qsub -l nodeset=ONEOF:PROCSPEED:350:400:450 bw.cmd
```

### NODESETCOUNT
**Format:** `<INTEGER>`
**Default:** ---
**Description:** Specifies how many node sets a job uses. See the Node Set Overview for more information.
**Example:**
```
> msub -l nodersetcount=2
```

### NODESETDELAY
**Format:** `[[DD:]HH:]MM:]SS`
**Default:** ---
**Description:** The maximum delay allowed when scheduling a job constrained by NODESETS until Moab discards the NODESET request and schedules the job normally.
**Example:**
```
> qsub -l nodersetdelay=300,walltime=16000 bw.cmd
```

### NODESETISOPTIONAL
**Format:** `<BOOLEAN>`
**Default:** ---
**Description:** Specifies whether the nodeset constraint is optional. (See the NodeSet Overview for more information.)

<table>
<thead>
<tr>
<th>Requires SCHEDCFG[] Flags</th>
<th>allowperjobnodersetisoptional</th>
</tr>
</thead>
</table>
**Example:**
```
> msub -l nodersetisoptional=true bw.cmd
```

### OPSYS
**Format:** `<OperatingSystem>`
**Default:** ---
**Description:** Specifies the job's required operating system.
**Example:**
```
> qsub -l nodes=1,opsys=rh73 chem92.cmd
```
### PARTITION

**Format:** `<STRING>{,|:}<STRING>...`

**Default:** `---`

**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.

### PREF

**Format:** `[feature|variable]:<STRING>[:<STRING>]...`

**Default:** `---`

**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>`

**Default:** `---`

**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>`

**Default:** `---`

**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.

| Example: | msub -l prologue=prologue_script.sh job.s |

QoS

**Format:** `<STRING>`

**Default:** `---`

**Description:** Requests the specified QoS for the job.

| Example: | > qsub -l walltime=1000,qos=highprio biojob.cmd |

**QUEUEJOB**

**Format:** `<BOOLEAN>`

**Default:** `TRUE`

**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:

`<ATTRIBUTE>[>=|>|<=|<|=_]<VERSION>`

**Default:** `---`

**Description:** Indicates required node attributes.

| Example: | > qsub -l reqattr=matlab=7.1 testj.txt |

**RESFAILPOLICY**

**Format:** one of CANCEL, HOLD, IGNORE, NOTIFY, or REQUEUE

**Default:** `---`

**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>`

**Default:** `---`
<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td><code>msub -l rmtype=ll</code> Only run job on a Loadleveler destination resource manager.</td>
</tr>
</tbody>
</table>

**SIGNAL**

<table>
<thead>
<tr>
<th>Format</th>
<th><code>&lt;INTEGER&gt;[@&lt;OFFSET&gt;]</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>---</td>
</tr>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
<tr>
<td>Example</td>
<td><code>&gt; msub -l signal=32@120 bio45.cmd</code></td>
</tr>
</tbody>
</table>

**SPRIORITY**

<table>
<thead>
<tr>
<th>Format</th>
<th><code>&lt;INTEGER&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>0</td>
</tr>
<tr>
<td>Description</td>
<td>Allows Moab administrators to set a system priority on a job (similar to <code>setspri</code>). This only works if the job submitter is an administrator.</td>
</tr>
<tr>
<td>Example</td>
<td><code>&gt; qsub -l nodes=16,spriority=100 job.cmd</code></td>
</tr>
</tbody>
</table>

**TASKDISTPOLICY**

<table>
<thead>
<tr>
<th>Format</th>
<th><code>RR</code> or <code>PACK</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>---</td>
</tr>
<tr>
<td>Description</td>
<td>Allows users to specify task distribution policies on a per job basis. (See Task Distribution Overview)</td>
</tr>
<tr>
<td>Example</td>
<td><code>&gt; qsub -l nodes=16,taskdistpolicy=rr job.cmd</code></td>
</tr>
</tbody>
</table>

**TEMPLATE**

<table>
<thead>
<tr>
<th>Format</th>
<th><code>&lt;STRING&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>---</td>
</tr>
<tr>
<td>Description</td>
<td>Specifies a job template to be used as a set template. The requested template must have <code>SELECT=TRUE</code> (See Job Templates.)</td>
</tr>
<tr>
<td>Example</td>
<td><code>&gt; msub -l walltime=1000,nodes=16,template=biojob job.cmd</code></td>
</tr>
</tbody>
</table>

**TERMTIME**

<table>
<thead>
<tr>
<th>Format</th>
<th><code>&lt;TIMESPEC&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>0</td>
</tr>
<tr>
<td>Description</td>
<td>Specifies the time at which Moab should cancel a queued or active job. (See Job Deadline)</td>
</tr>
</tbody>
</table>
### TPN

**Format:** `<INTEGER>[+]`

**Default:** 0

**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.

**Note on 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.

**Example:**

```bash
> msub -l nodes=10,walltime=600,tpn=4 job.cmd
```

### TRIG

**Format:** `<TRIGSPEC>`

**Default:** ---

**Description:** Adds trigger(s) to the job. (See the Trigger Specification Page for specific syntax.)

> Job triggers can only be specified if allowed by the QoS flag `trigger`.

**Example:**

```bash
> qsub -l trig=start:exec@/tmp/email.sh job.cmd
```

### TRL (Format 1)

**Format:** `<INTEGER>[@<INTEGER>][:<INTEGER>[@<INTEGER>]]...`

**Default:** 0

**Description:** Specifies alternate task requests with their optional walltimes.

**Example:**

```bash
> msub -l trl=2@500:4@250:8@125:16@62 job.cmd
```

or

```bash
> 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.

**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.

**TTC**

**Format:** `<INTEGER>`

**Default:** 0

**Description:** Total tasks allowed across the number of hosts requested. TTC is supported in the Wiki resource manager for SLURM. Compressed output must be enabled in the moab.cfg file. (See SLURMFLAGS for more information). NODEACCESSPOLICY should be set to SINGLEJOB and JOBNODEMATCHPOLICY should be set to EXACTNODE in the moab.cfg file.

**Example:**

```
> msub -l nodes=10,walltime=600,ttc=20 job.cmd
```

In this example, assuming all the nodes are 8 processor nodes, the first allocated node will have 10 tasks, the next node will have 2 tasks, and the remaining 8 nodes will have 1 task each for a total task count of 20 tasks.

**VAR**

**Format:** `<ATTR>:<VALUE>`

**Default:** ---

**Description:** Adds a generic variable or variables to the job.

**Example:**

```
VAR=testvar1:testvalue1
VAR=testvar1:testvalue1+testvar2:testvalue2+testvar3:testvalue3
```

---

### 13.3.3 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 1**

```
#@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 2**

```
# 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**

```
> 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.

**See Also**

- Resource Manager Overview
13.4 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.

- 13.4.1 Resource Manager Specific Interfaces
- 13.4.2 Wiki Interface
- 13.4.3 SSS Interface

13.4.1 Resource Manager Specific Interfaces

If you require tighter integration and need additional instruction, see 13.5 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 Cluster Resources.

13.4.2 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.

13.4.3 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.

See Also

- Creating New Tools within the Native Resource Manager Interface
13.5 Managing Resources Directly with the Native Interface

- 13.5.1 Native Interface Overview
- 13.5.2 Configuring the Native Interface
  - 13.5.2.1 Configuring the Resource Manager
  - 13.5.2.2 Reporting Resources
- 13.5.3 Generating Cluster Query Data
  - 13.5.3.1 Flat Cluster Query Data
  - 13.5.3.2 Interfacing to Ganglia
  - 13.5.3.3 Interfacing to FLEXlm
  - 13.5.3.4 Interfacing to Nagios
  - 13.5.3.5 Interfacing to Supermon
  - 13.5.3.6 Interfacing via HTTP
- 13.5.4 Configuring Node Specific Query URLs
- 13.5.5 Configuring Resource Types
- 13.5.6 Creating New Tools to Manage the Cluster

13.5.1 Native Interface Overview

Holistic Scheduling - The Native Resource Manager is a video tutorial of a session offered at Moab Con that offers further details for understanding the native resource manager.

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.

13.5.2 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.

13.5.2.1 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
```

### 13.5.2.2 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`, `HTTP`, `GANGLIA`, and `SQL` are allowed. If a protocol is not specified, the protocol `EXEC` is assumed.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXEC</td>
<td>Execute the script specified by the URL path. Use the script <code>stdout</code> as data.</td>
</tr>
<tr>
<td>FILE</td>
<td>Load the file specified by the URL path. Use the file contents as data.</td>
</tr>
<tr>
<td>GANGLIA</td>
<td>Query the Ganglia service located at the URL host and port. Directly process the query results using native Ganglia data formatting.</td>
</tr>
<tr>
<td>HTTP</td>
<td>Read the data specified by the URL. Use the raw data returned.</td>
</tr>
<tr>
<td>SQL</td>
<td>Load data directly from an SQL database using the <code>FULL</code> format described below.</td>
</tr>
</tbody>
</table>

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
```

### 13.5.3 Generating Cluster Query Data

#### 13.5.3.1 Flat Cluster Query Data

If the `EXEC`, `FILE`, or `HTTP` 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 Wiki Interface Specification where attributes are delimited by white space rather than `;`:

```
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
```

#### 13.5.3.2 Interfacing to Ganglia

Moab can use the information returned from Ganglia, a cluster or grid monitoring system. The information returned from Ganglia is combined with the information reported from other resource managers. An example configuration can be as follows:

```
RMCFG[TORQUE] TYPE=pbs
RMCFG[ganglia] TYPE=NATIVE CLUSTERQUERYURL=ganglia://<NodeName>:<Port>
RMCFG[ganglia] FLAGS=SLAVEPEER NODESTATEPOLICY=OPTIMISTIC
```

 `<nodeName>` is the name of a machine with Ganglia running on it. Also, `<Port>` is the xml port number to
query Ganglia. If only `ganglia://` is supplied as the CLUSTERQUERYURL, Moab will query the `localhost` on Ganglia’s default port, 8649.

If Ganglia and Moab are running on different machines, the machine running Moab needs to be specified as a trusted host in Ganglia’s configuration file.

Because Ganglia is not a real resource manager, in that it does not manage a job queue, Moab cannot control it or manage it, it can only read in information. TORQUE is a real resource manager in that it reports nodes and can start jobs on those nodes. The two can run concurrently without any issue, because their “responsibilities” do not overlap. However, it is mostly true that if Ganglia and TORQUE report conflicting data, you will want to trust TORQUE over Ganglia. For this reason you give the Ganglia RM the "slave" flag. Also, Ganglia cannot report node "state" where state means "availability to run jobs."

To verify that Ganglia is correctly reporting information, issue the `mdia -R -v` command or run `telnet localhost 8649` and verify that appropriate XML is displayed.

The following list of Ganglia metrics are supported up to Ganglia version 3.1.1:

- bytes_in
- bytes_out
- cpu_num
- cpu_speed
- disk_free
- disk_total
- load_one
- machine_type
- mem_free
- mem_total
- os_name
- os_release
- swap_free
- swap_total

Information reported by Ganglia can be used to prioritize nodes using the `NODECFG[] PRIORITYF` parameter in conjunction with the `NODEALLOCATIONPOLICY` of `PRIORITY`.

### 13.5.3.3 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:

```perl
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 `lmutil`’s `-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 UPDATING=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:

```plaintext
RMCFG[FLEXlm]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm]   CLUSTERQUERYURL=exec://$TOOLSDIR/license.mon.flexLM.pl
...
```

Once this change is made, restart Moab. The command `mdiaq -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 `mdiaq -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:

```plaintext
RMCFG[FLEXlm1]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm1]   CLUSTERQUERYURL=exec://$TOOLSDIR/license.mon.flexLM1.pl
RMCFG[FLEXlm2]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm2]   CLUSTERQUERYURL=exec://$TOOLSDIR/license.mon.flexLM2.pl
RMCFG[FLEXlm3]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm3]   CLUSTERQUERYURL=exec://$TOOLSDIR/license.mon.flexLM3.pl
...
```

For an overview of license management, including job submission syntax, see Section 13.7, License Management.

It may be necessary to increase the default limit, `MMAXGRES`. See Appendix D for more implementation details.

### 13.5.3.4 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 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 `mdia -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 `mdia -n -v`.

```
> mdia -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)
```

### 13.5.3.5 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 "mdia -R -v." The output should be similar to the following example, specifically there are no errors about the CLUSTERQURYURL.
Run the Perl script by itself. The script's results should look similar to this:

```
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: ---
```

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.
13.5.3.6 Interfacing via HTTP

Native resource managers using HTTP URLs send and receive information using the standard HTTP 1.0 protocol. Information is sent using the HTTP GET method, while results are to be returned in the HTTP body using the format described in the Flat Cluster Query Data section. Not all available native resource manager query URLs are currently supported. Following is a chart showing the supported query URLs and the parameters that will be provided by MOAB in the GET request.

<table>
<thead>
<tr>
<th>Query URL</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLUSTERQUERYURL</td>
<td>none</td>
</tr>
<tr>
<td>JOBCANCELURL</td>
<td>jobname=&lt;JOB_ID&gt;</td>
</tr>
<tr>
<td>JOBMODIFYURL</td>
<td>jobname=&lt;JOB_ID&gt; attr=&lt;Attribute_Name&gt; value=&lt;Attribute_Value&gt;</td>
</tr>
<tr>
<td>WORKLOADQUERYURL</td>
<td>none</td>
</tr>
</tbody>
</table>

CGI scripts pointed to by the query URLs should always return at least one line of output on success to insure that Moab does not consider empty result sets to be a failure. In the case of empty result sets, this can be accomplished by returning an empty comment line (i.e., the '#' character followed by a newline).

13.5.4 Configuring Node Specific Query URLs

It is possible to have a separate CLUSTERQUERYURL for each node. This is possible using the NODECFG parameter for each node or for the DEFAULT node. Moab will look first on the specific node for CLUSTERQUERYURL information. If no information is found on the specific node it will look for CLUSTERQUERYURL information on the Resource Manager. If the Resource Manager has no query information specified then it will use the CLUSTERQUERYURL command configured for the DEFAULT node.

The example configuration below demonstrates a possible setup.

```
RMCFG[local] TYPE=NATIVE
RESOURCENAME node1,node2,node3,node4,flexlm1
NODECFG[DEFAULT] CLUSTERQUERYURL=exec:///usr/local/bin/query.pl
NODECFG[flexlm1] CLUSTERQUERYURL=http://supercluster.org/usr/local/flquery.cgi
```

In the example above, a four node cluster and a license manager are controlled via the native interface. The state of the four compute nodes will be determined by running the /usr/local/bin/query.pl query command (remotely on the node) specified within the DEFAULT NODECFG parameter while querying the license manager will be accomplished using the /usr/local/bin/flquery.cgi script. For local executable scripts, the launched script is either locally generated or taken from the library of contributed native scripts included with the distribution file.

As above, an optional parameter, RESOURCENAME, may be specified to constrain which resources obtained by the native interface should be processed. By default, all resources described by the interface data are loaded. The RESOURCENAME parameter, if specified, acts as a filter eliminating either full or extension resource information from being incorporated. If an environment exists where data is not aggregated, and the native interface provides primary node information, the RESOURCENAME parameter is required to indicate to Moab which resources should be in the cluster.

13.5.5 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.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>EXPLANATION</th>
</tr>
</thead>
</table>


13.5.6 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.

<table>
<thead>
<tr>
<th>Resource Manager URL</th>
<th>Description</th>
<th>Input</th>
<th>Output</th>
<th>Example</th>
</tr>
</thead>
</table>
| **CLUSTERQUERYURL**  | Queries resource state, configuration, and utilization information for compute nodes, networks, storage systems, software licenses, and other resources. For more details, see RM configuration. | --- | Node status and configuration for one or more nodes. See Resource Data Format. | RM CFG[v-stor] CLUSTERQUERYURL=exec://$HOME/storquery.pl
Moab will execute the storquery.pl script to obtain information about ‘v-stor’ resources. |
| **JOBMODIFYURL**     | Modified a job or application. For more details, see RM configuration. | [-j <JOBEXPR>] [--s<et|clear|increment|decrement>] <ATTR>[=<VALUE>] [ATTR][=<VALUE>]]... | --- | RM CFG[v-stor] JOBMODIFYURL=exec://$HOME/jobmodify.pl
Moab will execute the jobmodify.pl script to modify the specified job. |
| **JOBRESUMEURL**     | Resumes a suspended job or application. | <JOBID> | --- | RM CFG[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>]
```

**Output:**

---

**Example:**

```bash
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>]
[RFLAGS=<RFLAGS>] [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**.

**Output:**

---

**Example:**

```bash
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>
```

**Output:**

---

**Example:**

```bash
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.

---

RMCFG[v-stor] JOBSTARTURL=exec://$HOME/jobstart.pl
RMCFG[v-stor] JOBSUBMITURL=exec://$HOME/jobsubmit.pl
RMCFG[v-stor] JOBSUSPENDURL=exec://$HOME/jobsuspend.pl
**NODEPOWERURL**

**Description:** Allows Moab to issue IPMI power commands.

**Input:** `<NODEID>[,<NODEID>] ON | OFF`

**Output:** --

**Example:**

```
RMCFG[warewulf] 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.

**Input:**

```
default <ATTR>
```

**ATTR** is one of images

**Output:** `<STRING>`

**Example:**

```
RMCFG[warewulf] SYSTEMMODIFYURL=exec://$HOME/checkimage.pl
```

Moab will load the list of images available from warewulf using the `checkimage.pl` script.

---

**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:**

```
deafult <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, webservice, etc) using the WIKI job description language.
```

For more details, see RM configuration.

**Input:** --

**Output:** `<STRING>`
Moab will load job/workload information by executing the `job.query.xt3.pl` script.

See Also

- `mdiafg -R` command (evaluate resource managers)
- License Management
13.6 Utilizing Multiple Resource Managers

13.6.1 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.

13.6.2 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.

13.6.3 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.

13.6.4 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:

```
RESOURCELIST    node01,node02
...              
RMCFG[base]     TYPE=PBS         
RMCFG[NETWORK]  TYPE=NATIVE:AFULL 
RMCFG[NETWORK]  CLUSTERQUERYURL=/tmp/network.sh 
RMCFG[FS]       TYPE=NATIVE:AFULL      
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` NETUSAGE=`echo "$RX + $TX" | bc`;
```

The preceding script would output something like the following:

```
node01 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)
  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:

```
.....
RMC[ RIGHT]  POLICINTERVAL=24:00:00  POLITICALISRGID=TRUE
RMC[FILE]   TYPE=NATIVE:AGFULL
RMC[FILE]   RESOURCE=FS
RMC[FILE]  CLUSTERQUERYURL=/tmp/fs.pl
.....
```

Assuming that `/tmp/fs.pl` outputs something of the following **format**: DEFAULT STATE=idle AFS=<fs id="u1" size="789456"></fs><fs id="u2" size="123456"></fs>

This will track disk usage for users user1 and user2 every 24 hours.
13.7 License Management

- 13.7.1 License Management Overview
- 13.7.2 Controlling and Monitoring License Availability
- 13.7.3 Requesting Licenses w/in Jobs

13.7.1 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.
- Identical licenses, regardless of case, are invalid because case recognition is insensitive. Thus, two licenses spelled the same with different capitalization are still recognized as the same license, and are thus invalid.

13.7.2 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.

13.7.2.1 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 |
| ... |

13.7.2.2 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.

13.7.2.3 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.

### 13.7.3 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
```

**See Also**

- Native Resource Manager License Configuration
- License Ownership with Advance Reservations
- Multi-Cluster License Sharing with Moab Workload Manager for Grids Interfaces
13.8 Resource Provisioning

- 13.8.1 Resource Provisioning Overview
- 13.8.2 Configuring Provisioning

13.8.1 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: (1) operating system (OS) and (2) 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.

13.8.2 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."

See Also

- Native Resource Manager Overview
- Appendix O: Resource Manager Integration
13.9 Managing Networks

13.9.1 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

13.9.2 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 or VPC requests. These VLAN’s essentially disconnect the job from either incoming or outgoing communication with other compute nodes.

13.9.2.1 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
```

13.9.2.2 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
```

13.9.3 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.

### 13.9.5 Providing Per-QoS and Per-Job Bandwidth Guarantees

Intra-job bandwidth guarantees can be requested on a per job and/or per QoS basis using the BANDWIDTH resource manager extensions (for jobs) and the MINBANDWIDTH QoS attribute (for QoS limits). 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**

```bash
> 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.

### 13.9.6 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.

### 13.9.7 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:

```bash
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.

```bash
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.

13.9.8 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.

See Also

- Native Resource Manager Overview
- Network Utilization Statistics
13.10 Intelligent Platform Management Interface

- 13.10.1 IPMI Overview
- 13.10.2 Node IPMI Configuration
- 13.10.3 Installing IPMItool
- 13.10.4 Setting-up the BMC-Node Map File
- 13.10.5 Configuring Moab's IPMI Tools
- 13.10.6 Configuring Moab
- 13.10.7 Ensuring Proper Setup

13.10.1 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.

13.10.2 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.

13.10.3 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. 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
```

13.10.4 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 "mdia -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.

### 13.10.5 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.

### 13.10.6 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.
13.10.7 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)
  Utilized   Resources: ---
  Dedicated  Resources: ---
  Generic Metrics:   CPU0_TEMP=42.00, CPU1_TEMP=30.00, SYS_TEMP=29.00
```

# IPMI - Power off compute node if its CPU0 temperature exceeds 100 degrees Celsius.
GEVENTCFG[CPU0_TEMP>100] action=off
13.11 Resource Manager Translation

- 13.11.1 Translation Overview
- 13.11.2 Translation Enablement Steps

13.11.1 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.

13.11.2 Translation Enablement Steps

To enable translation, you must:

- edit the Moab tools configuration file.
- copy/rename/link the emulation scripts to a shorter, easier-to-use name.

13.11.2.1 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}"
```

13.11.2.2 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
```
14.0 Troubleshooting and System Maintenance

- 14.1 Internal Diagnostics
- 14.2 Logging Facilities
- 14.3 Using the Message Buffer
- 14.4 Notifying Administrators of Failures and Critical Events
- 14.5 Issues with Client Commands
- 14.6 Tracking System Failures
- 14.7 Problems with Individual Jobs
- 14.8 Diagnostic Scripts
14.1 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.

### 14.1.1 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:

<table>
<thead>
<tr>
<th>Object/Subsystem</th>
<th>mdiag Flag</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account</td>
<td>-a</td>
<td>Shows detailed account configuration information.</td>
</tr>
<tr>
<td>Blocked</td>
<td>-b</td>
<td>Indicates why blocked (ineligible) jobs are not allowed to run.</td>
</tr>
<tr>
<td>Class</td>
<td>-c</td>
<td>Shows detailed class configuration information.</td>
</tr>
<tr>
<td>Config</td>
<td>-C</td>
<td>Shows configuration lines from <code>moab.cfg</code> and whether or not they are valid.</td>
</tr>
<tr>
<td>FairShare</td>
<td>-f</td>
<td>Shows detailed fairshare configuration information as well as current fairshare usage.</td>
</tr>
<tr>
<td>Group</td>
<td>-g</td>
<td>Shows detailed group information.</td>
</tr>
<tr>
<td>Job</td>
<td>-j</td>
<td>Shows detailed job information. Reports corrupt job attributes, unexpected states, and excessive job failures.</td>
</tr>
<tr>
<td>Frame/Rack</td>
<td>-m</td>
<td>Shows detailed frame/rack information.</td>
</tr>
<tr>
<td>Node</td>
<td>-n</td>
<td>Shows detailed node information. Reports unexpected node states and resource allocation conditions.</td>
</tr>
<tr>
<td>Priority</td>
<td>-p</td>
<td>Shows detailed job priority information including priority factor contributions to all idle jobs.</td>
</tr>
<tr>
<td>QoS</td>
<td>-q</td>
<td>Shows detailed QoS information.</td>
</tr>
<tr>
<td>Reservation</td>
<td>-r</td>
<td>Shows detailed reservation information. Reports reservation corruption and unexpected reservation conditions.</td>
</tr>
<tr>
<td>Resource Manager</td>
<td>-R</td>
<td>Shows detailed resource manager information. Reports configured and detected state, configuration, performance, and failures of all configured</td>
</tr>
</tbody>
</table>
### 14.1.2 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.

### 14.1.3 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.

### 14.1.4 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.

<table>
<thead>
<tr>
<th>Recovery Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>die</td>
<td>Moab will exit, and if core files are externally enabled, will create a core file for analysis. (This is the default behavior.)</td>
</tr>
<tr>
<td>ignore</td>
<td>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.</td>
</tr>
<tr>
<td>restart</td>
<td>When a <strong>SIGSEGV</strong> 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.</td>
</tr>
<tr>
<td>trap</td>
<td>When a <strong>SIGSEGV</strong> 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.</td>
</tr>
</tbody>
</table>

### 14.1.5 Recovering from Server and VM Failures

When Moab runs a job with a compute resource manager, such as TORQUE, and a node fails, Moab continues running the job by default. Typically the job aborts or runs past its walltime and fails. The JOBACTIONONNODEFAILRE and JOBACTIONONNODEFAILUREDURATION parameters change this default behavior.
With these settings, Moab waits 120 seconds after detecting a node is down before checking its status again. If the node does not recover, Moab requeues the workload. In the case of on-demand VM jobs, Moab attempts to destroy the underlying VMs and create new ones on either the same hypervisor or a new one, depending on whether the hypervisor or VM failed. As a result, Moab makes recovery from VM and server failures transparent and restarts the workload quickly.

To ensure a quick recovery, shorten some of the poll intervals, such as node_check_rate for pbs_server and RMPOLLINTERVAL and JOBACTIONONNODEFAILEDCUSTOMIZATION for Moab. For example, setting node_check_rate to 15, RMPOLLINTERVAL to 10, and JOBACTIONONNODEFAILEDCUSTOMIZATION to 30 causes Moab to recognize a node failure and a need to requeue in about 70 seconds.

**See Also**

- Troubleshooting Individual Jobs
14.2 Logging Facilities

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.

- 14.2.1 Log Facility Configuration
- 14.2.2 Status Information
- 14.2.3 Scheduler Warnings
- 14.2.4 Scheduler Alerts
- 14.2.5 Scheduler Errors
- 14.2.6 Searching Moab Logs
- 14.2.7 Event Logs
  - 14.2.7.1 Event Log Format
  - 14.2.7.2 Exporting Events in Real-Time
- 14.2.8 Enabling Syslog
- 14.2.9 Managing Log Verbosity

14.2.1 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. Currently, LOGLEVEL values between 0 and 9 are used to control the amount of information logged, with 0 being the most terse, logging only the most severe problems detected, while 9 is the most verbose, commenting on just about everything. 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. A LOGLEVEL of 2 will record virtually all critical messages, while a log level of 4 will provide general information describing all actions taken by the scheduler. 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.

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

```bash
# moab.cfg
# allow up to 30 100MB logfiles
LOGLEVEL        5
LOGDIR           /var/tmp/moab
LOGFILEMAXSIZE   100000000
```

The logging that occurs is of the following major types: subroutine information, status information, scheduler warnings, scheduler alerts, and scheduler errors.

14.2.2 Status Information

Critical internal status is indicated at low LOGLEVELs while less critical and more verbose status information is logged at higher LOGLEVELs. For example:

```
INFO: job orion.4228 rejected (max user jobs)
INFO: job fr4n01.923.0 rejected (maxjobperuser policy failure)
```

14.2.3 Scheduler Warnings

Warnings are logged when the scheduler detects an unexpected value or receives an unexpected result from a system call or subroutine. These messages are not necessarily indicative of problems and are not catastrophic to the scheduler. Most warnings are reported at loglevel 0 to loglevel 3. For example:

```
WARNING: cannot open fairshare data file '/opt/moab/stats/FS.87000'
```

14.2.4 Scheduler Alerts

Alerts are logged when the scheduler detects events of an unexpected nature that may indicate problems in other systems or in objects. They are typically of a more severe nature than warnings and possibly should be brought to the attention of scheduler administrators. Most alerts are reported at loglevel 0 to loglevel 2. For example:

```
ALERT: job orion.72 cannot run. deferring job for 360 Seconds
```

14.2.5 Scheduler Errors

Errors are logged when the scheduler detects problems of a nature that impacts the scheduler's ability to properly schedule the cluster. Moab will try to remedy or mitigate the problem as best it can, but the problem may be outside of its sphere of control. Errors should definitely be monitored by administrators. Most errors are reported at loglevel 0 to loglevel 1. For example:

```
ERROR: cannot connect to Loadleveler API
```

14.2.6 Searching Moab Logs

While major failures are reported via the `mdiat `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 should usually turn up empty. 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, \(<\text{LOGLEVEL}\> \) values from 0 to 9 are supported.

The \(\text{LOGLEVEL}\) can be changed dynamically by use of the \texttt{mschedctl -m} command, or by modifying the \texttt{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 \texttt{SIGUSR1} signal to the scheduler process. Repeated \texttt{SIGUSR1} signals continue to increase the log level. The \texttt{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.

### 14.2.7 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 \texttt{events.WWW_MMM_DD_YYYY} as in \texttt{events.Tue_Mar_18_2008}.

#### 14.2.7.1 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:

\[
\text{<EVENTTIME> <EPOCHTIME>:<EID> <OBJECT> <OBJECTID> <EVENT> <DETAILS>}
\]

**Example**

```
07:03:21 110244322:0 sched clusterA   start
07:03:26 110244327:1 rsv  system.1  start 1124142432 1324142432 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 \texttt{RECORDEVENTLIST} can be used to control which events are reported to the event log. See the sections on \texttt{job} and \texttt{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:

<table>
<thead>
<tr>
<th>Record Type</th>
<th>Event Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{gevent}</td>
<td>See \texttt{Enabling Generic Events} for \texttt{gevent} information.</td>
<td></td>
</tr>
</tbody>
</table>

Generic events are included within \texttt{node} records. See \texttt{node} detail format that follows.

| \texttt{job} | JOBCANCEL, JOBCHECKPOINT, JOBEND, JOBHOLD, JOBMIGRATE, JOBMODIFY, JOBPREEMPT, JOREJECT, JOBRESUME, JOBSTART, JOBSUBMIT | See \texttt{Workload Accounting Records}. |

<table>
<thead>
<tr>
<th>\texttt{node}</th>
<th>NODEDOWN, NODEFAILURE, NODEUP</th>
<th>\texttt{&lt;state&gt; &lt;partition&gt; &lt;disk&gt; &lt;memory&gt; &lt;maxprocs&gt; &lt;swap&gt; &lt;os&gt; &lt;rm&gt; &lt;nodeaccesspolicy&gt; &lt;class&gt; &lt;message&gt;}, where \texttt{&lt;state&gt;} is the node’s current state and \texttt{&lt;message&gt;} is a human readable message</th>
</tr>
</thead>
<tbody>
<tr>
<td>rm</td>
<td>RMDOWN, RMPOLLEND, RMPOLLSTART, RMUP</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>Human readable message indicating reason for resource manager state change.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>![info_icon]</td>
<td>RMUP and RMDOWN are only logged for PBS resource managers.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rsv</th>
<th>RSVCANCEL, RSVCREATE, RSVEND, RSVMODIFY, RSVSTART</th>
</tr>
</thead>
<tbody>
<tr>
<td>creationtime - epoch</td>
<td></td>
</tr>
<tr>
<td>starttime - epoch</td>
<td></td>
</tr>
<tr>
<td>endtime - epoch</td>
<td></td>
</tr>
<tr>
<td>alloc taskcount - integer</td>
<td></td>
</tr>
<tr>
<td>alloc nodecount - integer</td>
<td></td>
</tr>
<tr>
<td>total active proc-seconds - integer</td>
<td></td>
</tr>
<tr>
<td>total proc-seconds - integer</td>
<td></td>
</tr>
<tr>
<td>hostlist - comma-delimited list</td>
<td></td>
</tr>
<tr>
<td>owner - reservation owner</td>
<td></td>
</tr>
<tr>
<td>ACL - semicolon-delimited access control list</td>
<td></td>
</tr>
<tr>
<td>category - reservation usage category</td>
<td></td>
</tr>
<tr>
<td>comment - human-readable description</td>
<td></td>
</tr>
<tr>
<td>command - &lt;command&gt; &lt;argument(s)&gt;</td>
<td></td>
</tr>
<tr>
<td>human readable message - MSG=''&lt;message&gt;'</td>
<td></td>
</tr>
<tr>
<td>(See Reservation Accounting Records.)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sched</th>
<th>ALLSCHEDCOMMAND, SCHEDCOMMAND, SCHEDCYCLEEND, SCHEDCYCLESTART, SCHEDFAILURE, SCHEDMODIFY, SCHEDPAUSE, SCHEDRECYCLE, SCHEDRESUME, SCHEDSTART, SCHEDSTOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human readable message indicating reason for scheduler action.</td>
<td></td>
</tr>
<tr>
<td>![info_icon]</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>trigger</th>
<th>TRIGEND, TRIGFAILURE, TRIGSTART</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ATTR&gt;=&quot;&lt;VALUE&gt;&quot;[ &lt;ATTR&gt;=&quot;&lt;VALUE&gt;&quot;]...</td>
<td></td>
</tr>
<tr>
<td>where &lt;ATTR&gt; 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.</td>
<td></td>
</tr>
<tr>
<td>See Object Trigger Overview for more information.</td>
<td></td>
</tr>
</tbody>
</table>

### 14.2.7.2 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** directory included with the Moab distribution contains **event.create.sql.pl**, a sample
14.2.8 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

14.2.9 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.

See Also

- RECORDEVENTLIST parameter
- USESYSLOG parameter
- Notifying Admins
- Simulation Workload Trace Overview
- mscedctl -L command
14.3 Object Messages

14.3.1 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.

14.3.2 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.

14.3.2 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

14.3.2 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.
14.4 Notifying Administrators of Failures

14.4.1 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.

14.4.2 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:

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOBCORRUPTION</td>
<td>&lt;MESSAGE&gt;</td>
<td>An active job is in an unexpected state or has one or more allocated nodes that are in unexpected states.</td>
</tr>
<tr>
<td>JOBHOLD</td>
<td>&lt;MESSAGE&gt;</td>
<td>A job hold has been placed on a job.</td>
</tr>
<tr>
<td>JOBWCVIOLATION</td>
<td>&lt;MESSAGE&gt;</td>
<td>A job has exceeded its wallclock limit.</td>
</tr>
<tr>
<td>RESERVATIONCORRUPTION</td>
<td>&lt;MESSAGE&gt;</td>
<td>Reservation corruption has been detected.</td>
</tr>
<tr>
<td>RESERVATIONCREATED</td>
<td>&lt;RSVNAME&gt; &lt;RSVTYPE&gt; &lt;NAME&gt; &lt;PRESENTTIME&gt; STARTTIME &lt;ENDTIME&gt; &lt;NODECOUNT&gt;</td>
<td>A new reservation has been created.</td>
</tr>
<tr>
<td>RESERVATIONDESTROYED</td>
<td>&lt;RSVNAME&gt; &lt;RSVTYPE&gt; &lt;PRESENTTIME&gt; &lt;STARTTIME&gt; &lt;ENDTIME&gt; &lt;NODECOUNT&gt;</td>
<td>A reservation has been destroyed.</td>
</tr>
<tr>
<td>RMFAILURE</td>
<td>&lt;MESSAGE&gt;</td>
<td>The interface to the resource manager has failed.</td>
</tr>
</tbody>
</table>

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>);

See Also

- JOBREJECTPOLICY parameter
- MAILPROGRAM parameter
- Event Log Overview
14.5 Issues with Client Commands

- 14.5.1 Client Overview
- 14.5.2 Diagnosing Client Problems

14.5.1 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.

14.5.2 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=7
```

If you're using a larger system, use `-L7` instead. This creates a temporary log which will not overload the large system over time.

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.

If these steps do not reveal the source of the problem, the next steps may involve one or more of the following:

- Check with Adaptive Computing Enterprises, Inc.
- Search the online documentation.
- Search the online knowledge base inside the customer portal.
14.6 Tracking System Failures

14.6.1 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.

14.6.1.1 Disk Space

The scheduler uses a number of files. If the file system is full or otherwise inaccessible, the following behaviors might be noted:

<table>
<thead>
<tr>
<th>Unavailable File</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>moab.pid</td>
<td>Scheduler cannot perform <em>single instance</em> check.</td>
</tr>
<tr>
<td>moab.ck*</td>
<td>Scheduler cannot store persistent record of reservations, jobs, policies, summary statistics, and so forth.</td>
</tr>
<tr>
<td>moab.cfg/moab.dat</td>
<td>Scheduler cannot load local configuration.</td>
</tr>
<tr>
<td>log/*</td>
<td>Scheduler cannot log activities.</td>
</tr>
<tr>
<td>stats/*</td>
<td>Scheduler cannot write job records.</td>
</tr>
</tbody>
</table>

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.

14.6.1.2 Network

The scheduler uses a number of socket connections to perform basic functions. Network failures may affect the following facilities.

<table>
<thead>
<tr>
<th>Network Connection</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheduler client</td>
<td>Scheduler client commands fail.</td>
</tr>
<tr>
<td>resource manager</td>
<td>Scheduler is unable to load/update information regarding nodes and jobs.</td>
</tr>
<tr>
<td>allocation manager</td>
<td>Scheduler is unable to validate account access or reserve/debit account balances.</td>
</tr>
</tbody>
</table>

14.6.1.3 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.

14.6.1.4 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.

14.6.2 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.

### 14.6.2.1 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.

### 14.6.3 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://support.adaptivecomputing.com/

If you do not already have a support username and password, please send an e-mail message to info@adaptivecomputing.com to request one.
14.7 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.

- **-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.

**See Also**

- Diagnosing System Behavior/Problems
14.8 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 Cluster Resources support staff. This section introduces available diagnostic scripts.

14.8.1 The support.diag.pl Script

The `tools/moab/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`, `mdig -n`, and `mdig -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 Cluster Resources 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 Cluster Resources 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--include-log-lines=&lt;NUM&gt;</td>
<td>Instead of including the entire <code>moab.log</code> file, only the last <code>&lt;NUM&gt;</code> lines are captured in the diagnostics.</td>
</tr>
<tr>
<td>--diag-torque</td>
<td>Diagnostic commands pertinent to the TORQUE resource manager are included.</td>
</tr>
<tr>
<td>--no-upload</td>
<td>Prevents the system from asking the user if they want to upload the tarball to Adaptive Computing Customer Support.</td>
</tr>
<tr>
<td>--support-ticket=&lt;SUPPORT_TICKET_ID&gt;</td>
<td>Prevents the system from asking the user for a support ticket number.</td>
</tr>
</tbody>
</table>
15.0 Improving User Effectiveness

- 15.1 User Feedback Loops
- 15.2 User Level Statistics
- 15.3 Job Start Time Estimates
- 15.4 Collecting Performance Information on Individual Jobs
15.1 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.

15.1.1 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.

15.1.2 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.
15.2 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 % PSTgt 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 wall-clock estimates.

Moab must use an ODBC-compliant database to report statistics with Viewpoint reports.
15.3 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:

```plaintext
> showstart orion.13762
job orion.13762 requires 2 procs for 0:33:20
Estimated Rsv based start in 12:53:40
Estimated Rsv based completion in 14:33:40
Estimated Priority based start in 17:03:40
Estimated Priority based completion in 18:43:40
Estimated Historical based start in 11:48:45
Estimated Historical based completion in 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`.

**See Also**

- `ENABLESTARTESTIMATESTATS` parameter
- `showstart` command
15.4 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.
16.0 Cluster Analysis, Testing, and Simulation

- 16.1 Evaluating New Releases and Policies
- 16.2 Testing New Middleware
- 16.3 Simulation Overview

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.

Simulations allow organizations to evaluate many scenarios that could not be properly evaluated in real-world situations. In particular, these evaluations may be impossible due to time constraints, budgetary or personnel limitations, hardware availability, or even policy issues. In such cases, simulations provide information in countless scenarios and can help answer questions such as the following:

- What is the impact of additional hardware on cluster utilization?
- What delays to key projects can be expected with the addition of new users?
- How will new prioritization weights alter cycle distribution among existing workload?
- What total loss of compute resources will result from introducing a maintenance downtime?
- Are the benefits of cycle stealing from non-dedicated desktop systems worth the effort?
- How much will anticipated grid workload delay the average wait time of local jobs?
16.1 Testing New Releases and Policies

- 16.1.1 Moab Evaluation Modes
  - 16.1.1.1 MONITOR Mode
  - 16.1.1.2 TEST Mode
  - 16.1.1.3 INTERACTIVE Mode
- 16.1.2 Testing New Releases
- 16.1.3 Testing New Policies
  - 16.1.3.1 Verifying Correct Specification of New Policies
  - 16.1.3.2 Verifying Correct Behavior of New Policies
  - 16.1.3.3 Determining Long Term Impact of New Policies
- 16.1.4 Moab Side-by-Side

16.1.1 Moab Evaluation Modes

16.1.1.1 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.

16.1.1.2 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.

16.1.1.3 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
```
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.

### 16.1.2 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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdiag -n</td>
<td>Compute nodes, storage systems, network systems, and generic resources</td>
</tr>
<tr>
<td>mdiag -j</td>
<td>Applications, dynamic and static jobs</td>
</tr>
<tr>
<td>mdiag -u</td>
<td>User, group, and account credentials</td>
</tr>
<tr>
<td>mdiag -g</td>
<td></td>
</tr>
<tr>
<td>mdiag -a</td>
<td></td>
</tr>
<tr>
<td>mdiag -c</td>
<td>Queues and policies</td>
</tr>
<tr>
<td>mdiag -R</td>
<td>Resource manager interface and performance</td>
</tr>
<tr>
<td>mdiag -S</td>
<td>Scheduler/system level failures introduced by corrupt information</td>
</tr>
</tbody>
</table>

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.

### 16.1.3 Testing New Policies

#### 16.1.3.1 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:
This command will validate the configuration file and report any misconfiguration.

### 16.1.3.2 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**.

### 16.1.3.3 Determining Long Term Impact of New Policies

If a new policy has the potential to impact long-term performance or resource distribution, it may be desirable to run a Moab **simulation** to evaluate this change. Simulations allow locally recorded workload to be translated into simulation jobs and execute on a virtual cluster that emulates local resources. Simulations import all job and resource attributes that are loaded in a production environment as well as all policies specified in any configuration file. While running, all Moab commands and statistics are fully supported.

Using simulation, a control run can be made using the original policies and the behavior of this run compared to a second run that contains the specified change. Moab Cluster Manager's charting, graphing, and reporting features can be used to report on and visualize the differences in these two runs. Typically, a two-month real-time simulation can be completed in under an hour. For more information on simulations, see the [Simulation Overview](#).

### 16.1.4 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.

```plaintext
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
...
```

In the previous example, 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.

**Tip:** 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.

**See Also**

- Testing New Versions and Configurations
16.2 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.

<table>
<thead>
<tr>
<th>System</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation Manager</td>
<td>Use <strong>test</strong> or <strong>simulation</strong> mode to drive scheduling queries, allocation debits, and reservations to accounting packages. Verify synchronization of cluster statistics and stress test interfaces and underlying databases. (Set environment variable <strong>MOABAMTEST=yes</strong> to enable.)</td>
</tr>
<tr>
<td>On-Demand/Provisioning Services</td>
<td>Use <strong>simulation</strong> or <strong>native resource manager</strong> 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.</td>
</tr>
<tr>
<td>Resource Monitoring</td>
<td>Use <strong>test</strong> or <strong>native resource manager</strong> mode to actively load information from compute, network, storage, and software license managers confirming validity of data, availability during failures, and scalability.</td>
</tr>
</tbody>
</table>

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:

```plaintext
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:

```plaintext
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.
16.3.0 Simulations

- 16.3.1 Simulation Overview
- 16.3.2 Resource Traces
- 16.3.3 Workload Traces
- 16.3.4 Simulation Specific Configuration

Simulations allow organizations to evaluate many scenarios that could not be properly evaluated in the real world. In particular, these evaluations may be impossible due to time constraints, budgetary or man-power limitations, hardware availability, or may even be impossible due to policy issues.

In such cases, simulation can help answer questions in countless scenarios and provide information such as the following:

- What is the impact of additional hardware on cluster utilization?
- What delays to key projects can be expected with the addition of new users?
- How will new prioritization weights alter cycle distribution among existing workload?
- What total loss of compute resources will result from introducing a maintenance downtime?
- Are the benefits of cycle stealing from non-dedicated desktop systems worth the effort?
- How much will anticipated grid workload delay the average wait time of local jobs?
16.3.1 Simulation Overview

This section explains the following concepts:

- Value of simulations
- Specifying resources and workloads for simulation
- Where to specify policies
- Changing moab.cfg for simulation
- Starting a simulation
- Queue Status
- Job status
- Iteration control
- Determining why jobs are not running
- Dynamically changing parameters
- Reservations applying to the queue
- Fair scheduling
- System for maintenance

16.3.1.1 Determining Performance Metrics

The first step of most simulations is to determine the primary purpose of the simulation. Purposes may include identifying impact of certain resource or workload changes on current cluster performance. Simulations may also focus on system utilization or workload distribution across resources or credentials. Further, simulations may also be used for training purposes, allowing risk-free evaluation of behavior, facilities, and commands. With the purpose known, metrics of success can be specified and a proper simulation created. While performance metrics may not be critical to training based simulations, they are key to successful evaluation in most other cases.

16.3.1.2 Selecting Resources

As in the real world, a simulation requires a set of resources (compute hosts) on which to run. In Moab, this is specified using a resource trace file. This resource trace file may be obtained from specific hardware or generated for the specific purpose.

16.3.1.3 Selecting Workload

In addition to resources, a simulation also requires a workload (batch jobs) to schedule onto the available resources. This workload is specified within a workload trace file. Like the resource traces, this workload information may be based on recorded data or generated to meet the need of the particular simulation.

16.3.1.4 Selecting Policies

The final aspect of a simulation is the set of policies and configuration to be used to determine how a workload is to be scheduled onto the available resources. This configuration is placed in the moab.cfg file just as would be done in production (or normal) mode operation.

16.3.1.5 Initial Configuration Using the Sample Traces

While mastering simulations may take some time, initial configuration is straightforward. To start, edit the moab.cfg file and do the following:

- Change the SCHEDCFG attribute MODE from NORMAL or MONITOR to SIMULATION.
- Add the following lines:

```
SIMRESOURCETRACEFILE  samples/resource.testcluster.txt
SIMWORKLOADTRACEFILE  samples/workload.testcluster.txt
SIMSTOPITERATION  0
```
The preceding steps specify that the scheduler should run in simulation mode and use the referenced resource and workload trace files. In addition, leaving the `SIMSTOPITERATION` parameter at zero indicates that Moab should stop before the first scheduling iteration and wait for further instructions. If you want the simulation to run as soon as you start Moab, remove (or comment out) this line. To continue scheduling, run the `mschedctl -r` command.

- You also may need to add these lines to the `moab.cfg` file:

```plaintext
CREDDISCOVERY           TRUE
SIMAUTOSHUTDOWN         false
SIMSTARTTIME            1196987696
USERCFG[DEFAULT] ENABLEPROFILING=true
GROUPCFG[DEFAULT] ENABLEPROFILING=true
ACCOUNTCFG[DEFAULT] ENABLEPROFILING=true
CLASSCFG[DEFAULT] ENABLEPROFILING=true
QOSCFG[DEFAULT] ENABLEPROFILING=true
```

The second set of parameters is helpful if you want to generate charts or reports from Moab Cluster Manager. Since events in the workload trace may reference credentials that are not listed in your `moab.cfg` file, set `CREDDISCOVERY` to true, which allows Moab to create simulated credentials for credentials that do not yet exist. Setting `SIMAUTOSHUTDOWN` to false prevents Moab from terminating after it has finished running all the jobs in the workload trace, and it allows you to generate charts after all the simulated jobs have finished. Ensure that `SIMSTARTTIME` is set to the epoch time (in seconds) of the first event in your workload trace file. This causes the internal clock in Moab to be set to the workload trace's first event, which prevents issues caused by the difference between the time the workload trace was created and the time reported by the CPU clock. Otherwise, Moab thinks the current time is the time that the CPU clock reports, yet simulated jobs that are reported by `showq` as currently running will really be running at the time the workload trace was created. To avoid confusion, set the `SIMSTARTTIME`. The lines that specify `ENABLEPROFILING=true` are necessary for Moab to keep track of the statistics generated by the simulated jobs. Not setting these lines will cause charts and reports to contain all zero values.

### 16.3.1.6 Starting a Simulation

As in all cases, Moab should be started by issuing the command `moab`. It should be noted that in simulation mode, Moab does not daemonize itself and so will not background itself. Verification of proper operation is possible using any common user command such as `showq`. If the `showq` command is run, it will display the number of jobs currently in the scheduler's queue. The jobs displayed by the `showq` command are taken from the workload trace file specified earlier and those that are marked as running are running on resources described in the resource trace file. At any point, a detailed summary of available resources may be obtained by running the `mdiag -n` command.

### 16.3.1.7 Interactive Tutorial

The rest of this section provides an interactive tutorial to demonstrate the basics of the simulator's capacities in Moab. The commands to issue are formatted as follows: `> showq` along with the expected output.

The following commands are used:

- `showq [-r] [-i]
- showstats [-g] [-u] [-v]
- mschedctl -l
- mschedctl [{-s|-S} [I]] [-k]
- checkjob
- mschedctl -m
- mdiag -n
- showres [-n jobid]
- setres

Start by running Moab:
Next, verify that Moab is running by executing `showq`:

```bash
> showq
```

<table>
<thead>
<tr>
<th>JOBNAME</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROC</th>
<th>REMAINING</th>
</tr>
</thead>
<tbody>
<tr>
<td>fr8n01.187.0</td>
<td>570</td>
<td>Running</td>
<td>20</td>
<td>1:00:00:00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mon Feb 16 11:54:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr8n01.189.0</td>
<td>570</td>
<td>Running</td>
<td>20</td>
<td>1:00:00:00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mon Feb 16 11:54:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr8n01.190.0</td>
<td>570</td>
<td>Running</td>
<td>20</td>
<td>1:00:00:00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mon Feb 16 11:54:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr8n01.191.0</td>
<td>570</td>
<td>Running</td>
<td>20</td>
<td>1:00:00:00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mon Feb 16 11:54:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr8n01.276.0</td>
<td>550</td>
<td>Running</td>
<td>20</td>
<td>1:00:00:00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mon Feb 16 11:54:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr1n04.369.0</td>
<td>550</td>
<td>Running</td>
<td>20</td>
<td>1:00:00:00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mon Feb 16 11:54:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr1n04.487.0</td>
<td>550</td>
<td>Running</td>
<td>20</td>
<td>1:00:00:00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mon Feb 16 11:54:03</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7 active jobs  140 of 196 Processors Active (71.43%)  

<table>
<thead>
<tr>
<th>JOBNAME</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROC</th>
<th>WCLIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>fr1n04.362.0</td>
<td>550</td>
<td>Idle</td>
<td>20</td>
<td>1:00:00:00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mon Feb 16 11:53:33</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Out of the thousands of jobs in the workload trace, only 16 jobs are either active or eligible because of the default settings of the `SIMINITIALQUEUEDEPHT` parameter. Sixteen jobs are put in the idle queue, seven of which immediately run. Issuing the command `showq -r` allows a more detailed look at the active (or running) jobs. The output is sorted by job completion time and indicates that the first job will complete in one day (1:00:00:00).

While `showq` details information about the queues, scheduler statistics may be viewed using the `showstats` command. The field `Current Active/Total Procs` shows current system utilization, for example.

```bash
> showstats
```

```
moab active for 00:00:30 stats initialized on Mon Feb 16 11:53:33
Eligible/Idle Jobs: 9/9 (100.000%)
Active Jobs: 0
Successful/Completed Jobs: 0/0 (0.000%)
Avg/Max QTime (Hours): 0.00/0.00
Avg/Max XFactor: 0.00/0.00
Dedicated/Total ProcHours: 1.17/1.63 (71.429%)
Current Active/Total Procs: 140/196 (71.429%)
Avg WallClock Accuracy: N/A
Avg Job Proc Efficiency: N/A
Est/Avg Backlog (Hours): N/A / N/A
```

You might be wondering why there are only 140 of 196 Processors Active (as shown with `showq`) when
the first job (fr1n04.362.0) in the queue only requires 20 processors. We will use the `checkjob` command, which reports detailed job state information and diagnostic output for a particular job to determine why it is not running:

```bash
> checkjob fr1n04.362.0
job fr1n04.362.0
  State: Idle
  ...
  Network: hps_user Memory >= 256M Disk >= 0 Swap >= 0
  ...
  Job Eligibility Analysis -------
  job cannot run in partition DEFAULT (idle procs do not meet requirements : 8 of 20 procs found)
  idle procs:  56  feasible procs:   8
  Rejection Reasons: [Memory : 48][State : 140]
```

`Checkjob` not only tells us the job's wallclock limit and the number of requested nodes (they're in the ellipsis) but explains why the job was rejected from running. The `Job Eligibility Analysis` tells us that 48 of the processors rejected this job due to memory limitations and that another 140 processors rejected it because of their state (that is, they're running other jobs). Notice the \( \geq 256 \text{ M(B)} \) memory requirement.

If you run `checkjob` with the ID of a running job, it would also tell us exactly which nodes have been allocated to this job. There is additional information that the `checkjob` command page describes in more detail.

Advancing the simulator an iteration, the following happens:

```bash
> mschedctl -S
scheduling will stop in 00:00:30 at iteration 1
```

The scheduler control command, `mschedctl`, controls various aspects of scheduling behavior. It can be used to manage scheduling activity, kill the scheduler, and create resource trace files. The `-S` argument indicates that the scheduler run for a single iteration and stop. Specifying a number, \( n \), after `-S` causes the simulator to advance \( n \) steps. You can determine what iteration you are currently on using `showstats -v`.

```bash
> showstats -v
  current scheduler time: Mon Feb 16 11:54:03 1998 (887655243)
  moab active for 00:01:00 stats initialized on Mon Feb 16 11:53:33
  statistics for iteration 1 scheduler started on Wed Dec 31 17:00:00
  ...
```

The line that starts with `statistics for iteration <X>` specifies the iteration you are currently on. Each iteration advances the simulator `RMPOLLINTERVAL` seconds. To see what `RMPOLLINTERVAL` is set to, use the `showconfig` command:

```bash
> showconfig | grep RMPOLLINTERVAL
  RMPOLLINTERVAL 00:00:30
```

By default, `RMPOLLINTERVAL` is set to 30 seconds. Leaving the `RMPOLLINTERVAL` at the default will match your actual iteration period, resulting in realtime simulation. For example, increasing this number by a factor of 5 will run the simulations 5 times faster. With `showconfig`, you can see the current value of all configurable parameters.
The `showq -r` command can be used to display the running (active) jobs to see what happened in the last iteration:

```
> showq -r
active jobs------------------------
JOBID       MHOST  PROCS  REMAINING  S  PAR  EFFIC  XFACTOR  Q      USER    GROUP
fr8n01.804.0        R   1 ------      1.0  -       529      519
fr9n16     5 00:05:00  Mon Feb 16 11:54:03
fr8n01.187.0        R   1 ------      1.0  -       570      519
fr7n15  20 1:00:00:00  Mon Feb 16 11:54:03
...
fr8n01.960.0        R   1 ------      1.0  -       588      519
fr9n11  32 1:00:00:00  Mon Feb 16 11:54:03

9 active jobs     177 of  196 Processors Active (90.31%)
Total jobs:  9
```

Notice that two new jobs started (without waiting in the eligible queue). Also notice that job fr8n01.187.0, along with the rest that are summarized in the ellipsis, did NOT advance its `REMAINING` or `STARTTIME`. The simulator needs one iteration to do a sanity check. Setting the parameter `SIMSTOPITERATION` to 1 causes Moab to stop after the first scheduling iteration and wait for further instructions.

The `showq -i` command displays the idle (eligible) jobs.

```
> showq -i
eligible jobs----------------------
JOBID       WCLIMIT  CLASS  PRIORITY  XFACTOR  Q      USER    GROUP  PROCS
fr1n04.362.0* 1:00:00:00  batch  Mon Feb 16 11:53:33
fr1n04.363.0 1:00:00:00  batch  Mon Feb 16 11:53:33
fr1n04.365.0 1:00:00:00  batch  Mon Feb 16 11:53:33
fr1n04.366.0 1:00:00:00  batch  Mon Feb 16 11:53:33
fr1n04.501.0 1:00:00:00  batch  Mon Feb 16 11:53:33
fr1n04.580.0 1:00:00:00  batch  Mon Feb 16 11:53:33
fr1n04.597.0 1:00:00:00  batch  Mon Feb 16 11:53:33
fr1n04.598.0 1:00:00:00  batch  Mon Feb 16 11:53:33
fr1n04.602.0 1:00:00:00  batch  Mon Feb 16 11:53:33
fr1n04.743.0 1:00:00:00  batch  Mon Feb 16 11:54:03
fr1n04.744.0 1:00:00:00  batch  Mon Feb 16 11:54:03
fr1n04.746.0 1:00:00:00  batch  Mon Feb 16 11:54:03
fr1n04.747.0 1:00:00:00  batch  Mon Feb 16 11:54:03
fr1n04.748.0 1:00:00:00  batch  Mon Feb 16 11:54:03
fr1n04.749.0 1:00:00:00  batch  Mon Feb 16 11:54:03
fr1n04.750.0 1:00:00:00  batch  Mon Feb 16 11:54:03
fr1n04.751.0 1:00:00:00  batch  Mon Feb 16 11:54:03

Notice how none of the eligible jobs are requesting 19 or fewer jobs (the number of idle processors). Also notice the `*` after the job id fr1n04.362.0. This means that this job now has a reservation. The `showres` command shows all reservations currently on the system.

```
> showres
```
Here, the $S$ column is the job's state (R = running, I = idle). All the active jobs have a reservation along with idle job fr1n04.362.0. This reservation was actually created by the backfill scheduler for the highest priority idle job as a way to prevent starvation while lower priority jobs were being backfilled. (The backfill documentation describes the mechanics of the backfill scheduling more fully.)

To display information about the nodes that job fr1n04.362.0 has reserved, use `showres -n <JOBID>`.

```
ReservationID       Type S Start         End    Duration    N/P
StartTime
fr8n01.187.0         Job R    00:00:00  1:00:00:00  1:00:00:00   20/20
Mon Feb 16 11:54:03
fr8n01.189.0         Job R    00:00:00  1:00:00:00  1:00:00:00   20/20
Mon Feb 16 11:54:03
fr8n01.190.0         Job R    00:00:00  1:00:00:00  1:00:00:00   20/20
Mon Feb 16 11:54:03
fr8n01.191.0         Job R    00:00:00  1:00:00:00  1:00:00:00   20/20
Mon Feb 16 11:54:03
fr8n01.276.0         Job R    00:00:00  1:00:00:00  1:00:00:00   20/20
Mon Feb 16 11:54:03
fr1n04.362.0         Job I  1:00:00:00  2:00:00:00  1:00:00:00   20/20
Tue Feb 17 11:54:03
fr1n04.369.0         Job R    00:00:00  1:00:00:00  1:00:00:00   20/20
Mon Feb 16 11:54:03
fr1n04.487.0         Job R    00:00:00  1:00:00:00  1:00:00:00   20/20
Mon Feb 16 11:54:03
fr8n01.804.0         Job R    00:00:00    00:05:00    00:05:00    5/5
Mon Feb 16 11:54:03
fr8n01.960.0         Job R    00:00:00  1:00:00:00  1:00:00:00   32/32
Mon Feb 16 11:54:03
10 reservations located
```

Now advance the simulator an iteration to allow some jobs to actually run.

```
> mschedctl -S
scheduling will stop in 00:00:30 at iteration 2
```

Next, check the queues to see what happened.

```
> showq
active jobs------------------------
JOBNAME            USERNAME      STATE  PROC   REMAINING
STARTTIME
```
Two new jobs, fr8n01.963.0 and fr8n01.1016.0, are in the eligible queue. Also, note that the first job will now complete in 4 minutes 30 seconds rather than 5 minutes because we have just advanced now by 30 seconds, one RMPOLLINTERVAL. It is important to note that when the simulated jobs were created, both the job’s wallclock limit and its actual run time were recorded. The wallclock limit is specified by the user indicating their best estimate of an upper bound on how long the job will run. The run time is how long the job actually ran before completing and releasing its allocated resources. For example, a job with a wallclock limit of 1 hour will be given the needed resources for up to an hour but may complete in only 20 minutes.

Stop the simulation at iteration 6.

> mschedctl -s 6I
scheduling will stop in 00:03:00 at iteration 6

The $-s\ 6I$ argument indicates that the scheduler will stop at iteration 6 and will (I)gnore user input until it gets there. This prevents the possibility of obtaining showq output from iteration 5 rather than iteration 6.

Job fr8n01.804.0 is still 2 minutes 30 seconds away from completing as expected but notice that both jobs fr8n01.189.0 and fr8n01.191.0 have completed early. Although they had almost 24 hours remaining of wallclock limit, they terminated. In reality, they probably failed on the real world system where the trace file was being created. Their completion freed up 40 processors which the scheduler was able to immediately use by starting several more jobs.
Note the system statistics:

```
> showstats
...
Successful/Completed Jobs: 0/2 (0.000%)
...
Avg WallClock Accuracy: 0.150%
Avg Job Proc Efficiency: 100.000%
Est/Avg Backlog (Hours): 0.00/3652178.74
```

A few more fields are filled in now that some jobs have completed providing information on which to generate statistics.

Decrease the default LOGLEVEL with `mschedctl -m` to avoid unnecessary logging, and speed up the simulation.

```
> mschedctl -m LOGLEVEL 0
INFO: parameter modified
```

You can use `mschedctl -m` to immediately change the value of any parameter. The change is only made to the currently running Moab server and is not propagated to the configuration file. Changes can also be made by modifying the configuration file and restarting the scheduler.

Stop at iteration 580 and pull up the scheduler's statistics.

```
> mschedctl -s 580I; showq
scheduling will stop in 4:47:00 at iteration 580
...
   11 active jobs    156 of 196 Processors Active (79.59%)
eligible jobs----------------------
JOBNAME     USERNAME     STATE  PROC     WCLIMIT       QUEUETIME
fr8n01.963.0 586       Idle    32     9:00:00  Mon Feb 16 11:54:33
fr8n01.1075.0 560       Idle    32    23:56:00  Mon Feb 16 11:58:33
fr8n01.1076.0 560       Idle    16    23:56:00  Mon Feb 16 11:59:33
frn04.1953.0 520       Idle    46     7:45:00  Mon Feb 16 12:03:03
...
16 eligible jobs...
```

You may note that `showq` hangs a while as the scheduler simulates up to iteration 580. The output shows that currently only 156 of the 196 nodes are busy, yet at first glance 3 jobs, `fr8n01.963.0`, `fr8n01.1075.0`, and `fr8n01.1076.0` appear to be ready to run.

```
> checkjob fr8n01.963.0; checkjob fr8n01.1075.0; checkjob fr8n01.1076.0
job fr8n01.963.0
...
Network: hps_user Memory >= 256M Disk >= 0 Swap >= 0
...
The `checkjob` command reveals that job `fr8n01.963.0` only found 20 of 32 processors. The remaining 20 idle processors could not be used because the configured memory on the node did not meet the jobs' requirements. The other jobs cannot find enough nodes because of ReserveTime. This indicates that the processors are idle, but that they have a reservation in place that will start before the job being checked could complete.

Verify that the idle nodes do not have enough memory configured and they are already reserved with the `mdiag -n` command, which provides detailed information about the state of nodes Moab is currently tracking. The `mdiag` command can be used with various flags to obtain detailed information about accounts, fair share, groups, jobs, nodes, QoS, queues, reservations, the resource manager, and users. The command also performs a number of sanity checks on the data provided and will present warning messages if discrepancies are detected.

```
> mdiag -n -v | grep -e Name -e Idle
```

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>Procs</th>
<th>Memory</th>
<th>Disk</th>
<th>Swap</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opsys</td>
<td>Arch</td>
<td>Par</td>
<td>Load</td>
<td>Rsv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr10n09</td>
<td>Idle</td>
<td>1:1</td>
<td>256:256</td>
<td>9780:9780</td>
<td>411488:411488</td>
<td>1.00</td>
</tr>
<tr>
<td>AIX43 R6000 DEF</td>
<td>0.00</td>
<td>001 .</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr10n11</td>
<td>Idle</td>
<td>1:1</td>
<td>256:256</td>
<td>8772:8772</td>
<td>425280:425280</td>
<td>1.00</td>
</tr>
<tr>
<td>AIX43 R6000 DEF</td>
<td>0.00</td>
<td>001 .</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr10n13</td>
<td>Idle</td>
<td>1:1</td>
<td>256:256</td>
<td>9272:9272</td>
<td>441124:441124</td>
<td>1.00</td>
</tr>
<tr>
<td>AIX43 R6000 DEF</td>
<td>0.00</td>
<td>001 .</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr10n15</td>
<td>Idle</td>
<td>1:1</td>
<td>256:256</td>
<td>8652:8652</td>
<td>440776:440776</td>
<td>1.00</td>
</tr>
<tr>
<td>AIX43 R6000 DEF</td>
<td>0.00</td>
<td>001 .</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr11n01</td>
<td>Idle</td>
<td>1:1</td>
<td>256:256</td>
<td>7668:7668</td>
<td>438624:438624</td>
<td>1.00</td>
</tr>
<tr>
<td>AIX43 R6000 DEF</td>
<td>0.00</td>
<td>001 .</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr11n03</td>
<td>Idle</td>
<td>1:1</td>
<td>256:256</td>
<td>9548:9548</td>
<td>424584:424584</td>
<td>1.00</td>
</tr>
<tr>
<td>AIX43 R6000 DEF</td>
<td>0.00</td>
<td>001 .</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr11n05</td>
<td>Idle</td>
<td>1:1</td>
<td>256:256</td>
<td>11608:11608</td>
<td>454476:454476</td>
<td>1.00</td>
</tr>
<tr>
<td>AIX43 R6000 DEF</td>
<td>0.00</td>
<td>001 .</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr11n07</td>
<td>Idle</td>
<td>1:1</td>
<td>256:256</td>
<td>9008:9008</td>
<td>425292:425292</td>
<td>1.00</td>
</tr>
<tr>
<td>AIX43 R6000 DEF</td>
<td>0.00</td>
<td>001 .</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr11n09</td>
<td>Idle</td>
<td>1:1</td>
<td>256:256</td>
<td>8588:8588</td>
<td>424684:424684</td>
<td>1.00</td>
</tr>
<tr>
<td>AIX43 R6000 DEF</td>
<td>0.00</td>
<td>001 .</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr11n11</td>
<td>Idle</td>
<td>1:1</td>
<td>256:256</td>
<td>9632:9632</td>
<td>424936:424936</td>
<td>1.00</td>
</tr>
<tr>
<td>AIX43 R6000 DEF</td>
<td>0.00</td>
<td>001 .</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr11n13</td>
<td>Idle</td>
<td>1:1</td>
<td>256:256</td>
<td>9524:9524</td>
<td>425432:425432</td>
<td>1.00</td>
</tr>
<tr>
<td>AIX43 R6000 DEF</td>
<td>0.00</td>
<td>001 .</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr11n15</td>
<td>Idle</td>
<td>1:1</td>
<td>256:256</td>
<td>9388:9388</td>
<td>425728:425728</td>
<td>1.00</td>
</tr>
<tr>
<td>AIX43 R6000 DEF</td>
<td>0.00</td>
<td>001 .</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The grep gets the command header and the idle nodes listed. All the idle nodes with 256 MB of memory installed already have a reservation. (See the Rsv column.) The rest of the idle nodes only have 128 MB of memory.

```
> checknode fr10n09
node fr10n09
State:      Idle  (in current state for 4:21:00)
Configured Resources:  PROCS: 1  MEM: 256M  SWAP: 401G  DISK: 9780M
Utilized Resources:  [NONE]
Dedicated Resources:  [NONE]
Total Time: 4:50:00  Up: 4:50:00 (100.00%)  Active: 00:34:30 (11.90%)
Reservations:
   Job 'fr8n01.963.0' (x1)  3:25:00 -> 12:25:00 (9:00:00)
```

Using `checknode` revealed that Job `fr8n01.963.0` has the reservation.

Moving ahead:

```
> mschedctl -S 500I; showstats -v
scheduling will stop in 4:10:00 at iteration 1080
...
Eligible/Idle Jobs: 16/16  (100.000%)
Active Jobs: 11
Successful/Completed Jobs: 2/25  (8.000%)
Preempt Jobs: 0
Avg/Max QTime (Hours): 0.00/0.00
Avg/Max XFactor: 0.00/1.04
Avg/Max Bypass: 0.00/13.00
Dedicated/Total ProcHours: 1545.44/1765.63  (87.529%)
Preempt/Dedicated ProcHours: 0.00/1545.44  (0.000%)
Current Active/Total Procs: 156/196  (79.592%)
Avg WallClock Accuracy: 9.960%
Avg Job Proc Efficiency: 100.000%
Min System Utilization: 79.592% (on iteration 33)
Est/Avg Backlog (Hours): 0.00/20289.84
```

We now know that the scheduler is scheduling efficiently. So far, system utilization as reported by `showstats -v` looks very good. An important and subjective question is whether the scheduler is scheduling fairly. Look at the user and group statistics to see if there are any glaring problems.

```
> showstats -u
statist...
Suppose you need to now take down the entire system for maintenance on Thursday from 2:00 to 8:00 a.m. To do this, create a reservation with `mrsvctl -c`.

```
> mrsvctl -c -t ALL -s 2:00_02/17 -d 6:00:00
```

Shut down the scheduler.

```
> mschedctl -k
moab will be shutdown immediately
```
16.3.2 Resource Traces

Resource traces fully describe all scheduling relevant aspects of a batch system's compute resources. In most cases, each resource trace describes a single compute node providing information about configured resources, node location, supported classes and queues, and so forth.

The `mnodectl -q wiki ALL` command will query all nodes and provide the proper output for the resource trace file from your current system.

Sample Resource Trace:

```
node01 STATE=Idle PARTITION=native AMEMORY=32000 APROC=4 CMEMORY=32000 CPROC=4 OS=linux GMETRIC[temp]=100.00 RM=native NODEACCESSPOLICY=SHARED CCLASS=[short:4]
```

```
node02 STATE=Idle PARTITION=native AMEMORY=32000 APROC=4 CMEMORY=32000 CPROC=4 OS=linux GMETRIC[temp]=100.00 RM=native NODEACCESSPOLICY=SHARED FEATURE=[bigmem] CCLASS=[short:4]
```

```
node03 STATE=Idle PARTITION=native AMEMORY=32000 APROC=4 CMEMORY=32000 CPROC=4 OS=linux GMETRIC[temp]=100.00 RM=native NODEACCESSPOLICY=SHARED FEATURE=[bigmem] CCLASS=[short:4]
```

```
node04 STATE=Idle PARTITION=native AMEMORY=32000 APROC=4 CMEMORY=32000 CPROC=4 OS=linux GMETRIC[temp]=100.00 RM=native NODEACCESSPOLICY=SHARED FEATURE=[bigmem] CCLASS=[short:4]
```

For more information, see Appendix W: Wiki Interface Specification, version 1.2.

See Also

- `SIMRESOURCETRACEFILE`
- `mnodectl -q wiki ALL`
16.3.3 Workload Accounting Records

Moab workload accounting records fully describe all scheduling relevant aspects of batch jobs including resources requested and used, time of all major scheduling events (such as submission time and start time), the job credentials used, and the job execution environment. Each job trace is composed of a single line consisting of whitespace delimited fields as shown in the following table.

Moab can be configured to provide this information in flat text tabular form or in XML format conforming to the SSS 1.0 job description specification.

- 16.3.3.1 Workload Event Record Format
- 16.3.3.2 Workload Event Record Format (v 6.0.0)
- 16.3.3.3 Creating New Workload Accounting Records/Traces
- 16.3.3.4 Reservation Records/Traces
- 16.3.3.5 Recording Job events

16.3.3.1 Workload Event Record Format (v 5.0.0)

All job events (JOBSUBMIT, JOBSTART, JOBEND, and so forth) provide job data in a standard format as described in the following table:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Index</th>
<th>Data Format</th>
<th>Default Value</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Time (Human Readable)</td>
<td>1</td>
<td>HH:MM:SS</td>
<td>-</td>
<td>Specifies time event occurred.</td>
</tr>
<tr>
<td>Event Time (Epoch)</td>
<td>2</td>
<td>&lt;epochtime&gt;</td>
<td>-</td>
<td>Specifies time event occurred.</td>
</tr>
<tr>
<td>Object Type</td>
<td>3</td>
<td>job</td>
<td>-</td>
<td>Specifies record object type.</td>
</tr>
<tr>
<td>Object ID</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>Unique object identifier.</td>
</tr>
<tr>
<td>Object Event</td>
<td>5</td>
<td>one of jobcancel, jobcheckpoint, jobend, jobfailure, jobhold, jobmigrate, jobpreempt, jobreject, jobresume, jobstart or jobsubmit</td>
<td>-</td>
<td>Specifies record event type.</td>
</tr>
<tr>
<td>Nodes Requested</td>
<td>6</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Number of nodes requested (0 = no node request count specified).</td>
</tr>
<tr>
<td>Tasks Requested</td>
<td>7</td>
<td>&lt;INTEGER&gt;</td>
<td>1</td>
<td>Number of tasks requested.</td>
</tr>
<tr>
<td>User Name</td>
<td>8</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Name of user submitting job.</td>
</tr>
<tr>
<td>Group Name</td>
<td>9</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Primary group of user submitting job.</td>
</tr>
<tr>
<td>Wallclock Limit</td>
<td>10</td>
<td>&lt;INTEGER&gt;</td>
<td>1</td>
<td>Maximum allowed job duration (in seconds).</td>
</tr>
<tr>
<td>Job Event State</td>
<td>11</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Job state at time of event.</td>
</tr>
<tr>
<td>Required Class</td>
<td>12</td>
<td>&lt;STRING&gt;</td>
<td>[DEFAULT:1]</td>
<td>Class/queue required by job specified as square bracket list of QUEUE[:QUEUEINSTANCE] requirements. (For example: [batch:1]).</td>
</tr>
<tr>
<td>Submission Time</td>
<td>13</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Epoch time when job was submitted.</td>
</tr>
<tr>
<td>Dispatch Time</td>
<td>14</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Epoch time when scheduler requested job begin executing.</td>
</tr>
<tr>
<td>Start Time</td>
<td>15</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Epoch time when job began executing. This is usually identical to Dispatch Time.</td>
</tr>
<tr>
<td>Completion Time</td>
<td>16</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Epoch time when job completed execution.</td>
</tr>
<tr>
<td>Required Network Adapter</td>
<td>17</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Name of required network adapter if specified.</td>
</tr>
<tr>
<td>Required Node Architecture</td>
<td>18</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Required node architecture if specified.</td>
</tr>
<tr>
<td>Operating System</td>
<td>19</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Required node operating system if specified.</td>
</tr>
<tr>
<td>Required Node</td>
<td></td>
<td>one of &gt;, &gt;=, =, &lt;=, &lt;</td>
<td>-</td>
<td>Comparison for determining compliance with</td>
</tr>
</tbody>
</table>

Field Name:  Field Index:  Data Format:  Default Value:  Details:
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Comparison</td>
<td>Required Node Memory</td>
<td>20</td>
<td>&gt;= required node memory.</td>
</tr>
<tr>
<td>Required Node Memory</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Amount of required configured RAM (in MB) on each node.</td>
</tr>
<tr>
<td>Required Node Disk Comparison</td>
<td>one of &gt;, &gt;=, =, &lt;=, &lt; &gt;=</td>
<td>Comparison for determining compliance with required node disk.</td>
<td></td>
</tr>
<tr>
<td>Required Node Disk</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Amount of required configured local disk (in MB) on each node.</td>
</tr>
<tr>
<td>Required Node Attributes/Features</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Square bracket enclosed list of node features required by job if specified. (For example: [fast][ethernet])</td>
</tr>
<tr>
<td>System Queue Time</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Epoch time when job met all fairness policies.</td>
</tr>
<tr>
<td>Tasks Allocated</td>
<td>&lt;INTEGER&gt;</td>
<td>&lt;TASKS_REQUESTED&gt;</td>
<td>Number of tasks actually allocated to job.</td>
</tr>
<tr>
<td>Required Tasks Per Node</td>
<td>&lt;INTEGER&gt;</td>
<td>-1</td>
<td>Number of Tasks Per Node required by job or '-1' if no requirement specified.</td>
</tr>
<tr>
<td>QOS</td>
<td>&lt;STRING&gt;[:&lt;STRING&gt;]</td>
<td>-</td>
<td>QoS requested/assigned using the format &lt;QOS_REQUESTED&gt;[:&lt;QOS_DELIVERED&gt;]. (For example: hipriority:bottomfeeder)</td>
</tr>
<tr>
<td>JobFlags</td>
<td>&lt;STRING&gt;[:&lt;STRING&gt;...]</td>
<td>-</td>
<td>Square bracket delimited list of job attributes. (For example: [BACKFILL][BENCHMARK][PREEMPTEE])</td>
</tr>
<tr>
<td>Account Name</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Name of account associated with job if specified.</td>
</tr>
<tr>
<td>Executable</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Name of job executable if specified.</td>
</tr>
<tr>
<td>Resource Manager Extension String</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Resource manager specific list of job attributes if specified. See the Resource Manager Extension Overview for more information.</td>
</tr>
<tr>
<td>Bypass Count</td>
<td>&lt;INTEGER&gt;</td>
<td>-1</td>
<td>Number of times job was bypassed by lower priority jobs via backfill or '-1' if not specified.</td>
</tr>
<tr>
<td>ProcSeconds Utilized</td>
<td>&lt;DOUBLE&gt;</td>
<td>0</td>
<td>Number of processor seconds actually used by job.</td>
</tr>
<tr>
<td>Partition Name</td>
<td>&lt;STRING&gt;</td>
<td>[DEFAULT]</td>
<td>Name of partition in which job ran.</td>
</tr>
<tr>
<td>Dedicated Processors per Task</td>
<td>&lt;INTEGER&gt;</td>
<td>1</td>
<td>Number of processors required per task.</td>
</tr>
<tr>
<td>Dedicated Memory per Task</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Amount of RAM (in MB) required per task.</td>
</tr>
<tr>
<td>Dedicated Disk per Task</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Amount of local disk (in MB) required per task.</td>
</tr>
<tr>
<td>Dedicated Swap per Task</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Amount of virtual memory (in MB) required per task.</td>
</tr>
<tr>
<td>Start Date</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Epoch time indicating earliest time job can start.</td>
</tr>
<tr>
<td>End Date</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Epoch time indicating latest time by which job must complete.</td>
</tr>
<tr>
<td>Allocated Host List</td>
<td>&lt;hostname&gt;[,&lt;hostname&gt;...]</td>
<td>-</td>
<td>Comma delimited list of hosts allocated to job. (For example: node001,node004)</td>
</tr>
<tr>
<td>Resource Manager Name</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Name of resource manager if specified.</td>
</tr>
<tr>
<td>Required Host List</td>
<td>&lt;hostname&gt;[,&lt;hostname&gt;...]</td>
<td>-</td>
<td>List of hosts required by job. (If the job's taskcount is greater than the specified number of hosts, the scheduler must use these nodes in addition to others; if the job's taskcount is less...</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reservation</td>
<td>Name of reservation required by job if specified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application Simulator Data</td>
<td>Name of application simulator module and associated configuration data. (Example: HSM:IN=infile.txt:140000;OUT=outfile.txt:500000)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set Description</td>
<td>Set constraints required by node in the form &lt;SetConstraint&gt;:&lt;SetType&gt;[:&lt;SetList&gt;] where SetConstraint is one of ONEOF, FIRSTOF, or ANYOF, SetType is one of PROCSPEED, FEATURE, or NETWORK, and SetList is an optional colon delimited list of allowed set attributes. (For example: ONEOF:PROCSPEED:350:450:500)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Job Message</td>
<td>Job messages including resource manager, scheduler, and administrator messages if specified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Job Cost</td>
<td>Cost of executing job incorporating resource consumption metric, resource quantity consumed, and credential, allocated resource, and delivered QoS charge rates.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>History</td>
<td>List of job events impacting resource allocation (XML). History information is only reported in Moab 5.1.0 and higher.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Utilization</td>
<td>Comma delimited list of one or more of the following: &lt;ATTR&gt;=&lt;VALUE&gt; pairs where &lt;VALUE&gt; is a double and &lt;ATTR&gt; is one of the following: network (in MB transferred), license (in license-seconds), storage (in MB-seconds stored), or gmetric:&lt;TYPE&gt;. Cumulative resources used over life of job.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimate Data</td>
<td>List of job estimate usage.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completion Code</td>
<td>Job exit status/completion code.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extended Memory Load Information</td>
<td>Extended memory usage statistics (max, mem, avg, and so forth).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extended CPU Load Information</td>
<td>Extended CPU usage statistics (max, mem, avg, and so forth).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generic Metric Averages</td>
<td>Generic metric averages.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Effective Queue Duration</td>
<td>The amount of time, in seconds, that the job was eligible for scheduling.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If no applicable value is specified, the exact string should be entered.

Fields that contain a description string such as Job Message use a packed string format. The packed string format replaces white space characters such as spaces and carriage returns with a hex character representation. For example a blank space is represented as \20. Since fields in the event record are space delimited, this preserves the correct order and spacing of fields in the record.

Sample Workload Trace

```
13:21:05 110244355 job 1413 JOBEND 20 20 josh staff 86400 Removed [batch:1] 887343658 889585185 \
889585185 889585411 ethernet R6000 AIX53 >= 256 >= 0 - 889584538 20 0 
0 2 0 test.cmd \
1001 6 678.08 0 1 0 0 0 0 0 - - - - - - - - 0.0 - - - 0 - -
```
16.3.3.2 Workload Event Record Format (v 6.0.0.0)

All job events (JOBSUBMIT, JOBSTART, JOBEND, and so forth) provide job data in the native wiki format (ATTR=VALUE). This is to make events more readable and to allow format flexibility.

Examples

```
09:26:40 1288279600:1 sched Moab  SCHEDSTART  -
09:26:40 1288279600:2 rm  pbs  RMPSTART  -
09:26:40 1288279600:3 sched Moab  RMFOLLSTART  -
09:26:40 1288279600:4 job  58  JOBSUBMIT  58
REQUESTEDTDC=3 UNAME=wightman
GNAME=wightman WCLIMIT=60  STATE=Completed RCLASS=[batch:1]
SUBMITTIME=1288279493  RNEMCMP=>=  RDISKCMP=>=
FEATURES=[NONE]  SYSTEMQUEUETIME=1288279493  TASKS=1
FLAGS=RESTARTABLE PARTITION=pbs  DPRCCs=1
ENDDATE=2140000000  TASKMAP=proxy,GLOBAL  SRM=pbs
MESSAGE="\STARTLabel\20\20\20CreateTime\20ExpireTime\20\20\20Owner\20Prio\20Num\20Message\0a\STARTcheckpoint\20record\2EXITCODE=0  SID=2357
NODEALLOCATIONPOLICY=SHARED
```

16.3.3.3 Creating New Workload Simulation Traces

Because workload event records and simulation workload traces use the same format, these event records can be used as a starting point for generating a new simulation trace. In the Moab simple case, an event record or collection of event records can be used directly as the value for the SIMWORKLOADTRACEFILE as in the following example:

```
# collect all job records for December
> cat /opt/moab/stats/events.*Dec*2006 | grep JOBEND > /opt/moab/DecJobs.txt

# edit moab.cfg for use job records
> vi /opt/moab/etc/moab.cfg
(add 'SIMWORKLOADTRACEFILE  /opt/moab/DecJobs.txt')
(set SIMRESOURCETRACEFILE, SCHEDCFG[] MODE and other simulation parameters as described in the Simulation Overview)

# start the simulation
> moab
```

In the preceding example, all non-JOBEND events were filtered out. This step is not required but only JOBEND events are used in a simulation; other events are ignored by Moab.

Modifying Existing Job Event Records

When creating a new simulation workload, it is often valuable to start with workload traces representing a well-known or even local workload. These traces preserve distribution information about job submission times, durations, processor count, users, groups, projects, special resource requests, and numerous other factors that effectively represent an industry, user base, or organization.

When modifying records, a field or combination of fields can be altered, new jobs inserted, or certain jobs filtered out.

Because job event records are used for multiple purposes, some of the fields are valuable for statistics or auditing purposes but are ignored in simulations. For the most part, fields representing resource utilization information are ignored while fields representing resource requests are not.

Modifying Time Distribution Factors of a Workload Trace

In some cases, simulations focus on determining the effects of changing the quantities or types of jobs or on changing policies or job ownership to see changes to system performance and resource utilization. However, other times simulations tend to focus on response-time metrics as job submission and job duration aspects of the workload are modified. Which time-based fields are important to modify depend on the simulation purpose and the setting of the JOBSUBMISSIONPOLICY parameter.

<table>
<thead>
<tr>
<th>JOBSUBMISSIONPOLICY Value</th>
<th>Critical Time Based Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL</td>
<td>WallClock Limit,Submission Time</td>
</tr>
</tbody>
</table>
### 16.3.3.4 Reservation Records/Traces

All reservation events provide reservation data in a standard format as described in the following table:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Index</th>
<th>Data Format</th>
<th>Default Value</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Time (Human)</td>
<td>0</td>
<td>[HH:MM:SS]</td>
<td>-</td>
<td>Specifies time event occurred.</td>
</tr>
<tr>
<td>Event Time (Epoch)</td>
<td>1</td>
<td>&lt;epochtime&gt;</td>
<td>-</td>
<td>Specifies time event occurred.</td>
</tr>
<tr>
<td>Object Type</td>
<td>2</td>
<td>rsv</td>
<td>-</td>
<td>Specifies record object type.</td>
</tr>
<tr>
<td>Object ID</td>
<td>3</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Unique object identifier.</td>
</tr>
<tr>
<td>Object Event</td>
<td>4</td>
<td>one of rsvcreate, rsvstart, rsvmodify, rsvfail or rsven</td>
<td>-</td>
<td>Specifies record event type.</td>
</tr>
<tr>
<td>Creation Time</td>
<td>5</td>
<td>&lt;EPOCHTIME&gt;</td>
<td>-</td>
<td>Specifies epoch time of reservation start date.</td>
</tr>
<tr>
<td>Start Time</td>
<td>6</td>
<td>&lt;EPOCHTIME&gt;</td>
<td>-</td>
<td>Specifies epoch time of reservation start date.</td>
</tr>
<tr>
<td>End Time</td>
<td>7</td>
<td>&lt;EPOCHTIME&gt;</td>
<td>-</td>
<td>Specifies epoch time of reservation end date.</td>
</tr>
<tr>
<td>Tasks Allocated</td>
<td>8</td>
<td>&lt;INTEGER&gt;</td>
<td>-</td>
<td>Specifies number of tasks allocated to reservation at event time.</td>
</tr>
<tr>
<td>Nodes Allocated</td>
<td>9</td>
<td>&lt;INTEGER&gt;</td>
<td>-</td>
<td>Specifies number of nodes allocated to reservation at event time.</td>
</tr>
<tr>
<td>Total Active Proc-Seconds</td>
<td>10</td>
<td>&lt;INTEGER&gt;</td>
<td>-</td>
<td>Specifies proc-seconds reserved resources were dedicated to one or more job at event time.</td>
</tr>
<tr>
<td>Total Proc-Seconds</td>
<td>11</td>
<td>&lt;INTEGER&gt;</td>
<td>-</td>
<td>Specifies proc-seconds resources were reserved at event time.</td>
</tr>
<tr>
<td>Hostlist</td>
<td>12</td>
<td>&lt;comma delimited list of hostnames&gt;</td>
<td>-</td>
<td>Specifies list of hosts reserved at event time.</td>
</tr>
<tr>
<td>Owner</td>
<td>13</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Specifies reservation ownership credentials.</td>
</tr>
<tr>
<td>ACL</td>
<td>14</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Specifies reservation access control list.</td>
</tr>
<tr>
<td>Category</td>
<td>15</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Specifies associated node category assigned to reservation.</td>
</tr>
<tr>
<td>Comment</td>
<td>16</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Specifies general human readable event message.</td>
</tr>
<tr>
<td>Command Line</td>
<td>17</td>
<td>&lt;STRING&gt;</td>
<td>-</td>
<td>Displays the command line arguments used to create the reservation (only shows on the rsvcreate event).</td>
</tr>
</tbody>
</table>

### 16.3.3.5 Recording Job Events

Job events occur when a job undergoes a definitive change in state. Job events include submission, starting, cancellation, migration, and completion. Some site administrators do not want to use an external accounting system and use these logged events to determine their clusters’ accounting statistics. Moab can be configured to record these events in the appropriate event file found in the Moab stats/ directory. To enable job event recording for both local and remotely staged jobs, use the RECORDEVENTLIST parameter. For
This configuration records an event each time both remote and/or local jobs are canceled, run to completion, started, or submitted. The Event Logs section details the format of these records.

**See Also**

- Event Logging Overview
- SIMWORKLOADTRACEFILE
16.3.4 Simulation Specific Configuration

View the Simulation configuration demo.

To use any Moab simulation parameter, the **MODE** attribute of SCHEDCFG must be set to **SIMULATION**. Once set, the following parameters control the environment, behavior, and policies used within the simulation:

- **Simulation Resources**
  - **SIMNODECONFIGURATION** - Specifies node configuration policies.
  - **SIMNODECOUNT** - Limits number of nodes loaded.
  - **SIMRESOURCETRACEFILE** - Specifies source of node traces.

- **Simulation Workload Specification, Queuing, and Management**
  - **SIMCPUSCALINGPERCENT** - Adjusts job walltime.
  - **SIMDEFAULTJOBFLAGS** - Sets default job flags.
  - **SIMFLAGS** - Adjusts processing of workload traces.
  - **SIMIGNOREJOBFLAGS** - Ignores specified job flags if specified in workload traces.
  - **SIMINITIALQUEUEDEPTH** - Specifies simulation backlog.
  - **SIMSUBMISSIONPOLICY** - Specifies how simulation backlog is managed.
  - **SIMURGEBLOCKEDJOBS** - Removes jobs that can never run.
  - **SIMWORKLOADTRACEFILE** - Specifies source of job traces.

- **Time/Iteration Management**
  - **SIMAUTOSHUTDOWN** - Shuts down when all jobs have been scheduled.
  - **SIMEXITITERATION** - Exits simulation on specified iteration.
  - **SIMSTARTTIME** - Sets simulation clock to specified time.
  - **SIMSTOPITERATION** - Pauses simulation on specified iteration.
  - **SIMTIME_RATIO** - Adjusts simulation time as a ratio of real time.

- **General Simulation Management**
  - ** SIMMRMRANDOMDELAY** - Inserts artificial delays into simulation resource manager queries.

Special consideration should be made for large clusters.
17.0 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) source-destination 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.

- 17.1 Grid Basics
- 17.2 Grid Configuration Basics
- 17.3 Centralized Grid Management
- 17.4 Source-Destination Grid Management
- 17.5 Localized Grid Management
- 17.6 Resource Control and Access
- 17.7 Workload Submission and Control
- 17.8 Reservations in the Grid
- 17.9 Grid Usage Policies
- 17.10 Grid Scheduling Policies
- 17.11 Grid Credential Management
- 17.12 Grid Data Management
- 17.13 Accounting and Allocation Management
- 17.14 Grid Security
- 17.15 Grid Information Services
- 17.15 Grid Diagnostics and Validation
17.1 Grid Basics

- 17.1.1 Grid Overview
- 17.1.2 Grid Benefits
- 17.1.3 Scalability
- 17.1.4 Resource Access
- 17.1.5 Load-Balancing
- 17.1.6 Single System Image (SSI)
- 17.1.7 High Availability
- 17.1.8 Grid Relationships
  - 17.1.8.1 Grid Relationships
  - 17.1.8.2 Source-Destination Management
  - 17.1.8.3 Local Management
- 17.1.9 Submitting Jobs to the Grid
- 17.1.10 Viewing Jobs and Resources

17.1.1 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.

17.1.2 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.
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 cluster and presenting a single-system image back to the end-users, administrators, and managers.

17.1.4 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.

17.1.5 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.

17.1.6 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 mean 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.

17.1.7 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
17.1.8 Grid Relationships

There are three types of relationships you can implement within the grid:

- centralized management
- source-destination management
- localized management

17.1.8.1 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 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 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.

17.1.8.2 Source-Destination Management

As with the centralized management model (master/slave), the source-destination model allows users to submit jobs to a centralized source server running Moab. However, in the source-destination 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 source-destination 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 is not as necessary.

17.1.8.3 Localized Management

The localized management (peer-to-peer) model allows you to submit jobs on one cluster and schedule the jobs on another cluster. For example, a job may be submitted on MOAB1 and run on MOAB3. Jobs can also migrate in the opposite direction (that is, from MOAB3 to MOAB1). The source servers running Moab obtain full resource information from all clusters and make intelligent scheduling decisions across all 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 any 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.

17.1.9 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.

### 17.1.10 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.

**See Also**

- Resource Control and Access
17.2 Grid Configuration Basics

- 17.2.1 Peer Configuration Overview
- 17.2.2 Initial Configuration
- 17.2.3 Viewing Jobs From Other Peers

17.2.1 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.

17.2.2 Initial Configuration

At a minimum, only two parameters must be specified to establish a peer relationship: **RMCFG** and **CLIENTCFG**. **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** parameter in the peer *moab.cfg*.

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

```plaintext
MoabServer01 moab.cfg
SCHEDCFG[MoabServer01] MODE=NORMAL SERVER=hpc-01:41111
RMCFG[MoabServer02] TYPE=moab SERVER=hpc-02:40559
...

CLIENTCFG[RMMoabServer02] KEY=3esfv0=32re2-tdbne
...

MoabServer01 moab-private.cfg
CLIENTCFG[RM:MoabServer02] KEY=3esfv0=32re2-tdbne
....

MoabServer02 moab-private.cfg
CLIENTCFG[RM:MoabServer01] KEY=3esfv0=32re2-tdbne AUTH=admin1
...
```
17.3 Centralized Grid Management (Master/Slave)

17.3.1 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.

Master moab.cfg:

```
SCHEDCFG[master] SERVER=master:42559 MODE=NORMAL
...
```

Master moab-private.cfg:

```
CLIENTCFG[RM:slave1] KEY=3esfv0=32re2-tdbne
...
```

17.3.2 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.

Slave moab.cfg:

```
SCHEDCFG[slave1] SERVER=slave1:42559 MODE=SLAVE
...
```

Slave moab-private.cfg:

```
CLIENTCFG[RM:master] KEY=3esfv0=32re2-tdbne AUTH=admin1
...
```
17.4 Source-Destination Grid Management

- 17.4.1 Configuring a Peer Server (Source)
  - 17.4.1.1 Simple Source-Destination Grid

17.4.1 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.

17.4.1.1 Simple Source-Destination 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
```

In this example, 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 policies 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[RMM:C2] KEY=fastclu3t3r
CLIENTCFG[RMM: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[RMM:C1] KEY=fastclu3t3r AUTH=admin1
CLIENTCFG[RMM: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.
17.5 Localized Grid Management

- 17.5.1 Enabling Bi-Directional Job Flow
  - 17.5.1.1 True Peer-to-Peer Grid

17.5.1 Enabling Bi-Directional Job Flow

For each peer interface, an RMCFG 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.

17.5.1.1 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

<table>
<thead>
<tr>
<th>Command</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEDCFG[clusterA]</td>
<td>MODE=NORMAL SERVER=clusterA</td>
</tr>
<tr>
<td>RMCFG[clusterA]</td>
<td>TYPE=pbs</td>
</tr>
<tr>
<td>RMCFG[clusterB]</td>
<td>TYPE=moab SERVER=clusterB:40559</td>
</tr>
<tr>
<td>CLIENTCFG[RM:clusterB]</td>
<td>AUTH=admin1 KEY=banana16</td>
</tr>
</tbody>
</table>

Cluster B

<table>
<thead>
<tr>
<th>Command</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEDCFG[clusterB]</td>
<td>MODE=NORMAL SERVER=clusterB</td>
</tr>
<tr>
<td>RMCFG[clusterB]</td>
<td>TYPE=pbs</td>
</tr>
<tr>
<td>RMCFG[clusterA]</td>
<td>TYPE=moab SERVER=clusterA:40559</td>
</tr>
<tr>
<td>CLIENTCFG[RM:clusterA]</td>
<td>AUTH=admin1 KEY=banana16</td>
</tr>
</tbody>
</table>
17.6 Resource Control and Access

17.6.1 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

17.6.1.1 Direct Node View

Direct node import is the default resource information mode. No additional configuration is required to enable this mode.

17.6.1.2 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...
```

```
ode: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.

17.6.1.3 Managing Queue Visibility over the Grid

Queue information and access can be managed directly using the `RMLIST` attribute. This attribute can contain either a comma delimited list of resource managers which can view the queue or, if specified with a `!` (exclamation point) character, a list of resource managers which cannot view or access the queue. The
example below highlights the use of **RMLIST**.

```bash
# only the local resource manager, pbs2, can view/utilize the staff queue
CLASSCFG[staff] RMLIST=pbs2
```

If more advanced queue access/visibility management is required, consider using the resource manager **object map** feature.

### 17.6.2 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 restrict 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 1: Dedicated Grid Sandbox**

```bash
SRCFG[sandbox1] PERIOD=INFINITY HOSTLIST=node01,node02,node03
SRCFG[sandbox1] CLUSTERLIST=ALL FLAGS=ALLOWGRID
```

In the above example, 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.

### 17.6.2.1 Controlling Access on a Per Cluster Basis

```bash
# every resource manager other than chemgrid and biogrid
# may view/utilize the 'batch' queue
CLASSCFG[batch] RMLIST=!chemgrid,!biogrid

# only the local resource manager, pbs2, can view/utilize the staff queue
CLASSCFG[staff] RMLIST=pbs2
```
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.

The above sample configuration illustrates how cluster A could set up their sandboxes to follow a more complicated policy. In this policy, sandbox1 provides exclusive access to nodes 1 through 5 to jobs coming from peer ClusterB by including CLUSTERLIST=ClusterB in the definition. Reservation sandbox2 provides 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).

### 17.6.2.2 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
```

In the example above, 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.

### 17.6.2.3 Limiting Access To Peers (Source Side Limits)

In some cases, a site may want to constrain which users, accounts, queues or QOS's can utilize remote resources. Perhaps only certain users are trusted to participate in a special beta program or possibly only jobs in a certain queue will be able to find the needed applications or environment on the remote side.

Regardless of purpose, peer-to-peer job migration can be controlled on the source side using the RMCFG parameter's AUTHALIST, AUTHCLIST, AUTHGLIST, AUTHQLIST, and AUTHULIST attributes. These attributes are comma delimited and constrain who can utilize resources within the peer resource manager regardless of what authorization is allowed by the resource manager itself. Thus, even if a resource manager reports that it will accept jobs from any user, if the AUTHULIST parameter is set to steve,bob, then only jobs from those two users will be allowed to migrate to the peer cluster. If more than one authorized credential type is specified, jobs which satisfy *any* of the listed credentials will be allowed to use the resources.
17.6.2.4 Limiting Access From Peers (Destination Side Limits)

While source limits are set up on the source side of an interface and constrain which users can access remote resources, destination limits are set up on the destination side of an interface and constrain which remote workload requests will be accepted. These limits are useful when the remote peer is not under full local administrative control or can otherwise not be fully trusted. Often a remote source peer may allow unfettered access to peer resources while the destination may want to limit which jobs are allowed in locally.

Destination side credential limits are configured exactly like source side limits but are configured on the destination side of the interface. As with source side peer limits, these limits are enabled using the RMCFG parameter's AUTHALIST, AUTHLIST, AUTHGLIST, AUTHQLIST, and AUTHULIST attributes. These attributes are comma delimited and constrain which remote peer jobs can utilize local resources regardless of what authorization is allowed by the remote source resource manager itself.

```
SCHEDCFG SERVER=c1.hpc.org
# only allow staff or members of the research and demo account to use
# remote resources on c2

RMCFG[c2] SERVER=head.c2.hpc.org TYPE=moab
RMCFG[c2] AUTHGLIST=staff AUTHALIST=research,demo
...
```

```
SCHEDCFG SERVER=c1.hpc.org FLAGS=client
# only allow jobs from remote cluster c1 with group credentials staff
# or
# account research or demo to use local resources

RMCFG[c2] SERVER=head.c2.hpc.org TYPE=moab
RMCFG[c2] AUTHGLIST=staff AUTHALIST=research,demo
...
```
17.7 Workload Submission and Control

- 17.7.1 Controlling Peer Workload Information
- 17.7.2 Determining Resource Availability

17.7.1 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
...
```

The preceding 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.

See Also

- Job Start Time Estimates
17.8 Reservations in the Grid

17.8.1 Shared Resources

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 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.
17.9 Grid Usage Policies

- 17.9.1 Grid Usage Policy Overview
- 17.9.2 Peer Job Resource Limits
- 17.9.3 Usage Limits via Peer Credentials
- 17.9.4 Using General Policies in a Grid Environment
  - 17.9.4.1 Source Cluster Policies

17.9.1 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

17.9.2 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.

17.9.3 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, QoS's, 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

17.9.4 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 that is 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.

17.9.4.1 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 1: Constraining Remote Resource Access via Fairshare Caps**

```plaintext
# 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 2: Constraining Remote Resource Access via Fairshare Targets and Preemption**

```plaintext
# 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: Constraining Remote Resource Access via Priority and Usage Limits**

```plaintext
# 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
```
See Also

- Grid Sandbox - control grid resource access
17.10 Grid Scheduling Policies

- 17.10.1 Peer-to-Peer Resource Affinity Overview
- 17.10.2 Peer Allocation Policies
- 17.10.3 Per-partition Scheduling

17.10.1 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.

17.10.2 Peer Allocation Policies

A direct way to assign a peer allocation algorithm is with the PARALLOCATIONPOLICY parameter (does not apply to Master/Slave grids). Legal values are listed in the following table:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BestFit</td>
<td>Allocates resources from the eligible peer with the fewest available resources; measured in tasks (minimizes fragmentation of large resource blocks).</td>
</tr>
<tr>
<td>BestPFit</td>
<td>Allocates resources from the eligible peer with the fewest available resources; measured in percent of configured resources (minimizes fragmentation of large resource blocks).</td>
</tr>
<tr>
<td>FirstStart</td>
<td>Allocates resources from the eligible peer that can start the job the soonest.</td>
</tr>
<tr>
<td>FirstCompletion</td>
<td>Allocates resources from the eligible peer that can complete the job the soonest. (Takes into account data staging time and job-specific machine speed.)</td>
</tr>
<tr>
<td>LoadBalance</td>
<td>Allocates resources from the eligible peer with the most available resources; measured in tasks (balances workload distribution across potential peers).</td>
</tr>
<tr>
<td>LoadBalanceP</td>
<td>Allocates resources from the eligible peer with the most available resources; measured in percent of configured resources (balances workload distribution across potential peers).</td>
</tr>
<tr>
<td>RoundRobin</td>
<td>Allocates resources from the eligible peer that has been least recently allocated.</td>
</tr>
</tbody>
</table>

The mdiag -t -v command can be used to view current calculated partition priority values.

17.10.3 Per-partition Scheduling

Per-partition scheduling can be enabled by adding the following lines to moab.cfg:

```plaintext
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.
17.11 Grid Credential Management

- 17.11.1 Peer User Credential Management Overview
- 17.11.2 Credential Mapping Files
- 17.11.3 Constraining Access via Default and Mandatory Queues and QoS

17.11.1 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.

17.11.2 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 the special character `'*` (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:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>accounts/projects</td>
</tr>
<tr>
<td>class</td>
<td>classes/queues</td>
</tr>
<tr>
<td>file</td>
<td>files/directories</td>
</tr>
<tr>
<td>group</td>
<td>groups</td>
</tr>
<tr>
<td>node</td>
<td>nodes</td>
</tr>
<tr>
<td>qos</td>
<td>QoS</td>
</tr>
<tr>
<td>user</td>
<td>users</td>
</tr>
</tbody>
</table>

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[slave1] MODE=normal
RMCFG[slave1] OMAP=file:///opt/moab/omap.dat  FLAGS=client
...

user:*,cl_ *
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 cl_john, the group staff will be remapped to staff_grid and the account demo will be remapped to temp_demo.

17.11.3 Constraining Access via Default and Mandatory Queues and QoS

While credential mapping allows a source side peer to translate one credential to another, a destination side peer may require control over this mapping as a prerequisite to participating in the peer relationship. For example, a given cluster may select to join a peer grid only if they can map all remote jobs into a specific class and apply limits constraining when and where this class is able to run.

In Moab, destination-side credential constraints are enabled using the RMCFG parameter's SET.JOB, MIN.JOB, MAX.JOB, and DEFAULT.JOB attributes.

Set Job Attribute

If the resource manager's SET.JOB attribute is set, each credential attribute specified within the remap job template will be used to overwrite values specified by the peer service. This is done regardless of whether the source job attributes are set. For example, in the following configuration, the job's account and class credentials will be remapped regardless of original values:

```
# define connection to remote peer cluster 'clusterB'
RMCFG[clusterB.INBOUND] TYPE=moab SET.JOB=set1

# force jobs coming in from clusterB interface to use account 'peer'
# and class 'remote'
JOBCFG[set1] ACCOUNT=peer CLASS=remote
```
Minimum and Maximum Job Constraints

The **MIN.JOB** and **MAX.JOB** attributes can be used to link a particular job template to a resource manager interface. If set, jobs that do not meet these criteria will be rejected.

Default Job Settings

If the resource manager's **DEFAULT.JOB** attribute is set, each credential attribute specified within the default job template will be used to set job attribute values that are not explicitly set by the peer service. For example, in the following configuration, the job's account and class credentials will be remapped if no original values are specified:

```
JOBCFG[default-hq.INBOUND] TYPE=TEMPLATE ACCOUNT=peer CLASS=remote
RMCFG[hq] DEFAULT.JOB=default-hq
...```
17.12 Grid Data Management

- 17.12.1 Grid Data Management Overview
- 17.12.2 Configuring Peer Data Staging
- 17.12.3 Diagnostics
- 17.12.4 Peer-to-Peer SCP Key Authentication

17.12.1 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.

17.12.2 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`.

Automatic 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 17.0 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.

4. Associate the data resource manager with a peer resource manager.
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 `$MOABTOOLSDIR`; this file is generated from the config.dstage.pl.dist file each time Moab restarts. Moab replaces any strings of the form `@...@` with appropriate values.

**Advanced Configuration**

If you need a more customized data staging setup, contact Moab support.

### 17.12.3 Diagnostics

Verify data staging is properly configured by using the following diagnostic commands:

- `mdiag -R -v`: Displays the status of the storage manager. Notice that the automatic configuration sets up the necessary *URLs.

```
> mdiag -R -v data
diagnosing resource managers

RM[data]       State: Active  Type: NATIVE:AGFULL  ResourceType: STORAGE
  Server:       keche
  Timeout:      30000.00 ms
  Cluster Query URL: exec://$TOOLSDIR/cluster.query.dstage.pl
  RM Initialize URL: exec://$TOOLSDIR/setup.config.pl
  System Modify URL: exec://$TOOLSDIR/system.modify.dstage.pl
  System Query URL: exec://$TOOLSDIR/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.

```
> 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)
```

The number of bytes transferred for each file is currently not used.
17.12.4 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:
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:

```bash
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:

```bash
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:

```bash
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):

```bash
---
RSAAuthentication yes
PubkeyAuthentication yes
---
```

If configuration changes were required, the SSH daemon will need to be restarted:

```bash
/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:

```bash
scp ${DESTPEERHOST}:/etc/motd /tmp/
```
17.13 Accounting and Allocation Management

- 17.13.1 Peer-to-Peer Accounting Overview
- 17.13.2 Peer-to-Peer Allocation Management

17.13.1 Peer-to-Peer Accounting Overview

When Moab is used to manage resources across multiple clusters, there is a greater need to track and enforce the resource sharing agreements between the resource principals.

The Gold Allocation Manager is an open source accounting system that tracks resource usage on High Performance Computers. It acts much like a bank in which accounts are established to pre-allocate which users and projects can use resources on which machines and over which timeframes. Gold supports familiar operations such as deposits, withdrawals, transfers, and refunds. It provides balance and usage feedback to users, managers, and system administrators.

Gold can be used as a real-time debiting system in which jobs are charged at the moment of completion. When used in a multi-site (grid) environment, Gold facilitates trust by allowing lending organizations to manage what the costing rules are for usage of their resources and job submitters to determine how much their job is going to cost them before they start, ensuring all parties can agree to the transaction and giving each party a first-hand accounting record.

If the clusters are within a common administrative domain and have a common user space, then a single Gold Allocation Manager will suffice to manage the project allocation and accounting. This works best in Master/Slave grids.

17.13.2 Peer-to-Peer Allocation Management

The following steps provides an example of setting up the Gold Allocation Manager to manage the allocation and accounting for a multiple cluster grid within a single administrative domain.

First you will need to install Gold and its database on one or more head nodes.

```bash
# Install Prerequisites (Perl with suidperl, PostgreSQL, libxml2, ...
[root]  yum install perl perl-suidperl postgresql postgresql-libs postgresql-devel postgresql-server libxml2

# Unpack the tarball
[root]  passwd gold
[gold]  mkdir ~/src
[gold]  cd ~/src
[gold]  gzip -cd gold-2.0.0.0.tar.gz | tar xvf -
[gold]  cd gold-2.0.0.0

# Install
[gold]  ./configure
[gold]  make
[root]  make deps
[root]  make install
[root]  make auth_key

# Configure, create and bootstrap the database
[postgresql]  /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
[postgresql]  echo "host all all 192.168.1.1 255.255.255.255 trust" >>/usr/local/pgsql/data/pg_hba.conf
[postgresql]  /usr/local/pgsql/bin/postmaster -i -D
 [usr/local/pgsql/data] >var/log/pgsql 2>&1 &
[postgresql]  createdb gold
[gold]  /usr/local/pgsql/bin/createdb gold
[postgresql]  /usr/local/pgsql/bin/psql gold < bank.sql
```
See Also

- Grid Credential Management
17.14 Grid Security

17.14.1 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.
17.15 Grid Diagnostics and Validation

- 17.15.1 Peer Management Overview
- 17.15.2 Peer Diagnostic Overview

17.15.1 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.
- Use `mdiag -x` to view general grid configuration and system diagnostics.

17.15.2 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.
- 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.

```bash
> mdiag -R -V job peer1
```
18.0 Green Computing

- 18.1 Establishing Script Interaction between Moab and a Power Management Tool
- 18.2 Enabling Green Power Management
- 18.3 Allocating and Adjusting Green Pool Size
- 18.4 Miscellaneous Power Management Options

Green Computing Overview

Green Computing is a video tutorial that offers further details on green computing.

To conserve energy, Moab can turn off idle nodes that have no reservations on them. Conversely, Moab turns on additional nodes when jobs require such. Using the `MAXGREENSTANDBYPOOLSIZEx` parameter, you can specify a "green pool," which is the number of nodes Moab keeps turned on and ready to run jobs (even if some nodes are idle). Moab turns off idle nodes that exceed the number specified with the `MAXGREENSTANDBYPOOLSIZEx` parameter.

Example scenario:

Assume a cluster is empty and Moab starts. Moab turns off all nodes except those established by the green pool size. As jobs are submitted, Moab uses the machines that are already turned on, and Moab turns on additional nodes to keep the green pool at the configured size. As jobs complete, nodes turn off to keep the green pool at the configured size.

Requirements:

- A license for green computing.
- Moab 5.3.5 or later.
- A script that Moab can call to programatically turn nodes on and off.
- A resource manager that can monitor and report power state.
18.1 Establishing Script Interaction between Moab and a Power Management Tool

On the same node on which Moab is running, there must be a command for Moab to call to switch power on or off. The command is usually a script that interacts with your preferred power management tool. Moab calls the script as follows:

```
<Script Name> <Node Name/List> <ON|OFF>
```

```
> /opt/moab/tools/node.power.pl node002 ON
```

<Node List> is a comma delimited list of nodes. To power off multiple nodes, try:

```
> /opt/moab/tools/node.power.pl node002,node003,node004 OFF
```

To enable script interaction between Moab and your preferred power management tool, configure the `NODEPOWERURL` parameter on a per-resource manager basis.

```
RMCFG[prov] TYPE=NATIVE RESOURCETYPE=PROV
RMCFG[prov] NODEPOWERURL=exec://$HOME/tools/prov/node.power.pl
RMCFG[prov] CLUSTERQUERYURL=exec://$HOME/tools/prov/cluster.query.pl
```

Run `mdiag -R -v` to see if the provision resource manager configured successfully. The output will display a number of nodes if Moab accessed the node power script and Cluster Query URL script properly. If a number of nodes is not reported, the script is either not running properly or not reporting to Moab correctly. If the script is not working correctly, verify that the script can be run manually and that no environment variables are required for the script to run, such as:

* The Cluster Query script can call out commands that are normally in the $PATH when they are not set. When this occurs, the script fails and the commands may be difficult to find.

The script's purpose is to provide Moab information about whether power to nodes is on or off; such information is relayed for each node with a `POWER` value of either "on" or "off." The actual state of nodes is internally tracked by Moab. When Moab powers off a node, it is still eligible to run jobs, so even though it is powered off, its state is idle. Thus, this script must report a `STATE` of "Unknown" to prevent Moab from considering the node unavailable. When a node is powered off outside of a Moab action (such as a node failure), then Moab recognizes the state being reported from its resource manager as down (rendering it unavailable). To prevent Moab from considering a node unavailable, as previously mentioned, the `STATE` must be reported as "Unknown." The following is sample output for `cluster.query.pl`:

```
x01 POWER=on STATE=Unknown
x02 POWER=off STATE=Unknown
x03 POWER=off STATE=Unknown
x04 POWER=off STATE=Unknown
x05 POWER=off STATE=Unknown
```

* Managing Resources Directly with the Native Interface
18.2 Enabling Green Power Management

To enable green power management, modify the configuration file as follows:

```
NODECFG[DEFAULT] POWERPOLICY=OnDemand
```

Use the `mdiag -G` command to see which nodes are off. The following shows sample output from running the `mdiag -G` command:

```
$ mdiag -G
NOTE: power management enabled for all nodes

<table>
<thead>
<tr>
<th>NodeID</th>
<th>State</th>
<th>Power</th>
<th>Watts</th>
<th>PWatts</th>
</tr>
</thead>
<tbody>
<tr>
<td>x01</td>
<td>Idle</td>
<td>On</td>
<td>200.00</td>
<td>200.00</td>
</tr>
<tr>
<td>x02</td>
<td>Idle</td>
<td>Off</td>
<td>25.00</td>
<td>200.00 (powered off by Moab)</td>
</tr>
<tr>
<td>x03</td>
<td>Idle</td>
<td>Off</td>
<td>25.00</td>
<td>200.00 (powered off by Moab)</td>
</tr>
<tr>
<td>x04</td>
<td>Idle</td>
<td>Off</td>
<td>25.00</td>
<td>200.00 (powered off by Moab)</td>
</tr>
<tr>
<td>x05</td>
<td>Idle</td>
<td>Off</td>
<td>25.00</td>
<td>200.00 (powered off by Moab)</td>
</tr>
</tbody>
</table>
```
18.3 Allocating and Adjusting Green Pool Size

The\texttt{MAXGREENSTANDBYPOOLSIZE} allows you to allocate the number of nodes to keep powered on in the standby pool. Moab will make sure that \texttt{MAXGREENSTANDBYPOOLSIZE} nodes are turned on at all times and available to run jobs. Additional nodes will be turned on as needed.
18.4 Miscellaneous Power Management Options

- 18.4.1 Maximizing Scheduling Efficiency
- 18.4.2 Logging Power-Related Events
- 18.4.3 Enabling and Disabling Power Management for All Nodes

18.4.1 Maximizing Scheduling Efficiency

To facilitate more efficient scheduling, it is a good idea to specify the maximum amount of time a power on or power off action takes to complete.

```
PARCFG[torque] NODEPOWERONDURATION=2:00
PARCFG[torque] NODEPOWEROFFDURATION=2:00
```

With the NODEIDLEPOWERTHRESHOLD parameter, Moab can turn nodes off after they've been idle for a specific amount of seconds. The default value is 60 seconds.

```
NODEIDLEPOWERTHRESHOLD 600
```

18.4.2 Logging Power-Related Events

Power actions are considered NODEMODIFY events that are not, by default, recorded. Adding the following line to your configuration file enables recording power-related actions in the events file.

```
RECORDEVENTLIST +NODEMODIFY
```

18.4.3 Enabling and Disabling Power Management for All Nodes

You can enable or disable green power management for all nodes while Moab runs by using the changeparam command. For example:

To enable green power management: changeparam NODECFG[DEFAULT] POWERPOLICY=OnDemand
To disable green power management: changeparam NODECFG[DEFAULT] POWERPOLICY=STATIC

The default POWERPOLICY setting is STATIC.

```
NODECFG[DEFAULT] POWERPOLICY=OnDemand
```
19.0 Object Triggers

- 19.1 Trigger Creation
- 19.2 Trigger Management
- 19.3 Trigger Components
- 19.4 Trigger Types
- 19.5 Trigger Variables
- 19.6 Trigger Examples

Moab can launch triggers based on certain events. For example, an administrator may want to send an email to the owner of a particular reservation when the usage drops below a specific threshold. Or a user may want to launch an evaluation script five minutes before a job is scheduled for completion. Triggers can be associated with jobs, nodes, reservations, or the core scheduler, and they enable actions to be taken when the specified event, offset, or threshold criteria are satisfied.
19.1 Trigger Creation

- 19.1.1 Trigger Creation Syntax
- 19.1.2 Creating Triggers
- 19.1.3 Naming Triggers
- 19.1.4 Associating the JOBTRIGGER Attribute with a Class

19.1.1 Trigger Creation Syntax

Use the following format to create triggers:

```plaintext
<ATTR>=<VALUE>[[&]<ATTR>=<VALUE>]]...
```

19.1.2 Creating Triggers

Triggers can be created from both the configuration file and the command line. Triggers can be associated with an object such as a standing reservation, node, resource manager, or the scheduler when the object is instantiated. To do this, use the TRIGGER attribute with the associated object parameter, such as `SRCFG` (see Managing Reservations for more details), `NODECFG`, `RMCFG`, or `SCHEDCFG`.

To dynamically add a trigger to an existing object, use the `mschedctl` command. To add a trigger to an administrative reservation, use the `mrsvctl` command. Triggers may be added to jobs by specifying the `trig` resource manager extension when submitting the job through `msub`, as in the following example:

**Example**

```plaintext
msub my.job -l 'trig=AType=exec&Action="/jobs/my_trigger.pl"&EType=create&Offset=10
```

Triggers can also be created on system jobs using the `msub` command, as shown in the following example. The benefit of system job triggers is that system jobs do not have to require any system resources. System jobs exist until manually cancelled or completed, a process that can be handled by scripts or trigger actions. This provides one means for creating standing triggers that fire at a specified time or when certain external dependencies are fulfilled.

**Example**

```plaintext
echo true | msub -l flags=NORMSTART:NORESOURCES,trig=AType=exec&Action="/tmp/action.sh"&EType=SUBMIT
```

For security reasons, only QoS with the `trigger` flag can have jobs with attached triggers. This can be set up by creating an appropriate QoS in the `moab.cfg` file.

**Example**

```plaintext
QOSCFG[triggerok] QFLAGS=trigger
USERCFG[user1] QDEF=triggerok
```

In this example, a QoS named `triggerok` is created with the `trigger` flag. The user `user1` is then added to this QoS. This user will then be able to attach triggers to jobs.

Please note that when creating job triggers via `msub`, "&" must be used to delimit the fields. All job triggers are removed when the associated job is cancelled.

19.1.3 Naming Triggers

By default, triggers are assigned a numeric ID by Moab. However, in many cases, managing triggers can
become quite complicated if relying only on this ID. Therefore, users may assign an alphanumeric name up to 16 characters in length for each trigger. The Moab-assigned ID number is then appended to the end of the user-supplied name. The trigger name is specified at creation or modification using the *Name* trigger attribute.

**Example**

```plaintext
msub my.job -l
  'trig=Name=myTrigger\&AType=exec\&Action="/jobs/my_trigger.pl"\&EType=class'
```

In this case, the new trigger attached to *my.job* will be named `myTrigger.<TRIGID>`, where `<TRIGID>` is the ID assigned automatically by Moab. This name will appear in the output of the `mdig -T` command. Carefully chosen trigger names can greatly improve the readability of this output.

### 19.1.4 Associating the JOBTRIGGER Attribute with a Class

Job triggers can be directly associated with jobs submitted into a class using the *JOBTRIGGER* attribute. Job triggers are described using the standard trigger description language specified in the *Trigger* overview section. In the example that follows, users submitting jobs to the class *debug* will be notified with a descriptive message any time their job is preempted.

```plaintext
CLASSCFG[batch]
  JOBTRIGGER=atype=exec,etype=preempt,action=
  "$HOME/tools/preemptnotify.pl $OID $OWNER $HOSTNAME"
```

**See Also**

- Credential Overview
- Generic Metrics
- Generic Events
19.2 Trigger Management

- 19.2.1 Viewing Triggers
- 19.2.2 Modifying Triggers
- 19.2.3 Checkpointing Triggers

19.2.1 Viewing Triggers

A condensed listing of triggers can be viewed using the `mdia\$ -T \[<TRIGID>|<OBJECTID>\]` command. This will show all triggers to which the user has access, along with some basic information about the triggers. Specific triggers or all triggers attached to a specific object can be viewed by including the ID of the trigger or object.

**Example**

```
> mdiag -T
> mdiag -T trigger.34
> mdiag -T job.493
```

Additional information can be obtained using the verbose mode of the `mdia\$ -T -v \[<TRIGID>|<OBJECTID>\]` command. As before, a subset can be obtained by specifying either a trigger or object ID.

**Example**

```
> mdiag -T -v
> mdiag -T -v trigger.34
> mdiag -T -v job.493
```

The `mdia\$ -T -V` command is another viewing mode that provides additional state information about triggers, including reasons why triggers are currently blocked. This mode outputs information on a single line for each trigger, as opposed to the multiline output of the `mdia\$ -T -v \[<TRIGID>|<OBJECTID>\]` command. However, the `mdia\$ -T -V` command only operates in a global mode, showing all triggers available to the user.

**Example**

```
> mdiag -T -V
```

19.2.2 Modifying Triggers

Triggers may be modified using the `mschedctl -m trigger:<TRIGID> <ATTR>=<VAL>` command.

**Example**

```
> mschedctl -m trigger:2 AType=exec,Offset=200,OID=system.1
```

In this example, Trigger 2’s AType, Offset, and OID are changed to the new values shown.

19.2.3 Checkpointing Triggers

Checkpointing is the process of saving state information when Moab is shut down.

By default, triggers attached to objects including the scheduler, resource managers, credentials, and nodes are not checkpointed but are re-created from specifications in the Moab configuration (moab.cfg) file when Moab is restarted. If a trigger attached to one of these objects needs to be checkpointed because it was created at the command line (as opposed to in the configuration file), the checkpoint flag must be attached to the trigger. When creating a trigger using the `mschedctl` command, be sure to include the checkpoint flag.
See Also

- Generic Metrics
- Generic Events
# 19.3 Trigger Components

- 19.3.1 Trigger Attributes
- 19.3.2 Trigger Flags

## 19.3.1 Trigger Attributes

### Table 1: Trigger Attributes

<table>
<thead>
<tr>
<th>AType</th>
<th>Possible Values:</th>
<th>Description</th>
<th>Usage Notes:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cancel, changeparam, jobpreempt, mail, exec, internal</td>
<td>Specifies what kind of action the trigger will take when it fires.</td>
<td>jobpreempt indicates that the trigger should preempt all jobs currently allocating resources assigned to the trigger's parent object. cancel and jobpreempt only apply to reservation triggers.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
<th>Possible Values:</th>
<th>Description</th>
<th>Usage Notes:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;STRING&gt;</td>
<td>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).</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BlockTime</th>
<th>Possible Values:</th>
<th>Description</th>
<th>Usage Notes:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[[HH:]MM:]SS</td>
<td>Time (in seconds) Moab will suspend normal operation to wait for trigger execution to finish.</td>
<td>Use caution as Moab will completely stop normal operation until BlockTime expires.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>Possible Values:</th>
<th>Description</th>
<th>Usage Notes:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;STRING&gt;</td>
<td>Description of the trigger.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ExpireTime</th>
<th>Possible Values:</th>
<th>Description</th>
<th>Usage Notes:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;INTEGER&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Time at which trigger should be terminated if it has not already been activated.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Usage:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Notes:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **EType** | Possible Values: cancel, checkpoint, create, end, epoch, fail, hold, migrate, modify, preempt, standing, start, threshold |
| **Description:** | The type of event that signals that the trigger can fire. |
| **Usage Notes:** | The threshold value applies to reservation triggers and gmetrics on nodes. Cancel triggers are the only triggers to fire when the parent object is either canceled or deleted (though its dependencies, if any, must still be satisfied). Hold, preempt, and checkpoint only apply to job triggers. Standing allows regular periodic triggers that fire at regular offsets (standing triggers automatically set the interval and multifire attributes). Standing triggers can be used with only the scheduler; they are not valid for any other object type. |

| **FailOffset** | Possible Values: [[HH:]MM:]SS |
| **Description:** | Specifies the time (in seconds) that the threshold condition must exist before the trigger fires. |
| **Usage Notes:** | |

| **Flags** | Possible Values: (See Table 2.) |
| **Description:** | Specifies various trigger behaviors and actions. |
| **Usage Notes:** | |

| **Interval** | Possible Values: <BOOLEAN> |
| **Description:** | When used in conjunction with MultiFire and RearmTime trigger will fire at regular intervals. |
| **Usage Notes:** | Can be used with EType epoch to create a Standing Trigger. |

| **MaxRetry** | Possible Values: <INTEGER> |
| **Description:** | Specifies the number of times Action will be attempted before the trigger is designated a failure. |
| **Usage Notes:** | If Action fails, the trigger will be restarted immediately (up to MaxRetry times). If it fails more than MaxRetry times, the trigger will be considered to have failed. This restart ignores FailOffset and RearmTime. Multifire does not need to be specified in order to get this |
### MultiFire

**Possible Values:** `<BOOLEAN>`

**Description:** Specifies whether this trigger can fire multiple times.

**Usage Notes:**

### Name

**Possible Values:** `<STRING>`

**Description:** Name of 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.

### Offset

**Possible Values:** `[-][HH:]MM:SS`

**Description:** Relative time offset from event when trigger can fire.

**Usage Notes:** Offset cannot be used with cancel. Only a negative Offset can be used with end triggers.

### Period

**Possible Values:** `Minute, Hour, Day, Week, Month`

**Description:** Can be used in conjunction with Offset to have a trigger fire at the beginning of the specified period.

**Usage Notes:** Can be used with EType epoch to create a standing trigger.

### RearmTime

**Possible Values:** `[[HH:]MM:]SS`

**Description:** Time between MultiFire triggers; rearm time is enforced from the trigger event time.

**Usage Notes:**

### Requires

**Possible Values:** `',' delimited string`
<table>
<thead>
<tr>
<th>Description:</th>
<th>Variables this trigger requires to be set or not set before it will fire.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage Notes:</td>
<td>Preceding the string with an exclamation mark (!) indicates this variable must NOT be set. Used in conjunction with <strong>Sets</strong> to create trigger dependencies.</td>
</tr>
</tbody>
</table>

**Sets**

<table>
<thead>
<tr>
<th>Possible Values:</th>
<th>'.' delimited string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Variable values this trigger sets upon success or failure.</td>
</tr>
<tr>
<td>Usage Notes:</td>
<td>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 <strong>Requires</strong> to create trigger dependencies.</td>
</tr>
</tbody>
</table>

**UnSets**

<table>
<thead>
<tr>
<th>Possible Values:</th>
<th>'.' delimited string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Variable this trigger destroys upon success or failure.</td>
</tr>
<tr>
<td>Usage Notes:</td>
<td></td>
</tr>
</tbody>
</table>

**Threshold**

<table>
<thead>
<tr>
<th>Possible Values:</th>
<th>{&lt;METRIC&gt;[&lt;METRICNAME&gt;]} {&gt; &gt;= &lt; &lt;= == !=} &lt;FLOAT&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Reservation usage threshold - When reservation usage drops below <strong>Threshold</strong>, trigger will fire.</td>
</tr>
<tr>
<td>Usage Notes:</td>
<td>Threshold <strong>usage</strong> support is only enabled for reservations and applies to percent processor utilization. <strong>gmetric</strong> thresholds are supported with job, node, credential, and reservation triggers. See 19.4.3 Threshold Triggers for more information.</td>
</tr>
</tbody>
</table>

**Timeout**

| Possible Values: | [+|-][[HH:]MM:]SS |
|------------------|-------------------|
| Description:     | Time allotted to this trigger before it is marked as unsuccessful and its process (if any) killed. |
| Usage Notes:     | Specifying '+' or a '-' calculates the timeout relative to the parent object's duration. |

### 19.3.2 Trigger Flags
### Table 2: Trigger Flag Values

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attacherror</td>
<td>If the trigger outputs anything to stderr, Moab will attach this as a message to the trigger object.</td>
</tr>
<tr>
<td>cleanup</td>
<td>If the trigger is still running when the parent object completes or is canceled, the trigger will be killed.</td>
</tr>
<tr>
<td>checkpoint</td>
<td>Moab should always checkpoint this trigger. See Checkpointing Triggers for more information.</td>
</tr>
<tr>
<td>globalvars</td>
<td>The trigger will look in the name space of all nodes with the <code>globalvars</code> flag in addition to its own name space. A specific node to search can be specified using the following format: <code>globalvars+node_id</code></td>
</tr>
<tr>
<td>interval</td>
<td>Trigger is periodic.</td>
</tr>
<tr>
<td>multifire</td>
<td>Trigger can fire multiple times.</td>
</tr>
<tr>
<td>objectxmlstdin</td>
<td>Trigger passes its parent's object XML information into the trigger's stdin. This only works for exec triggers with reservation type parents.</td>
</tr>
<tr>
<td>user</td>
<td>The trigger will execute under the user ID of the object's owner. If the parent object is <code>sched</code>, the user to run under may be explicitly specified using the format <code>user+&lt;username&gt;</code>, for example flags=user+john.</td>
</tr>
</tbody>
</table>

When specifying multiple flags, each flag can be delimited by colons (:) or with square brackets ([ and ]); for example:

```
Flags=[user][cleanup][probe] or Flags=user:cleanup:probe
```

### See Also

- [Generic Metrics](#)
- [Generic Events](#)
19.4 Trigger Types

- 19.4.1 Mail Triggers
- 19.4.2 Internal Triggers
- 19.4.3 Threshold Triggers
- 19.4.4 Exec Triggers

19.4.1 Mail Triggers

When the mail option is used for the AType parameter, Moab will send mail (configurable using MAILPROGRAM) to the primary administrator. When AType=mail, the Action parameter contains the message body of the email. This can be configured to include certain variables. The following example sends an email to the primary administrator containing the name of the trigger's object, and the time:

```
> mrsvctl -c -h node01 -T AType=mail,EType=start \
   Action="rsv $OID started on $TIME on nodes $HOSTLIST"
rsv 'system.1' created
```

When this trigger launches, it will send an email with the variables (that begin with a $) filled in. Possible variables that can be inserted into the Action parameter include the following:

- **OID**: name of the object to which the trigger was attached.
- **TIME**: time of the trigger launch.
- **HOSTLIST**: HostList of the trigger's object (when applicable).
- **OWNER**: owner of the trigger's object (when applicable).
- **USER**: user (when applicable).

These 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.

19.4.2 Internal Triggers

Triggers can be used to modify object internals using the following format:

```
action="<OBJECT_TYPE>:<OBJECT_ID>:<ACTION>:<CONTEXT_DATA>
```

Several different actions are valid depending on what type of object the internal trigger is acting upon. The following table shows some of the different available actions:

<table>
<thead>
<tr>
<th>Action</th>
<th>Object Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancel</td>
<td>Jobs</td>
<td>Cancels job.</td>
</tr>
<tr>
<td>complete</td>
<td>System Jobs</td>
<td>Causes system job to exit as if it had completed successfully.</td>
</tr>
<tr>
<td>destroy</td>
<td>VPC</td>
<td>Destroys the specified VPC.</td>
</tr>
<tr>
<td>evacvms</td>
<td>VM</td>
<td>Evacuates all VMs from all nodes covered by a reservation.</td>
</tr>
<tr>
<td>modify</td>
<td>All</td>
<td>Allows modification of object internals.</td>
</tr>
</tbody>
</table>

Examples — Modifying scheduler internals

```
SRCFG[provision]
TRIGGER=atype=internal,ETYPE=start,action="node:$(HOSTLIST):modify:os=......
```
19.4.3 Threshold Triggers

Threshold triggers allow sites to configure triggers to launch based on internal scheduler statistics, such as the percentage of available nodes, the backlog of a particular QoS, or the xfactor of a particular group. For example, a site may need to guarantee a particular account a certain level of service on the cluster. Should the specified account have a backlog of over one hour, the administrators would like to receive an email, or create a reservation, or perhaps contact a hosting utility to allocate more nodes. All of this can be done using threshold triggers. The following sample configuration file contains a few examples:

```
RMCFG[TORQUE]
TRIGGER=atype=internal,action=RM:ODM:modify:nodes:1,etype=threshold,threshold=queuetime>100000,multifire=true

CLASSCFG[batch] TRIGGER=atype=mail,action="batch has a lot of jobs waiting",etype=threshold,threshold=queuetime>100000,multifire=true

QOSCFG[high] TRIGGER=atype=exec,action=/tmp/reserve_high.sh,etype=threshold,threshold=queuetime>100000,multifire=true

USERCFG[jdoe] TRIGGER=atype=mail,action="high xfactor on $OID",etype=threshold,threshold=xfactor>10.0,multifire=true,failoffset=0

ACCTCFG[hyper] TRIGGER=atype=exec,action="/tmp/notify.sh $TIME $OID",etype=threshold,threshold=xfactor>0.01,multifire=true,failoffset=0

NODECFG[node04] TRIGGER=atype=exec,action="$HOME/hightemp.py $OID",etype=threshold,threshold=gmetric[TEMP]>70
```

19.4.4 Exec Triggers

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.

```
> msub -l
trig=atype=exec\&etype=start\&offset=03:00\&action="monitor.pl"
job.cmd
```

See Also

- Generic Metrics
- Generic Events
19.5 Trigger Variables

- 19.5.1 Trigger Variables Overview
- 19.5.2 Default Internal Variables Available to Trigger Scripts
- 19.5.3 Externally Injecting Variables into Job Triggers
- 19.5.4 Exporting Variables to Parent Objects
- 19.5.5 Requiring Variables in Parent Objects

19.5.1 Trigger Variables Overview

Trigger variables can add greater flexibility and power to a site administrator who wants to automate certain tasks and system behaviors. Variables allow triggers to launch based on another trigger's behavior, state, and/or output.

Example

```
AType=exec,Action="/tmp/trigger1.sh",EType=start,Sets=Var1.Var2 \
AType=exec,Action="/tmp/trigger2.sh $Var1 $Var2",EType=start,Requires=Var1.Var2
```

In this example, the first trigger sets two variables (separated by a '.'), which are received in the second (separated by a ':'). As previously mentioned, those arguments could be accessed in the second trigger through the variables $1 and $2.

It is also possible to have a trigger set a variable when it fails using the '!' symbol:

Example

```
AType=exec,Action="/tmp/trigger1.sh",EType=start,Sets=!Var1.Var2 \
AType=exec,Action="/tmp/trigger2.sh",EType=start,Requires=Var1 Var2
```

In this example, the first trigger will set Var1 if it fails and Var2 if it succeeds. The second trigger will launch if Var1 has been set (the first trigger failed). The third trigger will launch if Var2 is set (the first trigger succeeded).

Variable requirements can be further refined to allow for the evaluation and comparison of the variable's value. That is, triggers can have a dependency on a variable having (or not having) a certain value. The format for this is as follows:

```
[*]<VARID>[:<TYPE>[:<VARVAL>]]
```

The optional * specifies that the dependencies are satisfied by an external source that must be previously registered. A number of valid comparison types exist:

<table>
<thead>
<tr>
<th>Type</th>
<th>Comparison</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>is set (exists)</td>
<td>Default</td>
</tr>
<tr>
<td>notset</td>
<td>not set (does not exits)</td>
<td>Same as specifying '!' before a variable</td>
</tr>
<tr>
<td>eq</td>
<td>equals</td>
<td></td>
</tr>
<tr>
<td>ne</td>
<td>not equal</td>
<td></td>
</tr>
<tr>
<td>gt</td>
<td>greater than</td>
<td>Integer values only</td>
</tr>
<tr>
<td>lt</td>
<td>less than</td>
<td>Integer values only</td>
</tr>
<tr>
<td>ge</td>
<td>greater than or equal to</td>
<td>Integer values only</td>
</tr>
</tbody>
</table>
Following is an example of how these comparative dependencies can be expressed when creating a trigger.

**Example**

```
AType=exec,Action="/tmp/trigger2.sh",EType=start,Requires=Var1:eq:45 \
AType=exec,Action="/tmp/trigger3.sh",EType=start,Requires=Var2:ne:failure1
```

In this example, the first trigger will fire if `Var1` exists and has a value of 45. The second trigger will only fire if `Var2` is not the string `failure1`.

The following example shows how triggers can set variables as strings:

**Example**

```
AType=exec,Action="/tmp/trigger2.sh",EType=start,Sets=Var1
```

The trigger sets Var1 to TRUE when it completes successfully. Because AType=exec, the script launched by the trigger can set a string value for Var1. To do this, declare it on its own line in the trigger stdout. For example:

```
EXITCODE=15
Var1=linux
```

Var1 has the value `linux` and, if defined with a caret (^), can be passed up to the job group. This is useful for workflows in which a trigger may depend on the value given by a previous trigger.

### 19.5.2 Default Internal Variables Available to Trigger Scripts

Several internal variables are available for use in trigger scripts. These can be accessed using `$<VARNAME>`:

- **ETYPE** is the type of event that signals that the trigger can fire; **ETYPE** values include cancel, checkpoint, create, end, epoch, fail, hold, migrate, modify, preempt, standing, start, and threshold.
- **HOSTLIST** is the HostList of the trigger’s object (when applicable).
- **OID** is the name of the object to which the trigger was attached.
- **OTYPE** is the type of object to which the trigger is attached; can be rsv, job, node, or sched.
- **OWNER** is the owner of the trigger's object (when applicable).
- **OWNERMAIL** is 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** is the time of the trigger launch.
- **USER** is the user (when applicable).

Other unique variables are available to triggers attached to specific objects:

**Jobs**

- **MASTERHOST** — The primary node for the job.
- **HOSTLIST** — The entire hostlist of the job.

**Reservations**

- **GROUPHOSTLIST** — The hostlist for the reservation and all of its peers. (In the case of VPCs, the aggregate hostlist for all VPC reservations.)
- **HOSTLIST** — The entire hostlist for the reservation.
- **OBJECTXML** — The XML representation of an object. Output is the same as that generated by `mdia -xml`.
- **OS** — The operating system on the first node of the reservation.

These reserved names cannot be used as variables. For example, if an OS variable is added to a
reservation and then accessed, it will contain information reported by the resource manager, not the value that the user inserted.

VPCs

- **VPCID** — The ID of the parent VPC.
- **VPCHOSTLIST** — The HostList for the entire VPC (which may or may not be dedicated).

By default, the reservation group master (first reservation created) of a VPC automatically imports the variables of the parent VPC. Other non-master reservation children of the VPC do not automatically import these VPC variables, and if this information is required by associated reservation triggers, it must be explicitly imported as described later.

Example

```
AType=exec,Action="/tmp/trigger3.sh $OID $HOSTLIST",EType=start
```

In this example, the object ID ($OID) and hostlist ($HOSTLIST) will be passed to /tmp/trigger3.sh as command line arguments when the trigger executes the script. The script can then process this information as needed.

### 19.5.3 Externally Injecting Variables into Job Triggers

For triggers that are attached to job objects, another method for supplying variables exists. The trigger is able to see the variables in the job object to which it is attached. Updating the job object’s variables effectively updates the variable for the trigger. This can be accomplished through the use of mjobctl using the `-m` flag.

```
> mjobctl -m var=Flag1=TRUE 1664
```

This sets the variable Flag1 to the value TRUE, creating Flag1, if necessary. This will be seen by any trigger attached to job 1664.

### 19.5.4 Exporting Variables to Parent Objects

Variables used and created by triggers are stored in the namespace of the object to which the trigger is attached. Sometimes it is desirable to make certain variables more accessible to triggers on other objects. When using the `Sets trigger attribute`, you can specify that a variable, created either by a success or failure, should be exported to the name space of the parent object when the current object is destroyed through a completion event. This is done by placing the caret (^) symbol in front of the variable name when specifying it.

Example

```
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
```

In this example, both Var1 and Var2 will be exported to the parent object when the trigger has completed. They can also be used by triggers at their own level, just as in previous examples.

### 19.5.5 Requiring Variables in Parent Objects

By default, triggers will only look for variables to fulfill dependencies in the object to which they are directly attached. In addition, 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 generations of parent objects, the caret (^) symbol must be specified, as in the following example:

Example
AType=exec,Action="/tmp/trigger2.sh",EType=start,Requires=^Var1

See Also

- Generic Metrics
- Generic Events
# 19.6 Trigger Examples

- **19.6.1 Trigger on a Standing Reservation**
  
  Create standing reservation `Mail2` with a trigger for the script `/tmp/email.sh`. Launch the script 200 seconds after the start of the reservation.

```
SRCFG[Mail2]
TRIGGER=EType=start,Offset=200,AType=exec,Action="/tmp/email.sh"
```

- **19.6.2 Job Trigger that Launches Script Prior to Wallclock Limit**
  
  Create a trigger associated with the job `job46` and 150 seconds before the job is scheduled to finish, launch the `/tmp/email.sh` script with the command line argument `Hello`.

```
> mschedctl -c trigger EType=end,offset=-150,AType=exec,Action="/tmp/email.sh Hello" -o Job:job46
```

- **19.6.3 Admin Reservation with Two Triggers**
  
  This example includes the reservation, the two scripts, and the output of the scripts.

```
> mrsvctl -c -h keiko 
-T "Sets=Var1.Var2,EType=start,AType=exec,Action="/tmp/trigs/trig1.sh ReservationStart NewReservation""
-T "Requires=Var1.Var2,EType=start,AType=exec,Action="/tmp/trigs/trig2.sh $Var1 stuff $Var2"

#!/bin/sh
echo -e "$1, $2
$*" >> /tmp/trigs/trig1
echo "Var1=1"
exit 0

#!/bin/sh
echo -e "$1, $2, $3
$*" > /tmp/trigs/trig2
```
The preceding example creates the following output:

```
exit 0
ReservationStart, NewReservation
ReservationStart NewReservation
1, stuff, 2
1 stuff 2
```

### 19.6.4 Launch a Local Script Each Time a Node Goes Down

This example places a trigger on all nodes and will fire when any node changes to a down state.

```
NODECFG[DEFAULT] TRIGGER=AType=exec,Action="/tmp/nodedown.sh $OID",EType=fail,MultiFire=TRUE,RearmTime=5:00
```

When any node changes its state to failure (from a non-failure state) Moab will execute the script `/tmp/nodedown.sh`. The `MultiFire` attribute means this trigger will fire every time a node fails and the `RearmTime` attribute means Moab will wait at least five minutes between firing the triggers in succession. (This example can be easily modified to trigger for only one node by replacing `DEFAULT` with the node name.)

The next example creates a diagnostic trigger that will fire when a node's state changes to down; then a second trigger fires based on the diagnostic information obtained by the first trigger.

```
NODECFG[DEFAULT] TRIGGER=atype=exec,action="/tmp/node_diagnostics.sh $OID",etype=fail,multifire=true,rearmtime=5:00,sets=OUTPUT
NODECFG[DEFAULT] TRIGGER=atype=exec,action="/tmp/node_recovery.sh $OID $OUTPUT",etype=fail,requires=OUTPUT,multifire=true,rearmtime=5:00,unset
```

In this example the first trigger will run the script "node_diagnostics.sh" and will set a variable called `OUTPUT`. The second trigger will use this `OUTPUT` information to decide what action to take.

### 19.6.5 Sending Email on Scheduler or Node Failure

This example places a mail trigger onto the scheduler that will fire whenever a failure is detected. In addition, the example configuration will also place a trigger on each compute node that will fire if the node goes down.

```
SCHEDCFG[MyCluster] TRIGGER=AType=mail,EType=fail,Action="scheduler failure detected on $TIME",MultiFire=TRUE,RearmTime=5:00
NODECFG[DEFAULT] TRIGGER=AType=mail,EType=fail,Action="node $OID has failed on $TIME",MultiFire=TRUE,RearmTime=15:00
```

... 

### 19.6.6 Resource Manager Failure Trigger

The `FAILTIME` is set on the resource manager, along with a failure trigger. This trigger will fire if the resource manager base goes down for more than three minutes.

```
RMCFG[base] TYPE=PBS FAILTIME=3:00
RMCFG[base] TRIGGER=atype=exec,action="/opt/moab/tools/diagnose_rm.pl $OID",etype=failure
```
19.6.7 Running the Support Script on Job-hold

The Support Diagnostic Script can be used to save a scheduler snapshot based on particular events. To save a system snapshot when the scheduler places a hold on a job, the following trigger can be configured:

```
CLASSCFG[batch]
JOBTRIGGER=atype=exec,etype=hold,action="$HOME/tools/support.diag.pl"
```

19.6.8 Creating a Periodic Standing Trigger

Standing triggers can be created on the scheduler object using the SCHEDCFG parameter as in the following example:

```
SCHEDCFG[base]
TRIGGER=atype=exec,name=jobsubmit_hour,etype=standing,action="/opt/moab$/bin/sh
echo nothing | msub -l
nodes=ALL,walltime=10:00,flags=NORMSTART:SYSTEMJOB:FRAGMENT,\
trig=atype=exec\&action=/tmp/diag.sh\$
\$HOSTLIST\&etype=start\&sets=DIAG,\
trig=atype=exec\&action=/tmp/step.sh\ $HOSTLIST\$
\$DIAG\&etype=start\&requires=DIAG\&sets=good.\!bad,\$
trig=atype=internal\&action=job:-:complete\&etype=start\&requires=good
```

This example will launch the createjobs_hour.pl script at 5 minutes past every hour and will run the createjobs_day.pl at 12:30 AM every morning.

19.6.9 Fragmenting a Diagnostic System Job Across the Entire Cluster

In this example a system job is submitted requesting a specific set of nodes. The job is submitted with the FRAGMENT flag, which will split the job up and run one job per distinct allocated host. Three triggers are then attached to each of those jobs:

1. Run a diagnostic script.
2. Run a recovery script.
3. If recovery script was successful, complete the system job.

```
#!/bin/sh
echo nothing | msub -l
nodes=ALL,walltime=10:00,flags=NORMSTART:SYSTEMJOB:FRAGMENT,\
trig=atype=exec\&action=/tmp/diag.sh\$
\$HOSTLIST\&etype=start\&sets=DIAG,\
trig=atype=exec\&action=/tmp/step.sh\ $HOSTLIST\$
\$DIAG\&etype=start\&requires=DIAG\&sets=good.\!bad,\$
trig=atype=internal\&action=job:-:complete\&etype=start\&requires=good
```

19.6.10 Successive Job Failure Trigger

Create a reservation on a node that has five successive job failures to block the node from running any other jobs until the problem is resolved.

```
NODECFG[DEFAULT] TRIGGER=atype=internal,action="node:=:reserve",etype=threshold,threshold=statistic[successivejobfailures]>5,
```

To reset the trigger and release the reservation, clear out the metric. This is done by either restarting Moab with mschedcti -R or modifying the reservation to grant yourself access and running a job inside of it.

19.6.11 Specifying an RSVPROFILE on an Internal Node Trigger
You can specify an RSVPROFILE on an internal node trigger reservation action.

```
NODECFG[DEFAULT] TRIGGER=atype=internal,action="node:-
:reserve:rsvprofile=<rsvprofile_name>"
```

### 19.6.12 Greater scheduling flexibility with migration internal actions

Two new internal actions (for kicking off overcommit and green migrations) can be attached to triggers, which allow for greater scheduling flexibility.

```
TRIGGER=atype=internal,etype=<whatever>,action=sched:-
:vmmigrate:[rsv|green|overcommit]
```

These three options still need to be enabled in moab.cfg and in moab.lic.

**See Also**

- [Generic Metrics](#)
- [Generic Events](#)
Virtual Private Clusters Overview

Within any organization, resources sometimes need to be set aside and configured for special projects or daily workflows. Requests may come from many different departments, each with its own needs and timetables. Set aside resources might include servers, databases, networks, or specialized hardware. Administrators can configure Moab to automatically service these requests, including configuration and provisioning, through Virtual Private Clusters (VPCs).

VPCs allow sites to group resources as a named package, which allows users to select the name of the package they need instead of requesting each resource individually. When a user requests a package, Moab schedules all of the resources required and lets the user know when the package is available for use.

Resources such as network, data, and security objects can be packaged together as a single VPC. Users must then just request the primary resource needed, such as a compute node, and other pre-configured resources such as storage and network are automatically added.

Additional setup and teardown steps can also be added to the VPC. Moab runs these extra steps automatically without the need for administrator action. Users request a package, wait for notification that the package is ready, and start using it while Moab, behind the scenes, takes care of scheduling, configuration and provisioning.

Administrators can place limits on profiles and restrict who is allowed to use them. Charging and accounting policies can also be configured to allow for automated billing.
20.1 Configuring VPC Profiles

- 20.1.1 Configuring VPC Profiles
- 20.1.2 Associating Resources with a VPC Profile
- 20.1.3 Using Reservation Profiles with VPCs
- 20.1.4 Coallocation of Additional Resources
- 20.1.5 Access Control
- 20.1.6 Charging for VPCs

20.1.1 Configuring VPC Profiles

Use the VCPROFILE parameter to configure a VPC profile. At its simplest, a VPC profile is a single line that gives the profile a name. The profile can then be requested by name and the user can fill in any other details required. The DESCRIPTION parameter is helpful, but not required. Users will be able to see the description in the listing of VPCs.

Example: A simple VPC with a description

```
VCPROFILE[example1]  DESCRIPTION="A Dedicated Compute Node"
```

The VPC can then be requested on the command line or through Moab Access Portal. Moab Access Portal provides a customizable portal where users can create, modify and manage VPCs. For more information, see Creating VPCs.

```
ACL
DESCRIPTION
NODEHOURCHARGERATE
NODESETLIST
NODESETUPCOST
OPRSVPROFILE
QUERYDATA
REQENDPAD
REQSETATTR
REQSTARTPAD
```

**ACL**

**Format:** List of zero or more comma delimited `<OTYPE>==<OID>[:<OID>]` pairs where `<OTYPE>` is one of the following:

- USER, GROUP, ACCT, CLASS, or QOS

**Default:** ---

**Details:** Specifies which credentials can view this profile and can create VPCs using this profile.

**Example:**

```
VCPROFILE[basic]  ACL=USER==steve:john,GROUP==staff
```

Users steve and john and any member of the group staff can use VPC resources.

**i** Inputting `USER==steve,USER==john` will only allow john to use the virtual cluster (as `==john overwrites ==steve`). For both to use resources, input `USER==steve,USER+=john`.

**DESCRIPTION**

**Format:** `<STRING>`

**Default:** ---

**Details:** Arbitrary human readable string describing VPC profile usage.
Example: VCPROFILE[dataset3] DESCRIPTION="custom 1 TB data mining environment with level 3 security"

End-users will see this description when specifying which VPC package to select.

**NODEHOURCHARGERATE**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;DOUBLE&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Details:</td>
<td>Per node chargerate multiplier associated with using resources and services in the specified VPC. See the Allocation Management overview for more information.</td>
</tr>
</tbody>
</table>

Example:

```
VCPROFILE[dataset1] NODEHOURCHARGERATE=1.5
VCPROFILE[dataset2] NODEHOURCHARGERATE=4.0
VCPROFILE[dataset3] NODEHOURCHARGERATE=6.0
```

Accounts associated with the creation of each data VPC will be charged for VPC utilization with the associated charge multiplier.

**NODESETLIST**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;FEATURE&gt;,&lt;FEATURE&gt;...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Details:</td>
<td>Nodeset constraints to be applied to explicitly specified reqs in format.</td>
</tr>
</tbody>
</table>

The VC profile NODESETLIST attribute only supports blocking node feature based node sets with the ONEOF policy enforced.

Example:

```
VCPROFILE[dataset3] NODESETLIST=fastio,myrinet
```

Resources will either be allocated entirely from the fastio or the myrinet node set.

**NODESETUPCOST**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;DOUBLE&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Details:</td>
<td>Per node cost in credits associated with provisioning resources and services to enable the associated VPC. See the Allocation Management overview for more information.</td>
</tr>
</tbody>
</table>

Example:

```
VCPROFILE[dataset1] NODESETUPCOST=20.5
VCPROFILE[dataset2] NODESETUPCOST=20.5
VCPROFILE[dataset3] NODESETUPCOST=65.0
```

Accounts associated with the creation of a new VPC will be assessed a per-node charge for each
node in a newly created VPC associated with the overhead of setting up the needed environment.

<table>
<thead>
<tr>
<th><strong>OPRSVPORFILE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
</tbody>
</table>

Resource requirements that are explicitly specified by the requestor and that do not have an explicit reservation profile will use the *fastp* reservation profile.

<table>
<thead>
<tr>
<th><strong>QUERYDATA</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
</tbody>
</table>

One *inode* will be allocated for the first 30 minutes of the VPC and one *vlan* generic resource will be allocated for the first hour.

*If a QUERYDATA attribute has a nodefeature constraint of $common, the co-allocation query will import the nodefeature constraint of the first explicitly specified requirement.*

<table>
<thead>
<tr>
<th><strong>REQENDPAD</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
</tbody>
</table>

Five minutes of additional walltime will be appended to the end of each explicitly specified requirement.

<table>
<thead>
<tr>
<th><strong>REQDEFFATTR</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>VCPROFILE[dataset3] OPRSVPROFILE=fastp</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>VCPROFILE[pkgA] QUERYDATA=minprocs=1,nodefeature=ionodes,duration=00:30:00 VCPROFILE[pkgA] QUERYDATA=mintasks=1,gres=vlan:1,duration=01:00:00</td>
</tr>
</tbody>
</table>

One *inode* will be allocated for the first 30 minutes of the VPC and one *vlan* generic resource will be allocated for the first hour.
Indicates a database profile with a default duration of one day. User can specify a shorter or longer duration when the VPC is requested.

**Example:**

```plaintext
VCPROFILE[database]  REQDEFATTR=duration=1:00:00:00
```

A request for four tasks and two generic resources named `tape` are appended to the user-specified resource request.

**Example:**

```plaintext
VCPROFILE[Large]  REQSETATTR=mintasks=4,gres=tape:2
```

Fifteen minutes of additional walltime will be prepended to the start of each explicitly specified requirement.

**Example:**

```plaintext
VCPROFILE[premium]  REQSTARTPAD=00:15:00
```

The reservation profiles of the VPC set with the **QUERYDATA** or **OPRSVPPROFILE** attributes can be used to manage provisioning of VPC resources. This provisioning is generally enabled via the use of triggers although it may also be enabled through a resource manager interface.

### 20.1.2 Associating Resources with a VPC Profile

Any number of resources or other attributes can be associated with a VPC profile. Examples include the number of nodes required, the duration of the VPC, or required node features. Default attributes that can be overridden when the VPC is requested are specified with **REQDEFATTR**. Required attributes that can not be overridden are specified with **REQSETATTR**.

For resources that need to be allocated once per request (coallocation), see Coallocation of Additional Resources.

**Example: Database server that will only run on specific nodes**

```plaintext
VCPROFILE[database]  DESCRIPTION="Database Server with Fast IO"
VCPROFILE[database]  REQSETATTR=nodefeature=fastio
VCPROFILE[database]  REQDEFATTR=duration=1:00:00:00
```

The preceding example indicates a database profile with a default duration of one day. The user can specify a shorter or longer duration when the VPC is requested. The database profile will only run on nodes with the fastio feature, which cannot be overridden by the user.
20.1.3 Using Reservation Profiles with VPCs

When a VPC is created, a reservation is created on the cluster. A reservation profile can be specified for the reservation. Setup and tear down actions, as well as other attributes, can be configured as part of the reservation profile. Reservation profiles are specified using the OPRSVPROFILE parameter.

Setup and tear down actions are specified using triggers. Triggers allow actions ranging from simple notification to complex workflows. If the setup time for the VPC is significant, the REQSTARTPAD and REQENDPAD parameters can be used to compensate the user for lost time. For actions that take place when the VPC is cancelled, a special variable (ACTIVE) can be passed to the trigger script. ACTIVE is set to the string True if the VPC is currently active, or Undefined if it is not. For more information see trigger variables.

Reservation profiles also allow a variety of flags to be specified. For more information see 7.1.2.2 Using Reservation Profiles.

Example: A VPC that provisions an operating system and logs information

```plaintext
VCPROFILE[test]   DESCRIPTION="Test Environment"
VCPROFILE[test]   OPRSVPROFILE=test
RSVPROFILE[test]  FLAGS=SYSTEMJOB
RSVPROFILE[test]  TRIGGER=ATYPE=exec,Action="$HOME/logdate.pl TEST START $VPCHOSTLIST $OID $HOSTLIST $ACTIVE",EType=start
RSVPROFILE[test]  TRIGGER=ATYPE=exec,Action="$HOME/installos.pl $HOSTLIST rhel",EType=start
RSVPROFILE[test]  TRIGGER=ATYPE=exec,Action="$HOME/logdate.pl TEST END $VPCHOSTLIST $OID $HOSTLIST $ACTIVE",EType=end
RSVPROFILE[test]  TRIGGER=ATYPE=exec,Action="$HOME/logdate.pl TEST CANCEL $VPCHOSTLIST $OID $HOSTLIST $ACTIVE",EType=cancel
```

In the preceding example, a logging script (logdate.pl) is called when the VPC starts, ends, or is cancelled. An operating system is installed on all nodes in the VPC when it starts.

20.1.4 Coallocation of Additional Resources

Additional units of other resources can also be packaged with VPC on a per instance basis. For example, if a user requests a four-node cluster, a head node (in addition to the four requested nodes) can be allocated automatically. This is done using the QUERYDATA parameter.

Example: Database profile with a single server and multiple client nodes

```plaintext
VCPROFILE[database]   DESCRIPTION="Database client/server package"
VCPROFILE[database]   QUERYDATA=minnodes=1,label=server,rsvprofile=dbserver
VCPROFILE[database]   OPRSVPROFILE=dbclient
```

In the preceding example, a database profile is configured that will give the user the number of database clients that they request (the dbserver profile), plus a single database server.

Example: LAMP stack profile with multiple parts

```plaintext
VCPROFILE[lampstack]   DESCRIPTION="LAMP stack"
VCPROFILE[lampstack]   OPRSVPROFILE=lampstack
VCPROFILE[lampstack]   NODEHOURCHARGERATE=6.6
VCPROFILE[lampstack]   NODESETUPCOST=66.6
VCPROFILE[lampstack]   FLAGS=AUTOACCESS

VCPROFILE[lampstack]   QUERYDATA=minnodes=1,label=apache_1b,rsvprofile=apache_1b
VCPROFILE[lampstack]   QUERYDATA=minnodes=1,label=apache_2b,rsvprofile=apache_2b
```

In the preceding example, additional resources are coallocated with the VPC.
In the preceding example, a profile is configured that makes a request for a LAMP stack that is comprised of an Apache load balancer, a pool of three Apache worker nodes, two nodes used to offload script-based computing, a database load balancer, a pool of two database nodes, and a database backup node.

The example shows how to handle two failure conditions. First, a shepherd trigger—a trigger that check the condition of a running job—is assigned to watch over the health of the lampstack in 5-minute intervals. If this trigger fails (returns -1), Moab sends an email to the administrator indicating the VPCID of the culprit VPC.

The second form of failure is illustrated using the db_pool example. The db_pool_zero script runs at VPC startup, resets the counter associated with the given VPCID (in this case stored in an SQLite3 database), and sets a variable allowing the second db_pool trigger to fire. If this trigger is successful, normal workflow conditions exist, and the rest of the VPC LAMP stack is created. If it fails, it sets a flag that the db_pool_retry script responds to. The db_pool_retry script increments the counter associated with the current VPCID, and resets the trigger variables to a state that allows the db_pool_ctl trigger to refire. If the db_pool_retry script gets called more than a prescribed number of times, this trigger fails, setting a variable that destroys the VPC.

20.1.5 Access Control

Administrators can specify the credentials that are allowed to create each VPC profile using the ACL parameter. The ACL parameter uses the Moab ACL syntax.

Example: A VPC that can only be created by members of the test group

VCPROFILE[test] DESCRIPTION="Testing Environment"
VCPROFILE[test] OPRSVPROFILE=test
VCPROFILE[test] ACL=GROUP:test

The preceding example specifies that only members of the test group may request the test profile. When a VPC is requested, an ACL can be specified for the reservation associated with the VPC. This allows users to give others access to their VPCs. This also allows users to submit batch jobs into the VPC that they have created. This capability can be used to allow trusted users to create limited reservations on the system and submit batch jobs into them. It also allows automatic charging for these reservations. If the AUTOACCESS flag is specified on the VPC, the owner of that VPC is automatically added to the ACL of its reservation.

Example: A profile that allows the owner and administrators to submit batch jobs into it

VCPROFILE[private] DESCRIPTION="Private cluster access"
VCPROFILE[private] REQSETATTR=nodefeature=dedicated,acl=group=administrators
VCPROFILE[private] FLAGS=AUTOACCESS
In the preceding example, the specified profile creates a reservation on the cluster that only administrators and the owner of the VPC may access. Only nodes with the dedicated feature may be reserved in this way.

**Example: A user creates a VPC and submits jobs into it**

To submit a job, the user specifies the VPC's reservation on the command line using `-1 advres`.

```bash
$ mshow -a -i -x -o --flags=tid,summary,future -p private -w
minnodes=1,duration=10000
Partition  Tasks  Nodes  Duration  StartOffset
StartDate  -----------  -----  -----  ------------  ----------
----------  -----  -----  ------------  ----------
-----
ALL         2      1       2:46:40      00:00:00
16:19:14_03/10  TID=13
ALL         2      1       2:46:40       2:46:40
19:05:54_03/10  TID=14
ALL         2      1       2:46:40       5:33:20
21:52:34_03/10  TID=15
$ mschedctl -c vpc -a resources=13
vpc.5
$ mschedctl -l vpc:vpc.5
VPC vpc.5 (active)  ---------
User:                  user1
Owner:                 user:user1
Size:                  2
Task Resources:        PROCS: [ALL]
Available Time:        16:19:14_03/10 (-00:01:56)
Available Duration:    2:46:40
Provision Start Time:  16:19:22_03/10 (-00:01:48)
Cleanup End Time:      19:05:54_03/10 (2:44:44)
PurgeTime:             19:05:54_03/11
Variables:             VPCHOSTLIST=node009;VPCID=vpc.5;
```

### 20.1.6 Charging for VPCs

VPC charging is configured with the `NODECHARGERATE` and `NODESETUPCOST` parameters.

**Example: Charging**

```
VCPROFILE[dbclient]  NODEHOURCHARGERATE=25.0  NODESETUPCOST=50.0
```

**See Also**

- `VCPROFILE` parameter
- `mschedctl` command
20.2 VPC Commands

- 20.2.1 List Profiles
- 20.2.2 Querying for Resources
- 20.2.3 Creating VPCs
- 20.2.4 Listing Current VPCs
- 20.2.5 Modifying VPCs
- 20.2.6 Destroying VPCs

20.2.1 List Profiles

mschedctl

$ mschedctl -l vpcprofile

VPCPROFILE[private] -------
Description: 'Private cluster access'
ReqSetAttrs: nodefeature=dedicated,acl=group=administrators
Flags: autoaccess

20.2.2 Querying for Resources

Use the mshow -a command to show available resources for the VPC, and to obtain a transactionID.

$ mshow -a -i -x -o --flags=tid,future,summary -p example1 -w

minnodes=1,duration=3000

<table>
<thead>
<tr>
<th>Partition</th>
<th>Tasks</th>
<th>Nodes</th>
<th>Duration</th>
<th>StartOffset</th>
<th>StartDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>2</td>
<td>1</td>
<td>00:50:00</td>
<td>00:00:00</td>
<td>13:53:15_02/24</td>
</tr>
<tr>
<td>TID=367</td>
<td>2</td>
<td>1</td>
<td>00:50:00</td>
<td>1:27:28</td>
<td>15:20:43_02/24</td>
</tr>
<tr>
<td>ALL</td>
<td>2</td>
<td>1</td>
<td>00:50:00</td>
<td>2:17:28</td>
<td>16:10:43_02/24</td>
</tr>
</tbody>
</table>

20.2.3 Creating VPCs

Use the mschedctl -c vpc command to create the virtual private cluster.

$ mschedctl -c vpc -a resources=367

vpc.79

20.2.4 Listing Current VPCs

mschedctl

$ mschedctl -l vpc

VPC vpc.81 (completed) -------
Owner: user:user1
Available Time: 16:20:58_03/02 (-21:35:48)
Available Duration: 2:46:40
20.2.5 Modifying VPCs

`mschedctl`

```
mschedctl -m vpc:<vpcid> <attribute>=<value>
```

20.2.6 Destroying VPCs

Destroy (cancel) VPCs via the `mschedctl -d vpc` command.

```
$ mschedctl -d vpc:vpc.82
```

Example: Delete all VPCs

```
$ mschedctl -d vpc:ALL
```

See Also

- `VCPROFILE` parameter
21.0 Miscellaneous

- 21.1 User Feedback Overview
- 21.2 Enabling High Availability Features
- 21.3 Identity Managers
- 21.4 Information Services for the Enterprise and Grid
- 21.5 Malleable Jobs
21.1 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. 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

```
FEEDBACKPROGRAM /opt/moab/tools/fb.pl
```
21.2 Enabling High Availability Features

- 21.2.1 Moab High Availability Overview
- 21.2.2 Configuring High Availability via a Networked File System
- 21.2.3 Confirming High Availability on a Networked File System
- 21.2.3 Other High Availability Configuration

21.2.1 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.

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.

Moab HA and TORQUE HA operate independently of each other. If a job is submitted with msub and the primary Moab 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. Below are some examples:

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.

21.2.2.1 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 parameter. Finally, the lock file is specified 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
```

FBSERVER does not take a port number. The primary server's port is used for both the primary server and the fallback server.

21.2.2.2 Confirming High Availability on a Networked File System

Adminstrators can run the mdiag -S -v command to view which Moab server is currently scheduling and responding to client requests.

21.2.3 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.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOBACTIONONNODEDEFailure</td>
<td>If a node allocated to an active job fails, it is possible for the job to</td>
</tr>
<tr>
<td></td>
<td>continue running indefinitely even though the output it produces is of no</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>SCHEDCFG[ ] RECOVERYACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>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 <strong>RESTART</strong>, <strong>TRAP</strong>, <strong>IGNORE</strong>, or <strong>DIE</strong>, as in the following example:</td>
</tr>
</tbody>
</table>

```
SCHEDCFG[bas] MODE=NORMAL RECOVERYACTION=RESTART
```
21.3 Identity Managers

- 21.3.1 Identity Manager Overview
- 21.3.2 Basic Configuration
- 21.3.3 Importing Credential Fairness Policies
- 21.3.4 Identity Manager Data Format
- 21.3.5 Identity Manager Conflicts
- 21.3.6 Refreshing Identity Manager Data
- 21.3.7 Exporting Data to Identity Managers
- 21.3.8 Creating External Credentials via an Identity Manager

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.

21.3.1 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 by which Moab obtains the needed information for a web service, text file, or database.

21.3.2 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.

<table>
<thead>
<tr>
<th>BLOCKCREDLIST</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> One or more comma delimited object types from the following list: <strong>acct</strong>, <strong>group</strong>, or <strong>user</strong></td>
</tr>
<tr>
<td><strong>Details:</strong> 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.</td>
</tr>
<tr>
<td>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:</td>
</tr>
<tr>
<td>FSPOLICY DEDICATEDPS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CREATECRED</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> <code>&lt;BOOLEAN&gt;</code> (default is FALSE)</td>
</tr>
</tbody>
</table>
| **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.

<table>
<thead>
<tr>
<th>CREATECREDURL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> <code>&lt;URL&gt;</code></td>
</tr>
<tr>
<td><strong>Details:</strong> Specifies the URL to use when creating a new credential.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REFRESHPERIOD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> <code>minute, hour, day, or infinity</code> <em>(default is infinity)</em></td>
</tr>
<tr>
<td><strong>Details:</strong> If specified, Moab refreshes identity manager information once every specified iteration. If <code>infinity</code> is specified, the information is updated only at Moab start up.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RESETCREDLIST</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> One or more comma delimited object types from the following list: <code>acct</code>, <code>group</code>, or <code>user</code></td>
</tr>
<tr>
<td><strong>Details:</strong> 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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SERVER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> <code>&lt;URL&gt;</code></td>
</tr>
<tr>
<td><strong>Details:</strong> Specifies the protocol/interface to use to contact the identity manager.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UPDATEREFRESHONFAILURE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> <code>&lt;BOOLEAN&gt;</code> <em>(default is FALSE)</em></td>
</tr>
<tr>
<td><strong>Details:</strong> 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.</td>
</tr>
</tbody>
</table>

IDIDCFG[info] SERVER=exec:///tmp/bad_script.pl REFRESHPERIOD=hour UPDATEREFRESHONFAILURE=TRUE

IDCFG[alloc] SERVER=exec://$TOOLSDIR/idquery.pl

---

21.3.3 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:

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.

21.3.4 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, acct, class, or qos.
- `<CREDID>` is the name of the credential.
- `<ATTR>` is one of adminlevel, alist, chargerate, comment, emailaddress, fstarget, gfstarget, gfsusage, maxjob, maxmem, maxnode, maxpe, maxproc, maxps, maxwc, plist, pref, priority, qlist, or role.
- `<VALUE>` is the value for the specified attribute.

To clear a comment, set its value to ""; for example: `comment=""`

Example

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
```

### 21.3.5 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.

### 21.3.6 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minute</td>
<td>update identity information once per minute</td>
</tr>
<tr>
<td>hour</td>
<td>update identity information once per hour</td>
</tr>
<tr>
<td>day</td>
<td>update identity information once per day</td>
</tr>
<tr>
<td>infinity</td>
<td>update identity information only at start-up (default)</td>
</tr>
</tbody>
</table>

Example

```
IDCFG[hq] SERVER=exec://$TOOLSDIR/updatepolicy.sh REFRESHPERIOD=hour
```

Job credential feasibility is evaluated at job submission and start time.

### 21.3.7 Exporting Data to Identity Managers

Local usage information can be exported to an identity manager. One possible use of this feature is for multiple clusters to export local usage to an identity manager and import global usage for usage and fairshare policies.
21.3.8 Creating External Credentials via an Identity Manager

To create or modify an external credential such as a user or group, the identity manager's CREDCREATEURL attribute must be specified. This URL can point to a database, a script, or a service and indicates the method to use to create a new external credential. If enabled, this method is called to create credentials on remote compute hosts if the credential is not currently defined on the master host. To enable Moab to automatically use this capability in a utility computing, grid, or cluster environment, the DYNAMICCRED flag must be set on the appropriate destination resource manager.

```
RMCFG[local] TYPE=PBS FLAGS=DYNAMICCRED
IDCFG[cred] CREATECREDURL=exec://$TOOLSDIR/user.create.nat.sh
```

One or more user.create.* tools may already exist in the $TOOLSDIR ($PREFIX/tools) directory. These can be used as is or customized appropriately for use in the local environment.

See Also

- 3.5 Credential Overview
- 6.2 Usage Limits/Throttling Policies
21.4 Information Services for Enterprises and Grids

Moab can be used to collect information from multiple scattered resources. Beyond information collection, Moab can also be set up to perform automated diagnostics, produce summary reports, and initiate automated resource recovery, event, and threshold based reprovisioning. Managed resources can include compute clusters, network resources, storage resources, license resources, system services, applications, and even databases.

21.4.1 General Collection Infrastructure

While significant flexibility is possible, a simple approach for monitoring and managing resources involves setting up a Moab Information Daemon (minfod) to access each of the resources to be monitored. These minfod daemons collect configuration, state, load, and other usage information and report it back to one or more central moab daemons. The central Moab is responsible for assembling this information, handling conflict resolution, identifying critical events, generating reports, and performing various automated actions.

The minfod daemon can be configured to import information from most existing HPC information sources, including both specialized application APIs and general communication standards. These interfaces include IPMI, Ganglia, SQL, Nagios, HTTP Services, Web/Soap based services, flat files, LSF, TORQUE/PBS, Loadleveler, SLURM, locally developed scripts, network routers, license managers, and so forth.

The information service feature takes advantage of the Moab peer-to-peer communication facility, identity management interface, generic event/metric facilities, generalized resource management infrastructure, and advanced accounting/reporting capabilities. With these technologies, solutions ranging from pure information services to more active systems that perform resource healing and automated load-balancing can be created.

With the flexibility of Moab, hybrid solutions anywhere along the active monitoring spectrum can be enabled. Services and resources associated with both open source/open standard protocols and vendor-specific protocols can be integrated and simultaneously managed by Moab. In real-time, the information gathered by Moab can be exported to a database, as HTML, or as a Web service. This flexibility allows the information to be of immediate use via human-readable and machine-readable interfaces.

21.4.2 Sample Uses

Organizations use this capability for multiple purposes including the following:

- Monitoring performance statistics of multiple independent clusters
- Detecting and diagnosing failures from geographically distributed clusters
- Tracking cluster, storage, network, service, and application resources
- Generating load-balancing and resource state information for users and middleware services

21.4.3 General Configuration Guidelines

1. Establish peer relationships between information service daemons (minfod or moab).
2. (optional) Enable Starttime Estimation Reporting if manual or automated load-balancing is to occur.
   - Set ENABLESTARTTESTIMATESTATS to generate local start estimation statistics.
   - Set REPORTPEERSTARTINFO to report start estimate information to peers.
3. (optional) Enable Generic Event/Generic Metric Triggers if automated resource recovery or alerts are to be used.
4. (optional) Enable automated periodic reporting.
5. (optional) Enable automated data/job staging and environmental translation.
6. (optional) Enable automated load/event based resource provisioning.

21.4.4 Examples

21.4.4.1 Grid Resource Availability Information Service
The objective of this project is to create a centralized service that can assist users in better utilizing geographically distributed resources within a loosely coupled-grid. In this grid, many independent clusters exist, but many jobs may only be able to use a portion of the available resources due to architectural and environmental differences from cluster to cluster. The information service must provide information to both users and services to allow improved decisions regarding job to resource mapping.

To address this, a centralized Moab information service is created that collects information from each of the participating grids. On each cluster where Moab is already managing the local workload, the existing cluster-level Moab is configured to report the needed information to the central Moab daemon. On each cluster where another system is managing local cluster workload, a Moab Information Service Daemon (minfod) is started.

Because load-balancing information is required, the Moab daemon running on each cluster is configured to report backlog and start estimate information using the REPORTPEERSTARTINFO parameter.

To make information available via a Web service, on the master Moab node, the cluster.mon.ws.pl service is started, allowing Moab to receive Web service based requests and report responses in XML over SOAP. To allow human-readable browser access to the same information and services, the local Web service is configured to use the moab.is.cgi script to drive the Web service interface and report results via a standard Web page.

Due to the broad array of users within the grid, many types of information are provided. This information includes the following:

- Per cluster configuration (operating system, architecture, node count, processor count, cumulative memory)
- Per cluster state (active, maintenance, down states)
- Per cluster messages (local admin-specified cluster messages)
- Per cluster usage (currently up and currently available node count, processor count, and cumulative memory)
- Per cluster backlog (in terms of processor seconds and estimated time to completion)
- Per cluster responsiveness matrix (job size/duration matrix of historical average queue time and xfactor)
- Per cluster starttime estimate matrix for generic workload (job size/duration matrix of estimated absolute and relative starttime for generic jobs based on priority, policy, backlog, reservation, system efficiency, resource failures, wallclock accuracy, and other factors)
- Per cluster starttime estimate for specific resource request (based on all factors listed plus job credentials and specific resource requests including memory, features, licenses, and so forth)
- Per cluster estimate accuracy statistics (indicate how accurate starttime estimates have been in the past)
- Adjusted starttime estimates (starttime estimates for both specific and generic job requests with estimate accuracy and composite estimate information integrated via an automated learning feedback algorithm)
- Best destination matrix for generic workload request (composite matrix representing best grid value and best target cluster for each cell)
- Prioritized best destination cluster report (list of potential destination clusters prioritized in order of best probable responsiveness first)

With these queries, users/services can obtain and process raw resource information or can ask a question as simple as What is the best cluster for this request?.
```
> mdiag -t -v
Partition Status

System Partition Settings: PList: clusterA,clusterB

<table>
<thead>
<tr>
<th>Name</th>
<th>Procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>1400</td>
</tr>
<tr>
<td>clusterA</td>
<td>800</td>
</tr>
<tr>
<td>RM=clusterA</td>
<td></td>
</tr>
<tr>
<td>clusterB</td>
<td>600</td>
</tr>
<tr>
<td>RM=clusterB</td>
<td></td>
</tr>
</tbody>
</table>

Partition | Configured | Up  | U/C | Dedicated | D/U |
----------|------------|-----|-----|-----------|-----|
| Active A/U|
| ALL      | 700        | 700 | 100.00% | 650 | 86.67% |
| 647      | 85.39%     |
| clusterA | 400        | 400 | 100.00% | 0   | 0.00%  |
| 0        | 0.00%      |
| clusterB | 300        | 300 | 100.00% | 1   | 100.00%|
| 1        | 100.00%    |

Processors

<table>
<thead>
<tr>
<th>Producers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
```

See Also

- Identity Managers
- Grid Basics
21.5 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., \( <\text{AverageJobEfficiency}> \ast <\text{AverageSystemUtilization}> \)).

Example

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
```
22.0 Database Configuration

Moab supports connecting to a database via native SQLite3, and it can also connect to other databases using the ODBC driver. 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.

- Moab must use an ODBC-compliant database to report statistics with Viewpoint reports.

  - 22.1 SQLite3
  - 22.2 Connecting to a MySQL Database with an ODBC Driver
22.1 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.
22.2 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 an 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 file versions:
- libmyodbc - 5.1.6
- unixodbc - 2.2.14

For a Debian-based system, unixodbc-dev is required, but it may not be for Red Hat flavors (such as CentOS and RHEL).

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`):

   ```
   # Turn on stat profiling
   USEDATABASE ODBC
   USERCFG[DEFAULT] ENABLEPROFILING=TRUE
   GROUPCFG[DEFAULT] ENABLEPROFILING=TRUE
   QOSCFG[DEFAULT] ENABLEPROFILING=TRUE
   CLASSCFG[DEFAULT] ENABLEPROFILING=TRUE
   ACCOUNTCFG[DEFAULT] ENABLEPROFILING=TRUE
   NODECFG[DEFAULT] ENABLEPROFILING=TRUE
   
   mysql -u root -p < moab-db.sql
   ``

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.sql
     ```

3. Configure the MySQL and ODBC driver. The `/etc/odbcinst.ini` file should contain content similar to what follows:

   ```
   [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. 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 = 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` user, `mypassword` password, and to connect to MySQL running on the localhost on port 3306. ODBC uses this information and connects 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 | ObjectTyoe |
| mcheckpoint | |
+---------------------------------------+

SQLRowCount returns 10
10 rows fetched
SQL>
```

If you encounter any errors using the `isql` command, then there were problems setting up the ODBC to MySQL connection.

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.

### 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 `$MOABHOMEDIR/etc/moab.cfg` file to connect to the appropriate database and reads the statistics files from `$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.
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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Location</th>
<th>Default</th>
<th>Max</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENTMAXCONNECTIONS</td>
<td>moab.cfg</td>
<td>128</td>
<td>512</td>
<td>Maximum number of connections that can simultaneously connect to Moab.</td>
</tr>
<tr>
<td>JOBMAXNODECOUNT</td>
<td>moab.cfg</td>
<td>1024</td>
<td>8192</td>
<td>Maximum number of compute nodes that can be allocated to a job. (Can also be specified within configure using --with-maxjobsize=&lt;NODECOUNT&gt;.)</td>
</tr>
<tr>
<td>MAXGRES</td>
<td>moab.cfg</td>
<td>24</td>
<td>1024</td>
<td>Total number of distinct generic resources that can be managed.</td>
</tr>
<tr>
<td>MAXJOB</td>
<td>moab.cfg</td>
<td>4096</td>
<td>50000</td>
<td>Maximum number of jobs that can be evaluated simultaneously. (Can also be specified within configure using --with-maxjobs=&lt;JOBCOUNT&gt;.)</td>
</tr>
<tr>
<td>MAXRSVPERNODE</td>
<td>moab.cfg</td>
<td>24</td>
<td>1024</td>
<td>Maximum number of reservations a node can simultaneously support.</td>
</tr>
<tr>
<td>MMAX_ATTR</td>
<td>moab.h</td>
<td>128</td>
<td>512</td>
<td>Total number of distinct node attributes (PBS node attributes/LL node features) that can be tracked.</td>
</tr>
<tr>
<td>MMAX_CLASS</td>
<td>moab.h</td>
<td>24</td>
<td>64</td>
<td>Total number of distinct job classes/queues available.</td>
</tr>
<tr>
<td>MMAX_FSDEPTH</td>
<td>moab.h</td>
<td>24</td>
<td>32</td>
<td>Number of active fairshare windows.</td>
</tr>
<tr>
<td>MMAX_NODE</td>
<td>moab.h</td>
<td>5120</td>
<td>16000</td>
<td>Maximum number of compute nodes supported. (Can be specified within configure using --with-maxnodes=&lt;NODECOUNT&gt;.)</td>
</tr>
<tr>
<td>MMAX_PAR</td>
<td>moab.h</td>
<td>16</td>
<td>16</td>
<td>Maximum number of partitions supported.</td>
</tr>
<tr>
<td>MMAX_QOS</td>
<td>moab.h</td>
<td>128</td>
<td>128</td>
<td>Total number of distinct QoS objects available to jobs.</td>
</tr>
<tr>
<td>MMAX_RACK</td>
<td>moab.h</td>
<td>200</td>
<td>200</td>
<td>Total number of distinct rack objects available within cluster.</td>
</tr>
<tr>
<td>MMAX_RANGE</td>
<td>moab.h</td>
<td></td>
<td></td>
<td>Total number of distinct timeframes evaluated.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMAX_RSV</td>
<td>Total number of distinct reservations allowed per cluster. (Can be specified within configure using <code>--with-maxrsv=&lt;RSVCOUNT&gt;</code>.)</td>
<td>1024</td>
<td>8192</td>
</tr>
<tr>
<td>MMAX_TASK</td>
<td>Total number of tasks allowed per job. (Can be specified within configure using <code>--with-maxtasks=&lt;TASKCOUNT&gt;</code>.)</td>
<td>4096</td>
<td>16000</td>
</tr>
</tbody>
</table>

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

**See Also**

- Appendix I: Considerations for Large Clusters
- 5.5 Task Distribution Policies
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.

- E.1 Authorization
  - E.1.1 Role Based Authorization Security Configuration
  - E.1.2 Host Based Authorization (Administrative Hosts)
- E.2 Authentication
  - E.2.1 Mauth Authentication
  - E.2.2 Munge Authentication
  - E.2.3 Server Response Control
  - E.2.4 Interface Development Notes
- E.3 Host Security
  - E.3.1 Minimal Host Security Enforcement
  - E.3.2 Medium Host Security Enforcement
  - E.3.3 Strict Host Security Enforcement
- E.4 Access Portal Security

E.1 Authorization

E.1.1 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.

E.1.1.1 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.

E.1.1.2 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.

E.1.1.3 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.

E.1.1.4 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:

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>changeparam</td>
<td>Change any scheduling policy or parameter (This command is deprecated. Use <code>mschedctl -m</code> instead).</td>
</tr>
<tr>
<td>checkjob</td>
<td>View detailed information for any job.</td>
</tr>
<tr>
<td>checknode</td>
<td>View detailed information for any node.</td>
</tr>
<tr>
<td>mbal</td>
<td>Perform real-time load-balancing of interactive commands.</td>
</tr>
<tr>
<td>mcredctl</td>
<td>View and modify credential attributes.</td>
</tr>
<tr>
<td>mdiag</td>
<td>Provide diagnostic reports for resources, workload, and scheduling.</td>
</tr>
<tr>
<td>mjobctl</td>
<td>Modify, control, and view jobs. Allows subcommand specification using subcommands <code>cancel</code> (you can also use <code>canceljob</code>), <code>checkpoint</code>, <code>diagnose</code>, <code>modify</code>, <code>query</code>, <code>requeue</code>, <code>resume</code>, <code>signal</code>, <code>submit</code>, <code>suspend</code>, <code>adjusthold</code>, and <code>adjustprio</code>.</td>
</tr>
<tr>
<td>mnodectl</td>
<td>Modify, control, and view nodes.</td>
</tr>
<tr>
<td>mrmctl</td>
<td>Modify, control, and view resource managers.</td>
</tr>
<tr>
<td>mrsvctl</td>
<td>Modify, control, and view reservations.</td>
</tr>
<tr>
<td>mschedctl</td>
<td>Modify, control, and view scheduler behavior.</td>
</tr>
<tr>
<td>mshow</td>
<td>View existing configuration and predicted resource availability.</td>
</tr>
<tr>
<td>showstats</td>
<td>View all scheduler and credential statistics.</td>
</tr>
<tr>
<td>releaseres</td>
<td>Release reservations (This command is deprecated. Use <code>mrsvctl -r</code> instead).</td>
</tr>
<tr>
<td>resetstats</td>
<td>Clear/reset all scheduler statistics.</td>
</tr>
<tr>
<td>runjob</td>
<td>Immediately execute any job (see <code>mjobctl -x</code>).</td>
</tr>
<tr>
<td>setqos</td>
<td>Set QoS on any job (This command is deprecated. Use <code>mjobctl -m</code> instead).</td>
</tr>
<tr>
<td>setres</td>
<td>Create a reservation (This command is deprecated. Use <code>mrsvctl -c</code> instead).</td>
</tr>
<tr>
<td>setspri</td>
<td>Set system priority on any job (This command is deprecated. Use <code>mjobctl -p</code> instead).</td>
</tr>
<tr>
<td>showconfig</td>
<td>Show all scheduler configuration parameters (This command is deprecated. Use <code>mschedctl -l</code> instead).</td>
</tr>
<tr>
<td>showres</td>
<td>Show detailed information for any reservation.</td>
</tr>
</tbody>
</table>

**E.1.1.5 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 feature.

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, including viewing cumulative and per job statistics, seeing job details, modifying job priorities and holds, cancelling and preempting jobs, and otherwise adjusting policies and constraints within the associated credential.

E.1.2 Host Based Authorization (Administrative Hosts)

If specified, the ADMINHOSTS parameter allows a site to specify a subset of trusted hosts. All administrative commands (level 1-3) will be rejected unless they are received from one of the hosts listed.

E.2 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.

E.2.1 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:

```
> vi /opt/moab/.moab.key
(insert key)
> cat /opt/moab/.moab.key
XXXXXXXX
# secure file by setting owner read-only permissions
> chmod 400 /opt/moab/.moab.key
# verify file is owned by root and permissions allow only root to read file
> ls -l /opt/moab/.moab.key
-r-------- 1 root root 15 2007-04-05 03:47 /opt/moab/.moab.key
```

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 authentication 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
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
```

### E.2.1.1 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 Gold 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Format</th>
<th>Default</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH</td>
<td>one of ADMIN1, ADMIN2, or ADMIN3</td>
<td>---</td>
<td>Specifies the level of control/information available to requests coming from this source/peer.</td>
<td>CLIENTCFG[RM:clusterB] AUTH=admin1 KEY=14335443</td>
</tr>
<tr>
<td>AUTHTYPE</td>
<td>one of DES, HMAC, HMAC64, or MD5.</td>
<td>DES</td>
<td>Specifies the encryption algorithm to use when generating the message checksum.</td>
<td>CLIENTCFG[AM:bio3] AUTHTYPE=HMAC64</td>
</tr>
<tr>
<td>HOST</td>
<td>STRING</td>
<td>---</td>
<td>Specifies the hostname of the remote peer. Peer requests coming from this host will be authenticated using the specified mechanism. This parameter is optional.</td>
<td>CLIENTCFG[RM:clusterA] HOST=orx.pb13.com KEY=banana6</td>
</tr>
<tr>
<td>KEY</td>
<td>STRING</td>
<td>---</td>
<td>Specifies the shared secret key to be used to generate the message checksum.</td>
<td>CLIENTCFG[RM:clusterA] KEY=banana6</td>
</tr>
</tbody>
</table>

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:bank]` 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.

### E.2.2 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.

### E.2.2.1 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 command, 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"
```

### E.2.3 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
```

### E.2.4 Interface Development Notes

Sample checksum generation algorithm code can be found in the Socket Protocol Description document.

### E.3 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.

#### E.3.1 Minimal Host Security Enforcement

For minimal host security, no additional configuration is required.

#### E.3.2 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` or `ps -js <SESSID>`.
- IPC (Inter-Process Communication)
  - Remove shared memory, semaphores, and message queues (use `ipcs/ipcrm`).
  - Remove named pipes.
- Network/Global Filesystem Access
  - Explicitly unmount user home and global file systems.
- Local Temporary Filesystems
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.

### E.3.3 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)

### E.4 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: 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, TRIGGER, PDEF, PLIST, QDEF, QLIST, usage limit, or a fairness usage limit specification (FSCAP, FSTARGET, and FSWEIGHT). | Default: --- | Details: 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: --- | Details: 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 | Details: Specifies the priority weight to be applied to the specified account priority. See Credential (CRED) Factor. | Example: ACCOUNTWEIGHT 100 |
| ADMIN1, ADMIN2, ADMIN3 | Format: Space delimited list of user names | Default: root | Details: Deprecated. Use ADMINCFG. Users listed under the parameter ADMIN1 are allowed to perform any scheduling function. They have full control over the scheduler and access to all data. The first user listed in the ADMIN1 user list is considered to be the 'primary admin' and is the ID under which Moab must be started and run. Valid values include user names or the keyword 'ALL'. Again, these parameters are deprecated; use ADMINCFG. |
All users listed have full access to Moab control commands and Moab data. Moab must be started by and run under the 'moabuser' user id since moabuser is the primary admin.

### ADMINCFG[X]

**Format:** One or more `<ATTR>=<VALUE>` pairs where `<ATTR>` is one of the following: `ENABLEPROXY`, `USERS`, `SERVICES`, or `NAME`

**Default:** ---

**Details:** 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'.

**Example:**

```
ADMINCFG[1] USERS=root,john
ADMINCFG[1] SERVICES=ALL
ADMINCFG[1] NAME=batchadmin

ADMINCFG[3] USERS=bob,carol,smoore
ADMINCFG[3] NAME=helpdesk
```

Members of the batchadmin admin role are allowed to run all commands. Members of the helpdesk role are allowed to cancel jobs, adjust job priority, and adjust job holds. They are also able to view and modify credential objects (ie, users, groups, accounts, etc) See the security overview for more details.

### ADMINHOSTS

**Format:** Space delimited list of host names.

**Default:** ---

**Details:** If specified, the ADMINHOSTS parameter allows a site to specify a subset of trusted hosts. All administrative commands (level 1-3) will be rejected unless they are received from one of the hosts listed.

**Example:**

```
ADMINHOSTS hostA hostB
```

### AGGREGATENODEACTIONS

**Format:** `<BOOLEAN>`

**Default:** FALSE

**Details:** 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

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;SECONDS&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>60</td>
</tr>
<tr>
<td>Details</td>
<td>The delay time for the <code>AGGREGATENODEACTIONS</code> parameter to aggregate requests before sending job batches.</td>
</tr>
<tr>
<td>Example</td>
<td><code>AGGREGATENODEACTIONSTIME 120</code></td>
</tr>
</tbody>
</table>

Sets the `AGGREGATENODEACTIONS` delay to two minutes..

### ALLOWROOTJOBS

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>FALSE</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies whether batch jobs from the root user (UID=0) are allowed to be execute. <strong>Note:</strong> The resource manager must also support root jobs.</td>
</tr>
<tr>
<td>Example</td>
<td><code>ALLOWROOTJOBS TRUE</code></td>
</tr>
</tbody>
</table>

Jobs from the root user can execute.

### ALLOWVMMIGRATION

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>FALSE</td>
</tr>
<tr>
<td>Details</td>
<td>Enables Moab to migrate VMs.</td>
</tr>
<tr>
<td>Example</td>
<td><code>ALLOWVMMIGRATION TRUE</code></td>
</tr>
</tbody>
</table>

### ALWAYSEVALUATEALLJOBS

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>FALSE</td>
</tr>
<tr>
<td>Details</td>
<td>When scheduling priority jobs, Moab stops scheduling when it encounters the first job that cannot run and cannot get a reservation. <strong>ALWAYSEVALUATEALLJOBS</strong> directs Moab to continue scheduling until all priority jobs (jobs that do not violate any limits) are evaluated.</td>
</tr>
<tr>
<td>Example</td>
<td><code>ALWAYSEVALUATEALLJOBS TRUE</code></td>
</tr>
</tbody>
</table>

### AMCFG

<table>
<thead>
<tr>
<th>Format</th>
<th>One or more <strong>key-value</strong> pairs as described in the Allocation Manager Configuration Overview.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### APPLICATIONLIST

**Format:** Space delimited list of generic resources.

**Default:** N/A

**Details:** Specifies which generic resources represent actual applications on the cluster/grid. See 12.4 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.

### ASSIGNVLANFEATURES

**Format:** `<BOOLEAN>`

**Default:** FALSE

**Details:** When set to TRUE, this forces all VMs to be contained in VLANs.

**Example:**

```
ASSIGNVLANFEATURES TRUE
```

If a VLAN is not requested when a VM is created, the VM is assigned to one.

### ATTRATTRWEIGHT

**Format:** `<INTEGER>`

**Default:** 0

**Details:** 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

**Details:** 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

**Details:** 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

**Details:** 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)

**Details:** 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.

### BACKFILLMETRIC

**Format:** One of the following: `PROCS`, `PROCSECONDS`, `SECONDS`, or `NODES`

**Default:** `PROCS`

**Details:** Specifies the criteria used by the backfill algorithm to determine the 'best' jobs to backfill. Only applicable when using BESTFIT or GREEDY backfill algorithms.

**Example:**
```
BACKFILLMETRIC PROCSECONDS
```

### BACKFILLPOLICY

**Format:** One of the following: `FIRSTFIT`, `BESTFIT`, `GREEDY`, `PREEMPT`, or `NONE`

**Default:** `FIRSTFIT`

**Details:** Specifies which backfill algorithm will be used. See Configuring Backfill for more information.

**Example:**
```
BACKFILLPOLICY BESTFIT
```
### BFCHUNKDURATION

**Format:** `[[[DD:]HH:]MM:]SS`

**Default:** 0 (chunking disabled)

**Details:** Specifies the duration during which freed resources will be aggregated for use by larger jobs. Used in conjunction with **BFCHUNKSIZE**. 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)

**Details:** Specifies the minimum job size which can utilize chunked resources. Used in conjunction with **BFCHUNKDURATION**. 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:** ---

**Details:** 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
```

### BFPRIOITYPOLICY

**Format:** One of **RANDOM**, **DURATION**, or **HWDURATION**

**Default:** ---

**Details:** Specifies policy to use when prioritizing backfill jobs for preemption

**Example:**
```
BFPRIOITYPOLICY DURATION
```

Use length of job in determining which backfill job to preempt.

### BFVIRTUALWALLTIMECONFLICTPOLICY

**Format:** One of the following: **PREEMPT**

---

---
<table>
<thead>
<tr>
<th>Option</th>
<th>Format</th>
<th>Default</th>
<th>Details</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Details</td>
<td>Specifies how to handle scheduling conflicts when a virtually scaled job &quot;expands&quot; to its original wallclock time. This occurs when the job is within one scheduling iteration - <code>RMPOLLINTERVAL</code> - of its virtually scaled wallclock time expiring.</td>
<td>BFVIRTUALWALLTIMECONFLICTPOLICY PREEMPT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BFVIRTUALWALLTIMESCALINGFACTOR</td>
<td>&lt;DOUBLE&gt;</td>
<td>0 (virtual scaling disabled)</td>
<td>Specifies the factor by which eligible jobs' wallclock time is virtually scaled (optimistic-like backfilling).</td>
<td>BFVIRTUALWALLTIMESCALINGFACTOR .4</td>
</tr>
<tr>
<td>Format</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Details</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLOCKLIST</td>
<td>DEPEND</td>
<td>NONE</td>
<td>Specifies the additional non-default criteria which are used to determine if a job should be filtered out before idle usage is updated and idle usage policies are enforced.</td>
<td>BLOCKLIST DEPEND</td>
</tr>
<tr>
<td>Format</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Details</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYPASSCAP</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Specifies the max weighted value allowed from the bypass count subfactor when determining a job's priority (see Priority Factors for more information).</td>
<td>BYPASSWEIGHT 5000 BYPASSCAP 30000</td>
</tr>
<tr>
<td>Format</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Details</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYPASSWEIGHT</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>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).</td>
<td>BYPASSWEIGHT 5000</td>
</tr>
<tr>
<td>Format</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Details</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHECKPOINTDIR</td>
<td>&lt;STRING&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Format</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Details</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>---</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Details:</strong></td>
<td>Specifies the directory for temporary job checkpoint files (usually of the form &quot;jobid.cp&quot;). This is <strong>not</strong> the directory for Moab's checkpoint file (.moab.ck).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td><code>CHECKPOINTEXPIRATIONTIME /tmp/moabcheckpoint</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CHECKPOINTEXPIRATIONTIME**

| **Format:** | `[[[DD::HH:]MM:]SS` or UNLIMITED |
| **Default:** | 3,000,000 seconds |
| **Details:** | 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 one day.

**CHECKPOINTFILE**

| **Format:** | <STRING> |
| **Default:** | moab.ck |
| **Details:** | 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 |
| **Details:** | Time between automatic Moab checkpoints. |
| **Example:** | `CHECKPOINTINTERVAL 00:15:00` 

Moab should checkpoint state information every 15 minutes.

**CHECKPOINTWITHDATABASE**

| **Format:** | <BOOLEAN> |
| **Default:** | FALSE |
| **Details:** | If set to TRUE, Moab stores checkpoint information to a database rather than to the .moab.ck flat text file. |
| **Example:** | `CHECKPOINTWITHDATABASE TRUE` 

**CHILDSTDERRCHECK**

| **Format:** | <BOOLEAN> |
| **Default:** | FALSE |
**Details:** 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, defaultGRES, default.MEM, default.NODESET, default.PROC, enableprofiing, EXCL.FEATURES, EXCLUDEUSERLIST, HOSTLIST, JOBEPILOG, JOBPROLOG, JOBTRIGGER, MAXPROCPERNODE, MAX.NODE, MAX.PROC, MAXPROC, MAX.WCLIMIT, MIN.NODE, MIN.PROC, MIN.TPN, MIN.WCLIMIT, PARTITION, PRIORITY, PRIORITYF, QDEF, QLIST, req.features, REQUIREDACCOUNTLIST, REQUIREDUSERLIST, RESFAILPOLICY, SYSPRIO, TRIGGER, WCOVERRUN, usage limit, or fairshare usage limit specification.

**Default:** ---

**Details:** 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

**Details:** 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:** ---

**Details:** 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 |
| Details: | 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 |
| Details: | Specifies how many times the client will attempt to retry its connection to the primary server if Moab is not available. |
| Example: | `CLIENTMAXPRIMARYRETRY 5`  
`CLIENTMAXPRIMARYRETRYTIMEOUT 1000`  
The client 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 2140000000 times. |

**CLIENTMAXPRIMARYRETRYTIMEOUT**

| Format: | `<integer>` (milliseconds) |
| Default: | 2 seconds |
| Details: | Specifies how much time to wait until the client will attempt to retry its connection to the primary server if Moab is not available. |
| Example: | `CLIENTMAXPRIMARYRETRY 3`  
`CLIENTMAXPRIMARYRETRYTIMEOUT 500`  
The client will attempt to retry its connection to the primary server 3 times with 1/2 second intervals before giving up. |

**CLIENTTIMEOUT**

| Format: | `[[[DD:]HH:]MM:]SS` |
| Default: | 00:00:30 |
| Details: | 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**

<p>| Format: | <code>TRUE</code> |
| Default: | <code>FALSE</code> |</p>
<table>
<thead>
<tr>
<th>Configuration</th>
<th>Details</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREDDISCOVERY</td>
<td>Specifies that Moab should create otherwise unknown credentials when it discovers them in the statistics files.</td>
<td>CREDDISCOVERY TRUE</td>
</tr>
</tbody>
</table>

**CREDWEIGHT**

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies the credential component weight associated with the credential priority. See Credential (CRED) Factor for more information.</td>
</tr>
<tr>
<td>Example</td>
<td>CREDWEIGHT 2</td>
</tr>
</tbody>
</table>

**DATASTAGEHOLDTYPE**

<table>
<thead>
<tr>
<th>Format</th>
<th>Any Job Hold type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>DEFER</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies what to do if a job's data staging operations fail.</td>
</tr>
<tr>
<td>Example</td>
<td>DATASTAGEHOLDTYPE BATCH</td>
</tr>
</tbody>
</table>

**DEADLINEPOLICY**

<table>
<thead>
<tr>
<th>Format</th>
<th>One of CANCEL, HOLD, IGNORE, or RETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>HOLD</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies what to do when a requested deadline cannot be reached (see Job Deadlines).</td>
</tr>
<tr>
<td>Example</td>
<td>DEADLINEPOLICY IGNORE</td>
</tr>
</tbody>
</table>

**DEFAULTCLASSLIST**

<table>
<thead>
<tr>
<th>Format</th>
<th>Space delimited list of one or more &lt;STRING&gt;'s.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>---</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies the default classes supported on each node for RM systems which do not provide this information.</td>
</tr>
<tr>
<td>Example</td>
<td>DEFAULTCLASSLIST serial parallel</td>
</tr>
</tbody>
</table>

**DEFAULTSUBMITLANGUAGE**

<table>
<thead>
<tr>
<th>Format</th>
<th>One of LSF, PBS, LL or SLURM.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>PBS</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies the default job language to use when interpreting commandline arguments and command file scripts associated with the msub command.</td>
</tr>
<tr>
<td>Example</td>
<td></td>
</tr>
</tbody>
</table>
### DEFAULTSUBMITPARTITION

**Format:** See parameter CLIENTCFG[].

**Default:** ---

**Details:** 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

**Details:** 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

**Details:** 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.

**Example:**
```
DEFERSTARTCOUNT 3
```

### DEFERTIME

**Format:** `[[[DD:]HH:]MM:]SS`

**Default:** 1:00:00

**Details:** 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.

**Example:**
```
DEFERTIME 0:05:00
```

### DELETESTAGEOUTFILES

**Format:** `<BOOLEAN>`

**Default:** FALSE

**Details:** 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
**Details:** 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 cancelled.
```

### DIRECTORYSERVER

**Format:** `<HOST>[:<PORT>]`
**Default:** ---
**Details:** Specifies the interface for the directory server.

**Example:**

```
DIRECTORYSERVER calli3.icluster.org:4702
```

### DISABLEEXCHLIST

**Format:** `<BOOLEAN>`
**Default:** FALSE
**Details:** 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
**Details:** Disallows interactive jobs submitted with `musb -I`.

**Example:**

```
DISABLEINTERACTIVEJOBS TRUE
```
**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.

### DISABLEREGEXCACHING

**Format:** `<BOOLEAN>`

**Default:** `FALSE`

**Details:** Turns off regular expression caching. Turning off regular expression caching preserves memory with hostlist reservations and speeds up start time.

**Example:**

```
DISABLEREGEXCACHING TRUE
```

### DISABLESAMECREDPREEMPTION

**Format:** Comma delimited list of one or more credentials: ACCT, CLASS, GROUP, QOS, or USER

**Default:** `---`

**Details:** This parameter prevents specified credentials from preempting its own jobs.

**Example:**

```
DISABLESAMECREDPREEMPTION QOS,USER
```

### DISABLESAMEQOSPREEMPTION

**Format:** `<BOOLEAN>`

**Default:** `FALSE`

**Details:** Specifies whether or not a preemptor job can preempt a preemptee job which possesses the same QoS.

**Example:**

```
DISABLESAMEQOSPREEMPTION TRUE
```

### DISABLESCHEDULING

**Format:** `<BOOLEAN>`

**Default:** `FALSE`

**Details:** 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`

**Details:** 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)

```
[tест@node04 moab-slurm]$ echo sleep 100 | msub
ERROR: cannot submit job from slave
```

**DISABLETHRESHOLDDRIGGERS**

**Format:** `<BOOLEAN>`

**Default:** FALSE

**Details:** This makes Moab not fire threshold-based triggers, but will log the intended action to the event logs. Similar to `DISABLEVMDECISIONS`.

**Example:**
```
DISABLETHRESHOLDDRIGGERS TRUE
```

**DISABLEVMDECISIONS**

**Format:** `<BOOLEAN>`

**Default:** FALSE

**Details:** 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 `DISABLETHRESHOLDDRIGGERS`.

**Example:**
```
DISABLEVMDECISIONS TRUE
```

**DISKWEIGHT**

**Format:** `<INTEGER>`

**Default:** 0

**Details:** 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 $10 \times 100 \times 12 \times 512$

**DISPLAYFLAGS**

**Format:** One or more of the following values (space delimited):

```
ACCOUNTCENTRIC, HIDEBLOCKED, NODECENTRIC, or USEBLOCKING
```

Default: ---

Details: Specifies flags that control how Moab client commands display various 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. **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

Details: 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

Details: If set to TRUE, Moab does not cancel interactive jobs that are held.

Example: DONTCANCELINTERACTIVEHJOBS TRUE

**DONTENFORCEPEERJOBLIMITS**

Format: <BOOLEAN>

Default: FALSE

Details: 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: ---

Details: 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).

### ENABLEFSVIOLATIONPREEMPTION

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>FALSE</td>
</tr>
<tr>
<td>Details</td>
<td>If set to <strong>TRUE</strong>, Moab will allow jobs within the same <em>class/queue</em> to <strong>preempt</strong> when the preemptee is violating a <em>fairshare</em> target and the preemptor is not.</td>
</tr>
<tr>
<td>Example</td>
<td>ENABLEFSVIOLATIONPREEMPTION TRUE</td>
</tr>
</tbody>
</table>

Moab can now accept hundreds of jobs per second using msub instead of 20-30.

### ENABLEHIGHTHROUGHPUT

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>FALSE</td>
</tr>
<tr>
<td>Details</td>
<td>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 (no job ID is returned). It is recommended that jobs be submitted with a &quot;-N &lt;JOBNAME&gt;&quot; argument so users can keep track of their jobs.</td>
</tr>
<tr>
<td>Example</td>
<td>ENABLEHIGHTHROUGHPUT TRUE</td>
</tr>
</tbody>
</table>

### ENABLEJOBARRAYS

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>True</td>
</tr>
<tr>
<td>Details</td>
<td>If set to <strong>TRUE</strong>, job arrays will be enabled.</td>
</tr>
<tr>
<td>Example</td>
<td>ENABLEJOBARRAYS TRUE</td>
</tr>
</tbody>
</table>

### ENABLEMULTIREQJOBS

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>FALSE</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies whether or not the scheduler will allow jobs to specify multiple independent resource requests (i.e., PBS jobs with resource specifications such as '-l nodes=3:fast+1:io').</td>
</tr>
<tr>
<td>Example</td>
<td>ENABLEMULTIREQJOBS TRUE</td>
</tr>
</tbody>
</table>

### ENABLENEGJOBPRIORITY

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
Details: If set to **TRUE**, the scheduler allows job priority value to range from \(-\text{INFINITY}\) to \(\text{MMAX\_Prio}\); otherwise, job priority values are given a lower bound of '1'. For more information, see `REJECTNEGPRIOJOBS`.

Example: 
```
ENABLENEGJOBPRIORITY TRUE
```

Job priority may range from \(-\text{INFINITY}\) to **MMAX\_Prio**.

---

**ENABLENODEADDRLOOKUP**

<table>
<thead>
<tr>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;BOOLEAN&gt;</code></td>
</tr>
</tbody>
</table>

Default: **FALSE**

Details: 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**

<table>
<thead>
<tr>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;BOOLEAN&gt;</code></td>
</tr>
</tbody>
</table>

Default: **FALSE**

Details: 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.

Example: 
```
ENABLEPOSUSERPRIORITY TRUE
```

Users may now specify positive job priorities and have them take effect (e.g. `msub -p 100 job.cmd`).

---

**ENABLESPVIOLATIONPREEMPTION**

<table>
<thead>
<tr>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;BOOLEAN&gt;</code></td>
</tr>
</tbody>
</table>

Default: **FALSE**

Details: 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
```

---

**ENABLESTARTESTIMATESTATS**

<table>
<thead>
<tr>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;BOOLEAN&gt;</code></td>
</tr>
</tbody>
</table>

Default: **FALSE**

Details: If set to **TRUE**, Moab will collect job start time estimation stats for all submitted jobs and will use those statistics to improve future estimates. Resource availability queries which can utilize this info can be made using the `showstart` command. Current estimation statistics can be viewed
using the `mdiaq -S` command. To flush current statistics, use the `mschedctl -f` command. **Note:** This parameter can be resource intensive so sites with large job queues (>10,000 jobs) may want to consider leaving this set to `FALSE`. See [Considerations for Large Clusters](#) for more details.

### ENFORCEACCOUNTACCESS

**Format:** `<BOOLEAN>`

**Default:** `FALSE`

**Details:** Specifies whether or not Moab will enforce account access constraints without an allocation manager.

**Example:**

```
ENFORCEACCOUNTACCESS TRUE
```

### ENFORCEGRESACCESS

**Format:** `<BOOLEAN>`

**Default:** `FALSE`

**Details:** If a user submits a job with a non-existent gres (e.g. in the case of a typo) and `ENFORCEGRESACCESS` 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
```

### ENFORCEINTERNALCHARGING

**Format:** `<BOOLEAN>`

**Default:** `FALSE`

**Details:** Specifies whether or not the scheduler will enforce internal credit tracking (see Internal Charging).

**Example:**

```
ENFORCEINTERNALCHARGING TRUE
```

### EVENTSERVER

**Format:** `<HOST>[:<PORT>]`

**Default:** `---`

**Details:** Specifies the interface for the event server.

**Example:**

```
EVENTSERVER calli3.icluster.org:4702
```

### FEATURENODETYPEHEADER

**Format:** `<STRING>`

**Default:** `---`
<table>
<thead>
<tr>
<th>Details:</th>
<th>Specifies the header used to specify node type via node features (i.e., LL features or PBS node attributes).</th>
</tr>
</thead>
</table>
| Example: | `FEATURENODETYPEHEADER xnt`  
Moab will interpret all node features with the leading string `xnt` as a nodetype specification - as used by Gold and other allocation managers, and assign the associated value to the node. i.e., `xntFast`. |

<table>
<thead>
<tr>
<th>FEATUREPARTITIONHEADER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
</tbody>
</table>
| **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`. |

<table>
<thead>
<tr>
<th>FEATUREPROCSPEEDHEADER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
</tbody>
</table>
| **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`. |

<table>
<thead>
<tr>
<th>FEATURERACKHEADER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
</tbody>
</table>
| **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**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;STRING&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Details:</td>
<td>Specifies the header used to extract node slot index via node features (i.e., LL features or PBS node attributes). <strong>Note:</strong> Adding a trailing <code>$</code> character will specify that only features with a trailing number be interpreted. For example, the header 'slot$' will match 'slot12' but not 'slotted'.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;STRING&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Details:</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Example:**

| FEEDBACKPROGRAM /var/moab/fb.pl |

Moab will run the specified program at the completion of each job.

---

**FILEREQUESTISJOBCENTRIC**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>FALSE</td>
</tr>
<tr>
<td>Details:</td>
<td>Specifies whether a job's file request is a total request for the job or a per task request.</td>
</tr>
</tbody>
</table>

**Example:**

| FILEREQUESTISJOBCENTRIC TRUE |

Moab will treat file requests as a total request per job.

---

**FILTERCMDFILE**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>TRUE</td>
</tr>
<tr>
<td>Details:</td>
<td>Running the <em>msub</em> command performs the following operations on the submission script:</td>
</tr>
<tr>
<td></td>
<td>- Replace all comments with spaces (excludes Resource Manager directives)</td>
</tr>
<tr>
<td></td>
<td>- Strip empty lines</td>
</tr>
<tr>
<td></td>
<td>- Replace \r with \n</td>
</tr>
<tr>
<td></td>
<td>- Lock job to a PBS resource manager if $PBS is found in the script</td>
</tr>
</tbody>
</table>

Include the **FILTERCMDFILE** parameter in the *moab.cfg* file that resides on the clients.
Example: FILTERCMDFILE FALSE

Running the `msub` command does not perform the actions detailed earlier.

### FORCERSVSUBTYPE

**Format:** <BOOLEAN>

**Default:** FALSE

**Details:** Specifies that admin reservations must have a subtype associated with them.

**Example:** FORCERSVSUBTYPE TRUE

Moab will require all admin reservations to include a subtype.

### FSACCOUNTWEIGHT

**Format:** <INTEGER>

**Default:** 0

**Details:** Specifies the weight assigned to the account subcomponent of the fairshare component of priority.

**Example:** FSACCOUNTWEIGHT 10

### FSCAP

**Format:** <DOUBLE>

**Default:** 0 (NO CAP)

**Details:** Specifies the maximum allowed absolute value for a job's total pre-weighted fairshare component.

**Example:** FSCAP 10.0

Moab will bound a job's pre-weighted fairshare component by the range +/- 10.0.

### FSCLASSWEIGHT

**Format:** <INTEGER>

**Default:** 0

**Details:** Specifies the weight assigned to the class subcomponent of the fairshare component of priority.

**Example:** FSCLASSWEIGHT 10

### FSDECAY

**Format:** <DOUBLE>
### FSDECAY

**Default:** 1.0  
**Details:** 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:**  
```
Moab will apply a decay rate of 0.8 to all fairshare windows.
```

### FSDEPTH

**Format:** `<INTEGER>`  
**Default:** 8  
**Details:** 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`  
**Details:** Fairshare priority will increase to target and stop.  
**Example:**  
```
FSENABLECAPPRIORITY TRUE
```

### FSGROUPWEIGHT

**Format:** `<INTEGER>`  
**Default:** 0  
**Details:**  
**Example:**  
```
FSGROUPWEIGHT 4
```

### FSINTERVAL

**Format:** `[[[DD:]HH:]MM:]SS`  
**Default:** `12:00:00`  
**Details:** Specifies the length of each fairshare window.  
**Example:**  
```
FSINTERVAL 12:00:00  
Track fairshare usage in 12 hour blocks.
```

### FSJPUWEIGHT

```
FSPOLICY  DEDICATEDPS  
FSDECAY  0.8  
FSDEPTH  8
```
**FSMOSTSPECIFICLIMIT**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>FALSE</td>
</tr>
<tr>
<td>Details:</td>
<td>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.</td>
</tr>
<tr>
<td>Example:</td>
<td>FSMOSTSPECIFICLIMIT TRUE</td>
</tr>
</tbody>
</table>

**FSPOLICY**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;POLICY&gt;[*] where &lt;POLICY&gt; is one of the following: DEDICATEDPS, DEDICATEDPES, UTILIZEDPS, PDEDICATEDPS, or SDEDICATEDPES.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Details:</td>
<td>Specifies the unit of tracking fairshare usage. <strong>DEDICATEDPS</strong> tracks dedicated processor seconds. <strong>DEDICATEDPES</strong> tracks dedicated processor-equivalent seconds. <strong>UTILIZEDPS</strong> tracks the number of utilized processor seconds. <strong>SDEDICATEDPES</strong> tracks dedicated processor-equivalent seconds scaled by the speed of the node. <strong>PDEDICATEDPS</strong> tracks dedicated processor seconds scaled by the processor speed of the node. If the optional '%' (percentage) character is specified, percentage based fairshare will be used. See Fairshare Overview and Fairshare Consumption Metrics or more information.</td>
</tr>
<tr>
<td>Example:</td>
<td>FSPOLICY DEDICATEDPES</td>
</tr>
</tbody>
</table>

Moab will track fairshare usage by dedicated process-equivalent seconds.

**FSPPUWEIGHT**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>Details:</td>
<td>Specifies the fairshare weight assigned to processors per user.</td>
</tr>
<tr>
<td>Example:</td>
<td>FSPPUWEIGHT 10</td>
</tr>
</tbody>
</table>

**FSPSPUWEIGHT**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>Details:</td>
<td>Specifies the fairshare weight assigned to processor-seconds per user.</td>
</tr>
</tbody>
</table>

**FSJPUWEIGHT** 10
### Example:

| Example: | FSPSPUWEIGHT 10 |

### FSQOSWEIGHT

**Format:** `<INTEGER>`

**Default:** 0

**Details:** Specifies the priority weight assigned to the QOS fairshare subcomponent.

**Example:** FSQOSWEIGHT 16

### FSTARGETISABSOLUTE

**Format:** `<BOOLEAN>`

**Default:** FALSE

**Details:** 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:** ---

**Details:** Specifies the share tree distribution for job fairshare prioritization (see Hierarchical Share Tree Overview).

**Example:** FSTREE[geo] SHARES=16 MEMBERLIST=geo103,geo313,geo422

### FSTREEACL_POLICY

**Format:** `OFF, PARENT, or FULL`

**Default:** FULL

**Details:** Specifies how Moab should interpret credential membership when building the FSTREE (see Hierarchical Share Tree Overview).

**Example:** FSTREEACL_POLICY PARENT

Credentials will be given access to their parent node when applicable.

### FSTREEISREQUIRED

**Format:** `<BOOLEAN>`

**Default:** FALSE

**Details:** 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.

---

### FSTREEISREQUIRED

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>FALSE</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies whether the user must be given explicit access to a branch in the FSTREE (see Hierarchical Share Tree Overview).</td>
</tr>
</tbody>
</table>

**Example:**

```
FSTREEISREQUIRED TRUE
```

Users must be given explicit access to FSTREE nodes in order to gain access to the FSTREE.

---

### FSUSERWEIGHT

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>0</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies the priority weight assigned to the user fairshare subfactor.</td>
</tr>
</tbody>
</table>

**Example:**

```
FSUSERWEIGHT 8
```

---

### FSWEIGHT

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies the priority weight assigned to the summation of the fairshare subfactors (see Priority Factor and Fairshare overviews).</td>
</tr>
</tbody>
</table>

**Example:**

```
FSWEIGHT 500
```

---

### GEVENTCFG[<GEVENT>]

<table>
<thead>
<tr>
<th>Format</th>
<th>List of zero or more space delimited &lt;ATTR&gt;=&lt;VALUE&gt; pairs where &lt;ATTR&gt; is ACTION, ECOUNT, REARM, or SEVERITY. See Responding to Generic Events for details on values you can assign to each attribute.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>---</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies how the scheduler should behave when various cluster events are detected. See the Generic Events Overview for more information.</td>
</tr>
</tbody>
</table>

**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...
### GRESCFG[<GRES>]

**Format:** List of zero or more space delimited `<ATTR> = <VALUE>` pairs where `<ATTR>` is one of the following: `TYPE` or `STARTDELAY`.

**Default:** ---

**Details:** Specifies associations of generic resources into resource groups. 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`.

### GRESTOJOBATTRMAP

**Format:** Comma delimited list of generic resources

**Default:** ---

**Details:** The list of generic resources will also be interpreted as JOB features. See 7.1.5 Managing Reservations.

**Example:**

```
GRESTOJOBATTRMAP 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`, `TRIGGER`, `QLIST`, `QDEF`, `PLIST`, `PDEF`, `FLAGS`, `usage limits`, or `a fairshare usage limit specification`.

**Default:** ---

**Details:** 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

**Details:** Specifies the priority weight assigned to the specified group priority (See Credential (CRED) Factor).
### GUARANTEEDPREEMPTION

**Format:** `<BOOLEAN>`  
**Default:** FALSE  
**Details:** Moab locks preemptors onto preempted nodes for `JOBRETRYTIME`. When `JOBRETRYTIME` expires, Moab attempts to run the job elsewhere.  
**Example:**  
```
GUARANTEEDPREEMPTION TRUE
```

### HAPOLLINTERVAL

**Format:** `[[DD:]HH:]MM:]SS`  
**Default:** 30  
**Details:** Specifies the amount of time between subsequent high availability pings. See High Availability Overview for more info.  
**Example:**  
```
HAPOLLINTERVAL 00:00:15
```

The Moab fallback server will check the health of the Moab primary every 15 seconds.

### HIDEVIRTUALNODES

**Format:** `<BOOLEAN>`  
**Default:** ---  
**Details:** 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`, `CREATECREDLURL`, `REFRESHPERIOD`, `RESETCREDLIST`, `SERVER` or `UPDATEREFRESHONFAILURE`.  
**Default:** ---  
**Details:** This parameter enables the identity manager interface allowing credential, policy, and usage information to be shared with an external information service.  
**Example:**  
```
IDCFG[info] SERVER=exec:///opt/moab/dbquery.pl REFRESHPERIOD=hour
```

Moab will refresh credential info every hour using the specified script.

### IGNOREMDATASTAGING

**Format:** `<BOOLEAN>`  
**Default:** FALSE  
**Details:** 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:**

<table>
<thead>
<tr>
<th>IGNORERMDATASTAGING TRUE</th>
</tr>
</thead>
</table>

### IGNORECLASSES

**Format:**  

![CLASS][,<CLASS>]

**Default:** ---

**Details:** 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:**

<table>
<thead>
<tr>
<th>IGNORECLASSES dque,batch</th>
</tr>
</thead>
</table>

Moab will ignore jobs from classes dque and batch.

### IGNOREJOBS

**Format:**  

![JOBID][,<JOBID>]

**Default:** ---

**Details:** 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:**

<table>
<thead>
<tr>
<th>IGNOREJOBS !14221,14223</th>
</tr>
</thead>
</table>

Moab will ignore jobs all classes except 14221 and 14223.

### IGNORENODES

**Format:**  

![NODE][,<NODE>]

**Default:** ---

**Details:** 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:**

<table>
<thead>
<tr>
<th>IGNORENODES !host3,host4</th>
</tr>
</thead>
</table>

Moab will only process nodes host3 and host4.

### IGNOREPREEMPTEEPRIORITY

**Format:**  

<BOOLEAN>

**Default:** FALSE

**Details:** By default, preemtter jobs can only preempt preemttee jobs if the preemtter has a higher job priority than the preemttee. When this parameter is set to true, the priority constraint is
removed allowing any preemtorto preempt any preemptee.

Example:  
**IGNOREPREEMTTEEPRIORITY TRUE**

All preemtort job can preempt any preemptee jobs.

---

**IGNOREUSERS**

<table>
<thead>
<tr>
<th>Format:</th>
<th>[!]&lt;USERNAME&gt;[,&lt;USERNAME&gt;]...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Details:</td>
<td>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 <em>not</em> (i.e., '!') character is specified, only jobs from listed users are processed. (See the Moab Side-by-Side Analysis for more information.)</td>
</tr>
</tbody>
</table>

Example:  
**IGNOREUSERS testuser1,annapolis**

Moab will ignore jobs from users testuser1 and annapolis.

---

**#INCLUDE**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;STRING&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Details:</td>
<td>Specifies another file which contains more configuration parameters. If &lt;STRING&gt; is not an absolute path, Moab will search its home directory for the file.</td>
</tr>
</tbody>
</table>

Example:  
**#INCLUDE moab.acct**

Moab will process the parameters in moab.acct as well as moab.cfg

---

**INSTANTSTAGE**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>FALSE</td>
</tr>
<tr>
<td>Details:</td>
<td>Specifies whether Moab should instantly stage jobs to the underlying resource manager when a job is submitted through msub.</td>
</tr>
</tbody>
</table>

Example:  
**INSTANTSTAGE TRUE**

---

**INVALIDFSTREEMSG**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&quot;&lt;STRING&gt;&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>&quot;no valid fstree node found&quot;</td>
</tr>
<tr>
<td>Details:</td>
<td>Specifies the error message that should be attached to jobs that cannot run because of a fairshare tree configuration violation.</td>
</tr>
</tbody>
</table>

Example:  
**INVALIDFSTREEMSG "account is invalid for requested partition"**
### JOBACTIONONNODEFAILURE

**Format:** CANCEL, FAIL, HOLD, IGNORE, NOTIFY, or **REQUEUE**

**Default:** ---

**Details:** 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.

### JOBACTIONONNODEFAILUREDURATION

**Format:** <INTEGER>

**Default:** 300

**Details:** Specifies how long a node must be marked as down before Moab performs the failure action. It eliminates failure actions resulting from short-lived network failures or glitches that cause nodes to appear down briefly when they are functional.

**Example:**

```
JOBACTIONONNODEFAILUREDURATION 60
```

Moab waits for a node to be down for 60 seconds before performing the failure action.

### JOBAGGREGATIONTIME

**Format:** [[[DD:]:HH:]MM:]SS

**Default:** 0

**Details:** 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 00:00:30
```

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, REQFEATURES, ROPSYS, or SELECT**
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, JOBSKRIT MEM (for MEM=<value>), MEMORY (for MEMORY=$LEARN), NODEACCESSPOLICY, NODEMOD, PARTITION, PREF, QOS, RESTARTABLE, RM, RMSERVICEJOB, SELECT, STAGEIN, 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 the job reported by one or more workload queries. See Wiki Attributes and Job Template Extensions.

Example:
JOBCFG[sql] REQFEATURES=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
Details: Specifies the amount of time Moab will preserve detailed information about a completed job (see JOBPURGETIME, showq -c and checkjob).

Example:
JOBCPURGETIME 02:00:00

Moab will maintain detailed job information for two hours after a job has completed.

JOBCTRUNCATENLCP
Format: <BOOLEAN>
Default: TRUE
Details: Specifies whether Moab will store only the first node of the node list for a completed job in the checkpoint file.

Example:
JOBCTRUNCATENLCP TRUE

JOBCTRUNCATENLCP reduces the amount of memory Moab uses to store completed job information.

JOBEXTENDSTARTWALLTIME
### JOBEXTENDSTARTWALLTIME

**Format:** <BOOLEAN>

**Default:** ---

**Details:** 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).

**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

**Details:** 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:** ---

**Details:** 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:** ---
| **Details:** | 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:** | 
| **Default:** | 
| **Details:** | 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 |
| **Details:** | 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 affect. The command mdiag -S will indicate if any job node count overflows have occurred. See Consideration for Large Clusters. |
| **Example:** | JOBMAXNODECOUNT 4000 |

### JOBMAXOVERRUN

| **Format:** | 
| **Default:** | (no soft limit), 10 minutes (hard limit) |
| **Details:** | 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 Resource Usage Limit Overview. |
| **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) |
| **Details:** | Maximum number of jobs allowed to be preempted per iteration. |
| **Example:** | JOBMAXPREEMPTPERITERATION 10 |
**JOBMAXSTARTPERITERATION**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>0 (No Limit)</td>
</tr>
<tr>
<td>Details:</td>
<td>Maximum number of jobs allowed to start per iteration.</td>
</tr>
<tr>
<td>Example:</td>
<td>JOBMAXSTARTPERITERATION 10</td>
</tr>
</tbody>
</table>

**JOBMAXSTARTTIME**

<table>
<thead>
<tr>
<th>Format:</th>
<th>[[DD:]HH:]MM:SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>-1 (NO LIMIT)</td>
</tr>
<tr>
<td>Details:</td>
<td>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.</td>
</tr>
<tr>
<td>Example:</td>
<td>JOBMAXSTARTTIME 2:00:00</td>
</tr>
</tbody>
</table>

Jobs may attempt to start for up to 2 hours before being canceled by the scheduler.

**JOBMIGRATEPOLICY**

<table>
<thead>
<tr>
<th>Format:</th>
<th>One of the following: IMMEDIATE, JUSTINTIME, or AUTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>AUTO</td>
</tr>
<tr>
<td>Details:</td>
<td>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.</td>
</tr>
<tr>
<td>Example:</td>
<td>JOBMIGRATEPOLICY JUSTINTIME</td>
</tr>
</tbody>
</table>

**JOBNAMEWEIGHT**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Details:</td>
<td>Specifies the weight to be applied to the job's name if the Name contains an integer. See Attribute (ATTR) Factor.</td>
</tr>
<tr>
<td>Example:</td>
<td>JOBNAMEWEIGHT 1</td>
</tr>
</tbody>
</table>

**JOBNODEMATCHPOLICY**

<table>
<thead>
<tr>
<th>Format:</th>
<th>EXACTNODE or EXACTPROC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
</tbody>
</table>
| Details:         | 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.

<table>
<thead>
<tr>
<th>JOBPREEMPTMAXACTIVETIME</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> [[DD:]HH:]MM:SS</td>
</tr>
<tr>
<td><strong>Default:</strong> 0</td>
</tr>
<tr>
<td><strong>Details:</strong> The amount of time in which a job may be eligible for preemption. See Job Preemption.</td>
</tr>
<tr>
<td><strong>Example:</strong> JOBPREEMPTMAXACTIVETIME 00:05:00</td>
</tr>
</tbody>
</table>

A job is preemptable for the first 5 minutes of its run time.

<table>
<thead>
<tr>
<th>JOBPREEMPTMINACTIVETIME</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> [[DD:]HH:]MM:SS</td>
</tr>
<tr>
<td><strong>Default:</strong> 0</td>
</tr>
<tr>
<td><strong>Details:</strong> The minimum amount of time a job must be active before being considered eligible for preemption. See Job Preemption.</td>
</tr>
<tr>
<td><strong>Example:</strong> JOBPREEMPTMINACTIVETIME 00:05:00</td>
</tr>
</tbody>
</table>

A job must execute for 5 minutes before Moab will consider it eligible for preemption.

<table>
<thead>
<tr>
<th>JOBPRIOACCRUALPOLICY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> ACCRUE or RESET</td>
</tr>
<tr>
<td><strong>Default:</strong> ACCRUE</td>
</tr>
<tr>
<td><strong>Details:</strong> Specifies how Moab should track the dynamic aspects of a job's priority. <strong>ACCRUE</strong> 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. <strong>RESET</strong> indicates that it will accrue priority from the time it is submitted unless it violates any of the JOBPRIOEXCEPTIONS. However, with <strong>RESET</strong>, if the job does violate JOBPRIOEXCEPTIONS then its queuetime based priority will be reset to 0.</td>
</tr>
</tbody>
</table>

Note: 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**

<table>
<thead>
<tr>
<th>Example:</th>
<th>JOBPRIOACCUALPOLICY</th>
<th>RESET</th>
</tr>
</thead>
</table>

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 |

**Details:** 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 inspite 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.

| 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: | --- |

**Details:** 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.

<table>
<thead>
<tr>
<th>Example:</th>
<th>JOBPRIOF</th>
<th>STATE[Running]=100</th>
<th>STATE[Suspended]=1000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ATTR[PREEMPTEE]=200</td>
<td>GRS[blocalc]=5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATTRATTRWEIGHT</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATTRSTATEWEIGHT</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

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) |

**Details:** 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

**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. (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:**

```text
JOBREJECTPOLICY MAIL,CANCEL
```  

**JOBREMOVEENVVARLIST**

**Format:** Comma-delimited list of strings

**Default:** ---

**Details:** 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:**

```text
JOBREMOVEENVVARLIST PBS_SERVER,TZ
```  

**JOBRETRYTIME**

**Format:** [[[DD:]HH:]MM:]SS

**Default:** 60 seconds

**Details:** 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. See Reservation Policies, RESERVATIONRETRYTIME, and JOBRETRYTIME for more information.

**Example:**

```text
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.
### JOBSYNCTIME
**Format:** `[[[DD:]HH:]MM:]SS`
**Default:** 00:10:00
**Details:** Specifies the length of time after which Moab will synchronize a job's expected state with an unexpected reported state. **IMPORTANT Note:** Moab will not allow a job to run while its expected state does not match the state reported by the resource manager.
**Example:** JOBSYNCTIME 00:01:00

### LIMITEDJOBCP
**Format:** `<BOOLEAN>`
**Default:** FALSE
**Details:** Specifies whether there should be limited job checkpointing (see Consideration for Large Clusters).
**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
**Details:** 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
**Details:** 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

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;STRING&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>Details</td>
<td>Specifies the path for the lock (pid) file used by Moab.</td>
</tr>
<tr>
<td>Example</td>
<td>LOCKFILE /var/spool/moab/lock</td>
</tr>
</tbody>
</table>

### LOGDIR

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;STRING&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>log</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies the directory in which log files will be maintained. If specified as a relative path, <strong>LOGDIR</strong> will be relative to $(MOABHOMEDIR) See Logging Overview for more information.</td>
</tr>
<tr>
<td>Example</td>
<td>LOGDIR /var/spool/moab</td>
</tr>
</tbody>
</table>

Moab will record its log files directly into the /var/spool/moab directory

### LOGFACILITY

<table>
<thead>
<tr>
<th>Format</th>
<th>Colon delimited list of one or more of the following: fCORE, fSCHED, fSOCK, fUI, fLL, fCONFIG, fSTAT, fSIM, fSTRUCT, fFS, fCKPT, fBANK, fRM, fPBS, fWIKI, fALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>fALL</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies which types of events to log (see Logging Overview).</td>
</tr>
<tr>
<td>Example</td>
<td>LOGFACILITY fRM:fPBS</td>
</tr>
</tbody>
</table>

Moab will log only events involving general resource manager or PBS interface activities.

### LOGFILE

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;STRING&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>moab.log</td>
</tr>
<tr>
<td>Details</td>
<td>Name of the Moab log file. This file is maintained in the directory pointed to by &lt;LOGDIR&gt; unless &lt;LOGFILE&gt; is an absolute path (see Logging Overview)</td>
</tr>
<tr>
<td>Example</td>
<td>LOGFILE moab.test.log</td>
</tr>
</tbody>
</table>

Log information will be written to the file moab.test.log located in the directory pointed to by the **LOGDIR** parameter.

### LOGFILEMAXSIZE

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>10000000</td>
</tr>
<tr>
<td>Details</td>
<td>Maximum allowed size (in bytes) of the log file before it will be rolled (see Logging Overview).</td>
</tr>
</tbody>
</table>

Moab will record its log files directly into the /var/spool/moab directory
### LOGFILEMAXSIZE

<table>
<thead>
<tr>
<th>Format:</th>
<th>INTEGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>50000000</td>
</tr>
<tr>
<td>Details:</td>
<td>Log files will be rolled when they reach 50 MB in size</td>
</tr>
</tbody>
</table>

### LOGFILEROLLDEPTH

<table>
<thead>
<tr>
<th>Format:</th>
<th>INTEGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>3</td>
</tr>
<tr>
<td>Details:</td>
<td>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.</td>
</tr>
</tbody>
</table>

#### Example:

<table>
<thead>
<tr>
<th>LOGFILEROLLDEPTH 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moab will maintain and roll the last 5 log files.</td>
</tr>
</tbody>
</table>

### LOGLEVEL

<table>
<thead>
<tr>
<th>Format:</th>
<th>INTEGER (0-9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>Details:</td>
<td>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.</td>
</tr>
</tbody>
</table>

#### Example:

<table>
<thead>
<tr>
<th>LOGLEVEL 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moab will write all Moab log messages with a threshold of 4 or lower to the moab.log file.</td>
</tr>
</tbody>
</table>

### LOGLEVELOVERRIDE

<table>
<thead>
<tr>
<th>Format:</th>
<th>BOOLEAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>FALSE</td>
</tr>
<tr>
<td>Details:</td>
<td>When this parameter is on, if someone runs a command with --loglevel=&lt;x&gt;, 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 &quot;gap&quot; 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).</td>
</tr>
</tbody>
</table>

#### Example:

| LOGLEVELOVERRIDE TRUE |

### LOGROLLACTION

<table>
<thead>
<tr>
<th>Format:</th>
<th>STRING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
</tbody>
</table>
### LOGROLLACTION

**Details:** Specifies a script to run when the logs roll. The script is run as a trigger and can be viewed using `mdiq -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

**Details:** If set to **NONE**, no mail will be sent. If set to Default:, the default mail program, `/usr/bin/sendmail`, will be used. **Note:** By default, Moab mail notification is disabled. To enable, MAILPROGRAM must be set to Default: or to the 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. See Notify Admins.

**Example:**

```
MAILPROGRAM /usr/local/bin/sendmail
```

### MAXGMETRIC

**Format:** `<INTEGER>`

**Default:** NONE

**Details:** Specifies how many generic metrics Moab should manage.

**Example:**

```
MAXGMETRIC 20
```

### MAXGRES

**Format:** `<INTEGER>`

**Default:** 1024

**Details:** Specifies how many generic resources Moab should manage.

**Example:**

```
MAXGRES 1024
```

### MAXJOB

**Format:** `<INTEGER>`

**Default:** 4096

**Details:** Specifies the maximum number of simultaneous jobs which can be evaluated by the scheduler. If additional jobs are submitted to the resource manager, Moab will ignore these jobs until previously submitted jobs complete. **Note:** Moab must be restarted for any changes to this parameter to take affect. The command `mdiq -S` will indicate if any job overflows have occurred.

**Example:**

```
MAXJOB 45000
```

### MAXRSVPERNODE

```
```
**MAXRSVPERNODE**

**Format:** `<INTEGER>`

**Default:** 24

**Details:** 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.

Moab must be restarted for any changes to this parameter to take effect. The command `mdia -S` indicates whether any node reservation overflows have occurred. See Considerations for Large Clusters.

**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:** ---

**Details:** 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:**

```
```
Moab will perform garbage collection once a day.

### MEMWEIGHT

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>0</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies the coefficient to be multiplied by a job's MEM (dedicated memory in MB) factor. See Resource Priority Overview.</td>
</tr>
</tbody>
</table>
| Example         | RESWEIGHT 10
                  MEMWEIGHT 1000 |

Each job's priority will be increased by 10 * 1000 * <request memory>.

### MINADMINSTIME

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>60 seconds</td>
</tr>
<tr>
<td>Details</td>
<td>Specifies the minimum time a job will be suspended if suspended by an administrator or by a scheduler policy.</td>
</tr>
<tr>
<td>Example</td>
<td>MINADMINSTIME 00:10:00</td>
</tr>
</tbody>
</table>

Each job suspended by administrators or policies will stay in the suspended state for at least 10 minutes.

### MISSINGDEPENDENCYACTION

<table>
<thead>
<tr>
<th>Format</th>
<th>CANCEL, HOLD, or RUN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>HOLD</td>
</tr>
<tr>
<td>Details</td>
<td>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.</td>
</tr>
<tr>
<td>Example</td>
<td>MISSINGDEPENDENCYACTION CANCEL</td>
</tr>
</tbody>
</table>

Any job that has a dependent job that cannot be found is cancelled.

### MSUBQUERYINTERVAL

<table>
<thead>
<tr>
<th>Format</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>5 seconds</td>
</tr>
<tr>
<td>Details</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:**

```bash
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`  

**Details:** Specifies how node resources will be shared by various tasks (See the Node Access Overview for more information).

**Example:**

```bash
NODEACCESSPOLICY SINGLEUSER
```

Moab will allow resources on a node to be used by more than one job provided that the job's are all owned by the same user.

### NODEALLOCATIONPOLICY

**Format:** One of the following: `FIRSTAVAILABLE`, `LASTAVAILABLE`, `MINRESOURCE`, `CPULOAD`, `LOCAL`, `CONTIGUOUS`, `MAXBALANCE`, `PRIORITY`, or `FASTEST`  

**Default:** `LASTAVAILABLE`  

**Details:** Specifies how Moab should allocate available resources to jobs. See Node Allocation Overview for more information.

**Example:**

```bash
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`  

**Details:** 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:**

```bash
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**

**Details:** 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 **NODEAVAILABILITYPOLICY**, 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)

**Details:** 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:** **---**

**Details:** 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
```

---
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, LOGLEVEL, MAXJOB, MAXJOBPERUSER, MAXLOAD, MAXPE, NODETYPE, OSLIST, PARTITION, POOL, POWERPOLICY, PRIORITY, PRIORITYF, PROCSPEED, RACK, RADISK, SLOT, SPEED, or TRIGGER

**Default:** ---

**Details:** 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 two 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)

**Details:** 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:** ---

**Details:** 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 ten 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)

**Details:** 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:**

```plaintext
| 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

**Details:** 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.

**Example:**

```plaintext
| NODEFAILURERESERVETIME | 10:00 |
```

Moab will reserve failed nodes for 10:00

### NODEIDFORMAT

**Format:** `<STRING>`

**Default:** `$N`

**Details:** 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:**

```plaintext
| NODEIDFORMAT | *$R*$S |
```

Moab will extract rack and slot information from the cluster node ids (ie, tg-13s08).

### NODELOADPOLICY

**Format:** One of the following: `ADJUSTSTATE` or `ADJUSTPROCS`

**Default:** `ADJUSTSTATE`
**Details:** Specifies whether a node's load affects its state or its available processors. The `ADJUSTSTATE` policy specifies that Moab should mark the node busy when the node's maximum load is reached. The `ADJUSTPROCS` policy causes the node's available procs to be equivalent to `MIN(ConfiguredProcs - DedicatedProcs, MaxLoad - CurrentLoad)` **Note:** `NODELOADPOLICY` only affects a node if `MAXLOAD` has been set.

**Example:**
```
NODELOADPOLICY ADJUSTSTATE
```

Moab will mark a node busy if its measured load exceeds its maximum cpu load limit.

---

**NODEMAXLOAD**

**Format:** `<DOUBLE>`

**Default:** 0.0

**Details:** Specifies that maximum cpu load on a idle or running node. If the node's load reaches or exceeds this value, Moab will mark the node busy (See Load Balancing Features)

**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:** ---

**Details:** 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)

**Details:** 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.

---

**NODEPURGETIME**
**NODEPURGETIME**

Format: `[[DD:]HH:]MM:]SS`

Default: `---` (never purge node info)

Details: The amount of time Moab will keep a node record which is no longer reported by the resource manager. Useful when using a resource manager which drops information about a node due to internal failures.

Example: `NODEPURGETIME 00:05:00`

Moab will maintain a node record for 5 minutes after the last update regarding that object received from the resource manager.

**NODESETATTRIBUTE**

Format: one of `ARCH`, `CLASS`, `FEATURE`, `MEMORY`, or `PROCSPEED`

Default: `---`

Details: Specifies the type of node attribute by which node set boundaries will be established. See Node Set Overview.

Example: `NODESETPOLICY ONEOF
NODESETATTRIBUTE PROCSPEED`

Moab will create node sets containing nodes with common processor speeds.

**NODESETDELAY**

Format: `[[DD:]HH:]MM:]SS`

Default: `0:00:00`

Details: Specifies the length of time Moab will delay a job if adequate idle resources are available but are not available within node set constraints. Setting `NODESETDELAY` to any non-zero value will force Moab to always use nodesets. A value of zero will cause Moab to use nodesets on a best effort basis. **Note**: This attribute is deprecated - use `NODESETISOPTIONAL` instead. See Node Set Overview for more information.

Example: `NODESETATTRIBUTE PROCSPEED
NODESETDELAY 0:00:00`

Moab will create node sets containing nodes with common processor speeds. Moab will allocate resources within the nodeset if possible. If no resources exist within a single nodeset, Moab will attempt to run the job using any available resources.

**NODESETISOPTIONAL**

Format: `<BOOLEAN>`

Default: `TRUE`

Details: 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.

<table>
<thead>
<tr>
<th><strong>NODESETLIST</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
</tbody>
</table>
| **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.

<table>
<thead>
<tr>
<th><strong>NODESETPLUS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th><strong>NODESETPOLICY</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
</tbody>
</table>
| **Example:**      | NODESETPOLICY ONEOF  
                   NODESETATTRIBUTE NETWORK |

Moab will create node sets containing nodes with common network interfaces.

<table>
<thead>
<tr>
<th><strong>NODESETPRIORITYTYPE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
</tbody>
</table>
### NODESETTOLERANCE

**Format:** `<FLOAT>`  
**Default:** `0.0` (Exact match only)  
**Details:** Specifies the tolerance for selection of mixed processor speed nodes. A tolerance of X allows a range of processors to be selected subject to the constraint .  
\[(\text{Speed.Max} - \text{Speed.Min}) / \text{Speed.Min} \leq X\]  
**Note:** Tolerances are only applicable when `NODESETATTRIBUTE` is set to `PROCSPEED`. See [Node Set Overview](#).

**Example:**

```plaintext
NODESETATTRIBUTE PROCSPEED  
NODESETTOLERANCE 0.5
```

Moab will only allocate nodes with up to a 50% procspeed difference.

### NODESYNCTIME

**Format:** `[[[DD:]HH:]MM:]SS`  
**Default:** `00:10:00`  
**Details:** 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:**

```plaintext
NODESYNCTIME 1:00:00
```

### NODETOJOBATTRMAP

**Format:** Comma delimited list of node features  
**Default:** `---`  
**Details:** 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:**

```plaintext
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`
| **Details** | 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. |
| **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. |
| | 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. |

**NODEWEIGHT**

**Format**: `<INTEGER>`

**Default**: 0

**Details**: 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**

**NOJOBHOLDNORESOURCES**

**Format**: `<BOOLEAN>`

**Default**: FALSE

**Details**: 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**: ---

**Details**: Specifies the name of the program to handle all notification call-outs.

**Example**: **NOTIFICATIONPROGRAM tools/notifyme.pl**

**NOWAITPREEMPTION**

**Format**: `<BOOLEAN>`

**Default**: ---

**Details**: 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
<table>
<thead>
<tr>
<th><strong>OPTIMIZEDCHECKPOINTING</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td>&lt;BOOLEAN&gt;</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td>FALSE</td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
<tr>
<td>If TRUE, Moab checkpoints VPCs on creation, modification and shutdown only, instead of in its normal checkpointing interval. <strong>OPTIMIZEDCHECKPOINTING</strong> only works when using a database.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>OPTIMIZEDCHECKPOINTING TRUE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>OSCREDLOOKUP</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td>NEVER</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
<tr>
<td>Disables all Moab OS credential lookups, including UID, GID, user to group mappings, and any other OS specific information.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>OSCREDLOOKUP NEVER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>PARALLOCATIONPOLICY</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td>One of BestFit, BestFitP, FirstStart, FirstCompletion, LoadBalance, LoadBalanceP, or RoundRobin</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td>FirstStart</td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
<tr>
<td>Specifies the approach to use to allocate resources when more than one eligible partition can be found. See Grid Scheduling Policies for more information.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>PARALLOCATIONPOLICY LOADBALANCE</td>
</tr>
</tbody>
</table>

New jobs will be started on the most lightly allocated partition.

<table>
<thead>
<tr>
<th><strong>PARCFG</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td>NODEPO WEROFFDURATION, NODEPOWERON DURATION, or one or more key-value pairs as described in the Partition Overview.</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td><strong>Details:</strong></td>
</tr>
<tr>
<td>Specifies the attributes, policies, and constraints for the given partition.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>PARCFG[oldcluster] MAX.WCLIMIT=12:00:00</td>
</tr>
</tbody>
</table>

Moab will not allow jobs to run on the oldcluster partition which has a wallclock limit in excess of 12 hours.
<table>
<thead>
<tr>
<th>PBSACCOUNTINGDIR</th>
<th>PBSACCOUNTINGDIR /var/spool/torque/sched_logs/</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
<td>&lt;PATH&gt;</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Details:</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>Job events will be written to the specified directory (can be consumed by PBS's tracejob command).</td>
</tr>
</tbody>
</table>

| PEWEIGHT | RESWEIGHT 10  
<table>
<thead>
<tr>
<th></th>
<th>PEWEIGHT 100</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
<td>&lt;INTEGER&gt;</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Details:</strong></td>
<td>Specifies the coefficient to be multiplied by a job's PE (processor equivalent) priority factor.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>Each job's priority will be increased by 10 * 100 * its PE factor.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PREEMPTPOLICY</th>
<th>PREEMPTPOLICY CHECKPOINT</th>
</tr>
</thead>
</table>
| **Format:**  | one of the following:  
|              | CANCEL, REQUEUE, SUSPEND, or CHECKPOINT |
| **Default:** | REQUEUE |
| **Details:** | 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. |
| **Example:** | Jobs that are to be preempted will be checkpointed and restarted at a later time. |

<table>
<thead>
<tr>
<th>PREEMPTPRIJOBSELECTWEIGHT</th>
<th>PREEMPTPRIJOBSELECTWEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong></td>
<td>&lt;DOUBLE&gt;</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>256.0</td>
</tr>
<tr>
<td><strong>Details:</strong></td>
<td>Adjusts the impact of job run priority versus job size in selecting which jobs to preempt. Setting the value higher increases the impact of priority; setting it lower decreases the impact.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>If set to 0, the cost of the job is the job's size.</td>
</tr>
</tbody>
</table>
If set to -1, the cost of the job is the wasted resource percentage.

<table>
<thead>
<tr>
<th>Example:</th>
<th>PREEMPTPRIOJOBSELECTWEIGHT 220.5</th>
</tr>
</thead>
</table>

**PREEMPTRTIMEWEIGHT**

**Format:** <DOUBLE>

**Default:** 0

**Details:** If set to 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.

<table>
<thead>
<tr>
<th>Example:</th>
<th>PREEMPTRTWEIGHT 1</th>
</tr>
</thead>
</table>

**PREEMPTSEARCHDEPTH**

**Format:** <INTEGER>

**Default:** unlimited

**Details:** Specifies how many preemptible jobs will be evaluated as potential targets for serial job preemptors. See Preemption Overview for more information.

<table>
<thead>
<tr>
<th>Example:</th>
<th>PREEMPTSEARCHDEPTH 8</th>
</tr>
</thead>
</table>

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:** ---

**Details:** 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.

<table>
<thead>
<tr>
<th>Example:</th>
<th>WALLTIMEWEIGHT -2500</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PRIORITYTARGETDURATION 1:00:00</td>
</tr>
</tbody>
</table>

**PRIORITYTARGETPROCCOUNT**

**Format:** <INTEGER>{+|-|%}

**Default:** ---

**Details:** 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.

<table>
<thead>
<tr>
<th>Example:</th>
<th>PROCWEIGHT -1000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PRIORITYTARGETPROCCOUNT 64</td>
</tr>
</tbody>
</table>
**PROCWEIGHT**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>Details:</td>
<td>Specifies the coefficient to be multiplied by a job's requested processor count priority factor.</td>
</tr>
<tr>
<td>Example:</td>
<td><strong>PROCWEIGHT</strong> 2500</td>
</tr>
</tbody>
</table>

**PROFILECOUNT**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>600</td>
</tr>
<tr>
<td>Details:</td>
<td>Specifies the number of statistical profiles to maintain.</td>
</tr>
<tr>
<td>Example:</td>
<td><strong>PROFILECOUNT</strong> 300</td>
</tr>
</tbody>
</table>

**PROFILEDURATION**

<table>
<thead>
<tr>
<th>Format:</th>
<th>[DD:]HH:]MM:]SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>00:00:00</td>
</tr>
<tr>
<td>Details:</td>
<td>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.</td>
</tr>
<tr>
<td>Example:</td>
<td><strong>PROFILEDURATION</strong> 24:00:00</td>
</tr>
</tbody>
</table>

**PURGETIME**

<table>
<thead>
<tr>
<th>Format:</th>
<th>[DD:]HH:]MM:]SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>Details:</td>
<td>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. <strong>Note:</strong> This parameter is superseded by <strong>JOBPURGETIME</strong> and <strong>NODEPURGETIME</strong>.</td>
</tr>
<tr>
<td>Example:</td>
<td><strong>PURGETIME</strong> 00:05:00</td>
</tr>
</tbody>
</table>

Moab will maintain a job or node record for 5 minutes after the last update regarding that object received from the resource manager.

**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, MEMBERULIST, QTWEIGHT, QTARGET, XFWEIGHT, XFTARGET, PREEMPTMINTIME, PREEMPTMAXTIME, PREEMPTQTTHRESHOLD, PREEMPTXFTHRESHOLD, RMAXPROC, RMAXDURATION |

---

**Example:**

- **PROCWEIGHT** 2500
- **PROFILECOUNT** 300
- **PROFILEDURATION** 24:00:00
- **PURGETIME** 00:05:00
RSVQTHRESHOLD, RSVXFTHRESHOLD, ACLBLTHRESHOLD, ACLQTHRESHOLD, ACLXFTHRESHOLD, PLIST, PDEF, QFLAGS, TRIGGER, or a usage limit.

**Default:** ---

**Details:** 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.

---

## QOSOPTIONAL

**Format:** `<BOOLEAN>`

**Default:** FALSE

**Details:** 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:**

```
QOSOPTIONAL 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**

**Details:** 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. Any combination of attributes may be specified. (see **JOBREJECTPOLICY**).

**Example:**

```
QOSREJECTPOLICY MAIL,CANCEL
```

---

## QOSWEIGHT

**Format:** `<INTEGER>`

**Default:** 1

**Details:** 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)

**Details:** 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

**Details:** 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.

### RECORDEVENTLIST

**Format:** One or more comma (',') or plus ('+') separated events of `GEVENT`, `ALLSCHEDCOMMAND`, `JOBCANCEL`, `JOBCHECKPOINT`, `JOBEND`, `JOBFAILURE`, `JOBMIGRATE`, `JOBMODIFY`, `JOBPREEMPT`, `JOBREJECT`, `JOBRESUME`, `JOBSTART`, `JOBSUBMIT`, `NODEDOWN`, `NODEUP`, `QOSVIOLATION`, `RMDOWN`, `RMPOLLEND`, `RMPOLLSTART`, `RMUP`, `RSVCANCEL`, `RSVCREATE`, `RSVEND`, `RSVMODIFY`, `RSVSTART`, `SCHEDCOMMAND`, `SCHEDCYCLEEND`, `SCHEDCYCLESTART`, `SCHEDPAUSE`, `SCHEDSTART`, `SCHEDSTOP` or `ALL`

**Default:** `JOBSTART`, `JOBCANCEL`, `JOBEND`, `JOBFAILURE`, `SCHEDPAUSE`, `SCHEDSTART`, `SCHEDSTOP`, `TRIGEND`, `TRIGFAILURE`, `TRIGSTART`

**Details:** 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.

### REJECTINFEASIBLEJOBS

**Format:** `<BOOLEAN>`

**Default:** FALSE

**Details:** 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.
REJECTINEASIBLEJOBS TRUE
JOBREJECTPOLICY MAIL,CANCEL

Any job with zero feasible nodes for execution will be rejected.

REJECTINEASIBLEJOBS
Format: <BOOLEAN>
Default: TRUE
Details: 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
REJECTINEASIBLEJOBS TRUE

Any job with a priority less than zero will be rejected.

REMAPCLASS
Format: <ClassID>
Default: ---
Details: 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.

Example: 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: ---
Details: 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: REMAPCLASS batch
REMAPCLASSLIST short,medium,long

Class batch will be re-mapped to one of the listed classes.
### REMOTEFAILTRANSIENT

**Format:** <BOOLEAN>  
**Default:** FALSE  
**Details:** 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.  

**Example:**

```
REMOTEFAILTRANSIENT    TRUE
```

### REMOVETRIGOUTPUTFILES

**Format:** <BOOLEAN>  
**Default:** FALSE  
**Details:** 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)  
**Details:** 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  
**Details:** 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**  
**Details:** 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 two currently highest priority jobs.

### RESERVATIONQOSLIST[X]

**Format:** One or more QOS values or [ALL]  
**Default:** [ALL]  
**Details:** 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  
**Details:** 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. See Reservation Policies and JOBRETRYTIME.  
**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.

### RESOURCENLIMMULTIPLIER[<PARID>]

**Format:** <RESOURCE>:[MULTIPLIER>[,[...]]  
**Default:** 1.0  
**Details:** Where <RESOURCE> is one of the following: **NODE, PROC, JOBPROC, MEM, JOBMEM, SWAP, DISK, or WALLTIME**
### RESOURCELIMITMULTIPLIER

**Details:** 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 Resource Usage Limits.

**Example:**

```
RESOURCENAME LIMITMULTIPLIER  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>:[<POLICY>,]<HPOLICY>[:<ACTION>,]<ACTION>[:<SVIOLATIONTIME>,<HVIOLATIONTIME>]...`

Where `RESOURCE` is one of PROC, JOBPROC, JOBMEM, DISK, SWAP, MEM 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.

**Details:** Specifies how the scheduler should handle jobs which utilize more resources than they request. See Resource Usage Limits.

**Example:**

```
RESOURCENAME LIMITPOLICY  MEM:ALWAYS,BLOCKEDWORKLOADONLY:REQUEUE,CANCEL
```

Moab will cancel all jobs which exceed their requested memory limits.

### RESOURCENAMELIST

**Format:** `<node>[[<node>]]...

**Default:** ---

**Details:** Specifies which resources will have their information gathered from the NATIVE RM interface.

**Example:**

```
RESOURCENAMELIST  node01,node05,node06
```

Moab will query the native rm for information about nodes node01, node05, and node06.

### RESTARTINTERVAL

**Format:** `[[[DD:]HH:]MM:]SS`

**Default:** ---

**Details:** 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.
<table>
<thead>
<tr>
<th><strong>RESOURCEQUERYDEPTH</strong></th>
<th><strong>Format:</strong></th>
<th>&lt;INTEGER&gt;</th>
<th><strong>Default:</strong></th>
<th>3</th>
<th><strong>Details:</strong></th>
<th>Maximum number of options which will be returned in response to an <code>mshow -a</code> resource query.</th>
<th><strong>Example:</strong></th>
<th>RESOURCEQUERYDEPTH 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The 'mshow -a' command will return at most one valid collection of resources.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **RESWEIGHT**          | **Format:** | <INTEGER> | **Default:** | 1 | **Details:** | 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 |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The job priority resource factor will be calculated as MIN(2000,5 * (10 * JobMemory + 100 * JobProc)).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>RMCFG</strong></th>
<th><strong>Format:</strong></th>
<th>One or more <strong>key-value</strong> pairs as described in the Resource Manager Configuration Overview</th>
<th><strong>Default:</strong></th>
<th>N/A</th>
<th><strong>Details:</strong></th>
<th>Specifies the interface and policy configuration for the scheduler-resource manager interface. Described in detail in the Resource Manager Configuration Overview.</th>
<th><strong>Example:</strong></th>
<th>RMCFG[TORQUE3] TYPE=PBS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The PBS server will be used for resource management.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>RMMSGIGNORE</strong></th>
<th><strong>Format:</strong></th>
<th>&lt;BOOLEAN&gt;</th>
<th><strong>Default:</strong></th>
<th>FALSE</th>
<th><strong>Details:</strong></th>
<th>Specifies whether or not Moab should adjust node state based on generic resource manager failure messages. See RM Health Check for more info.</th>
<th><strong>Example:</strong></th>
<th>RMMSGIGNORE TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Moab will load and report resource manager failure messages but will not adjust node state as a result of them.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## RMPOLLINTERVAL

**Format:** `<MINPOLLTIME>,<MAXPOLLTIME>` where poll time is specified as `[[DD:]HH:]MM:]SS`

**Default:** 00:00:30

**Details:** Specifies interval between RM polls. The poll interval will be no less than `MINPOLLTIME` and no more than `MAXPOLLTIME`. A single interval interval (interpreted by Moab as the maximum interval) can be used, or you can specify a minimum and maximum interval. If using a single interval, Moab sometimes has iterations of less than the specified interval.

**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

**Details:** 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.

## RSVCTLPOLICY

**Format:** `ADMINONLY` or `ANY`

**Default:** `ADMINONLY`

**Details:** Specifies who can create admin reservations.

**Example:**

```
RSVCTLPOLICY ANY
```

Any valid user can create an arbitrary admin reservation.

## RSVLIMITPOLICY

**Format:** `HARD` or `SOFT`

**Default:** ---

**Details:** Specifies what limits should be enforced when creating reservations.

**Example:**

```
RSVLIMITPOLICY HARD
```

Moab will limit reservation creation based on the `HARD` limits configured.
RSVPROFILE

Format: One or more of the following <ATTR>=<VALUE> pairs
ACCESS, ACCOUNTLIST, CHARGEACCOUNT, CLASSLIST, CLUSTERLIST, DAYS, DEPTH,
ENDTIME, FLAGS, GROUPLIST, HOSTLIST, JOBATTRLIST, MAXTIME, NODEFEATURES,
OWNER, PARTITION, PERIOD, PRIORITY, QOSLIST, RESOURCES, RSVACCESSLIST,
STARTTIME, TASKCOUNT, TIMELIMIT, TPN, TRIGGER, or USERLIST.

Note: Lists of more than one ACL value cannot be whitespace delimited. Such lists must be
delimited with either the comma, pipe, or colon characters.

Default: ---

Details: 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.

RSVREALLOCPOLICY

Format: One of FAILURE, NEVER, OPTIMAL, PRESTARTFAILURE, PRESTARTOPTIMAL, or REMAP

Default: NEVER

Details: Specifies the policy Moab uses to dynamically reallocate nodes to an existing reservation. See
Reservation Resource Allocation Policy for more information.

Example:
RSVREALLOCPOLICY FAILURE

Should any node go down within a reservation Moab will attempt to allocate a replacement.
Moab will NOT allocate a busy or otherwise conflicting node. If Moab should allocate any node for
replacement the IGNRSV flag must be used when creating the reservation.

SCHEDCFG

Format: List of zero or more space delimited <ATTR>=<VALUE> pairs where <ATTR> is one of the
following: CHARGEMETRICPOLICY, CHARGERATEPOLICY, FBSERVER, FLAGS,
MAXRECORDEDJOBID, MINJOBID, HTTPSERVERPORT, MODE, RECOVERYACTION, SERVER,
or TRIGGER

Default: ---

Details: Specifies scheduler policy and interface configuration

Example:
SCHEDCFG[zylem3] SERVER=geronimo.scc.com:3422 MODE=NORMAL

Moab will execute in NORMAL mode on the host geronimo.scc.com.

SERVERHOST

Format: <HOSTNAME>

Default: ---
| **Details:** Deprecated. Hostname of machine on which moab will run. See SCHEDCFG for replacement parameter. |
|**Example:** SERVERHOST geronimo.scc.edu Moab will execute on the host geronimo.scc.edu. |

**SERVERMODE**

| **Format:** One of the following: INTERACTIVE, MONITOR, NORMAL, SIMULATION, or SLAVE |
| **Default:** NORMAL |
| **Details:** Deprecated. Specifies how Moab interacts with the outside world. See SCHEDCFG for replacement parameter. |
|**Example:** SERVERMODE SIMULATION |

**SERVERNAME**

| **Format:** <STRING> |
| **Default:** <SERVERHOST> |
| **Details:** 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 |
| **Details:** 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. |

**SERVICEWEIGHT**

| **Format:** <INTEGER> |
| **Default:** 1 |
| **Details:** 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 |
### Details:
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

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;BOOLEAN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Details: If TRUE, the scheduler will end simulations when the active queue and idle queue become empty.

### SIMCPUSCALINGPERCENT

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>100 (no scaling)</td>
</tr>
</tbody>
</table>

Details: Specifies whether to increase or decrease the runtime and wallclock limit of each job in the workload trace file.

### SIMDEFAULTJOBFLAGS

<table>
<thead>
<tr>
<th>Format:</th>
<th>One or more of the following: ADVRES, HOSTLIST, RESTARTABLE, PREEMPTEE, DEDICATED, PREEMPTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
</tbody>
</table>

Details: Cause Moab to force the specified job flags on all jobs supplied in the workload trace file.

### SIMEXITITERATION

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;INTEGER&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>0 (no exit iteration)</td>
</tr>
</tbody>
</table>

Details: Iteration on which a Moab simulation will create a simulation summary and exit.

Example: `SIMEXITITERATION 36000`
**SIMFLAGS**

**Format:** Comma delimited list of zero or more of the following:
- IGNHOSTLIST
- IGNCLASS
- IGNQOS
- IGNMODE
- IGNFEATURES
- IGNGRES

**Default:** ---

**Details:** Controls how Moab handles trace based simulation information (see Simulation Overview).

**Example:**

```
SIMFLAGS IGNHOSTLIST,IGNQOS
```

Moab will ignore hostlist and qos requests specified in the workload trace file.

**SIMIGNOREJOBFLAGS**

**Format:** One or more of the following:
- ADVRES
- HOSTLIST
- RESTARTABLE
- PREEMPTEE
- DEDICATED
- PREEMPTOR

**Default:** ---

**Details:** Moab will ignore specified job flags if supplied in the workload trace file (see Simulation Overview).

**Example:**

```
SIMIGNOREJOBFLAGS DEDICATED
```

Moab will ignore the DEDICATED job flag if specified in any job trace.

**SIMINITIALQUEUEDEPTH**

**Format:** `<INTEGER>`

**Default:** 16

**Details:** 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
- Replay

**Default:** CONSTANTJOBDEPTH

**Details:** 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 processeconds 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).
<table>
<thead>
<tr>
<th>Example:</th>
<th>SIMJOBSUBMISSIONPOLICY NORMAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moab will submit jobs with the relative time distribution specified in the workload trace file.</td>
<td></td>
</tr>
</tbody>
</table>

### SIMNODECONFIGURATION

**Format:** UNIFORM or NORMAL  
**Default:** NORMAL  
**Details:** Specifies whether or not Moab will filter nodes based on resource configuration while running a simulation.  
**Example:** SIMNODECONFIGURATION UNIFORM

### SIMNODECOUNT

**Format:** <INTEGER>  
**Default:** 0 (no limit)  
**Details:** Specifies the maximum number of nodes Moab will load from the simulation resource file.  
**Example:** SIMNODECOUNT 256

### SIMPURGEBLOCKEDJOBS

**Format:** <BOOLEAN>  
**Default:** TRUE  
**Details:** Specifies whether Moab should remove jobs which can never execute (see Simulation Overview).  
**Example:** SIMPURGEBLOCKEDJOBS FALSE

### SIMRESOURCETRACEFILE

**Format:** <STRING>  
**Default:** Traces/resource.trace  
**Details:** Specifies the file from which Moab will obtain node 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 Resource Trace Format.  
**Example:** SIMRESOURCETRACEFILE traces/nodes.1  
Moab will obtain resource traces when running in simulation mode from the <MOABHOMEDIR>/traces/nodes.1 file.

### SIMRMRANDOMDELAY

**Format:** <INTEGER>
| **Default:** | 0 |
| **Details:** | 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:** | --- |
| **Details:** | 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 |

**SIMSTOPITERATION**

| **Format:** | `<INTEGER>` |
| **Default:** | -1 (don't stop) |
| **Details:** | Specifies on which scheduling iteration a Moab simulation will stop and wait for a command to resume scheduling. See Simulation Overview for more information. |
| **Example:** | `SIMSTOPITERATION 10` Moab should pause after iteration 10 of simulated scheduling and wait for administrator commands. |

**SIMSTOPTIME**

| **Format:** | `[HH[:MM[:SS]][_MO[/DD[/YY]]]` |
| **Default:** | --- |
| **Details:** | 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. |

**SIMTIMEPOLICY**

<p>| <strong>Format:</strong> | REAL or NONE |
| <strong>Default:</strong> | --- |
| <strong>Details:</strong> | Determines simulation time management policy (see SIMTIMERATIO). |
| <strong>Example:</strong> | <code>SIMTIMEPOLICY REAL</code> Moab simulation time will advance in line with real time. |</p>
<table>
<thead>
<tr>
<th><strong>SIMTIMERATIO</strong></th>
<th>Format: &lt;INTEGER&gt;</th>
<th>Default: (no time ratio)</th>
<th>Details: Determines wall time speedup. Simulated Moab time will advance &lt;SIMTIMERATIO&gt; * faster than real wall time (see SIMTIMEPOLICY).</th>
<th>Example:</th>
<th>SIMTIMERATIO 10</th>
<th>SIMTIMEPOLICY REAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moab simulation time will advance 10 times faster than real world wall time. For example, in 1 hour, Moab will process 10 hours of simulated workload.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **SIMWORKLOADTRACEFILE** | Format: <STRING> | Default: Traces/workload.trace | Details: 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: --- | Details: Specifies the directory for temporary spool files created by Moab while submitting a job to the RM. | Example: | SPOOLDIR /tmp/moab/spool |
|------------------|------------------|----------------------|----------------------------------------------------------------------------|-------------------|-----------------|------------------|

| **SPVIOLATIONWEIGHT** | Format: <INTEGER> | Default: 0 | Details: Specifies the weight to be applied to a job which violates soft usage limit policies (see Service Priority Component Overview). | Example: | SPVIOLATIONWEIGHT 5000 |
|--------------------------|------------------|----------------------|----------------------------------------------------------------------------|-------------------|-----------------|------------------|

| **SRCFG[X]** | Format: One or more of the following <ATTR>=<VALUE> pairs | ACCESS, ACCOUNTLIST, CHARGEACCOUNT, CHARGEUSER, CLASSLIST, CLUSTERLIST, COMMENT, DAYS, DEPTH, DISABLE, ENDTIME, FLAGS, GROUPLIST, HOSTLIST, JOBBATTRLIST, MAXTIME, NODEFEATURES, OWNER, PARTITION, PERIOD, PRIORITY, | |
|------------------|------------------|----------------------|----------------------------------------------------------------------------|-------------------|-----------------|------------------|
**QOSLIST, 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

---

### STARTCOUNTCAP

**Format:** `<INTEGER>`

**Default:** 0

**Details:** 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

**Details:** 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

**Details:** Specifies the directory in which Moab statistics will be maintained.

**Example:**

STATDIR /var/adm/moab/stats

### STATPROCMAX

**Format:** `<INTEGER>`

**Default:** 1

**Details:**

---

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.

**SRCFG[fast]**

STARTTIME=9:00:00 ENDTIME=15:00:00

SRCFG[fast] HOSTLIST=node0[1-4]

SRCFG[fast] QOSLIST=high,low
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
STATPROCSTECOUNT 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

**Details:** 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
STATPROCSTECOUNT 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.

### STATPROCSTECOUNT

**Format:** `<INTEGER>`

**Default:** 5

**Details:** 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
```
### STATPROCESIZE

**Format:** `<INTEGER>`

**Default:** 4

**Details:**
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
STATPROCESIZE 4
```

### STATTIMEMAX

**Format:** `[[DD:]HH:]MM:]SS`

**Default:** 00:15:00

**Details:**
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
STATTIMESTEPSEIZE 4
```

### STATTIMEMIN

**Format:** `[[DD:]HH:]MM:]SS`

**Default:** 00:15:00

**Details:**
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:
```
<table>
<thead>
<tr>
<th>STATTIMEMIN</th>
<th>00:02:00</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATTIMESTEPCOUNT</td>
<td>4</td>
</tr>
<tr>
<td>STATTIMESTEPsize</td>
<td>4</td>
</tr>
</tbody>
</table>
```
Each matrix output will display data in columns for jobs requesting between 2 and 128 minutes.

### STATTIMESTEPcount

**Format:** `<INTEGER>`

**Default:** 6

**Details:** 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:
```
<table>
<thead>
<tr>
<th>STATTIMEMIN</th>
<th>00:02:00</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATTIMESTEPcount</td>
<td>4</td>
</tr>
<tr>
<td>STATTIMESTEPsize</td>
<td>4</td>
</tr>
</tbody>
</table>
```
Each matrix output will display data in columns for jobs requesting between 2 and 128 minutes.

### STATTIMESTEPsize

**Format:** `<INTEGER>`

**Default:** 4

**Details:** 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:
```
<table>
<thead>
<tr>
<th>STATTIMEMIN</th>
<th>00:02:00</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATTIMESTEPcount</td>
<td>4</td>
</tr>
<tr>
<td>STATTIMESTEPsize</td>
<td>4</td>
</tr>
</tbody>
</table>
```
Each matrix output will display data in columns for jobs requesting between 2 and 128 minutes.

### STRICTPROTOCOLCHECK

**Format:** `<BOOLEAN>`

**Default:** FALSE

**Details:** 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 |
```

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.
Moab will reject any XML communication that does not strictly conform to the expected protocol definition.

### SUBMITFILTER

**Format:** `<STRING>`

**Default:** --

**Details:** Specifies the directory of a given submit filter script.

**Example:** `SUBMITFILTER /home/submitfilter/filter.pl`

### SUBMITHOSTS

**Format:** space delimited list of host names

**Default:** --

**Details:** 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:** --

**Details:** 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`
- `PDEF`
- `FLAGS`

**Default:** --

**Details:** 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
**Details:** Specifies the priority weight assigned to the virtual memory request of a job.
**Example:**
```
SWAPWEIGHT 10
```

### SYSTEMMAXPROCPERJOB
**Format:** `<INTEGER>`
**Default:** -1 (NO LIMIT)
**Details:** 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)
**Details:** 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 proc 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)
**Details:** 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 one day of walltime.

### TARGETQUEUETIMEWEIGHT
**Format:** `<INTEGER>`
**Default:** 0
**Details:** Specifies the weight assigned to the time remaining until the queuetime is reached.
**Example:**
```
TARGETQUEUETIMEWEIGHT 10
```

### TARGETWEIGHT
### TARGETWEIGHT

**Format:** `<INTEGER>`  
**Default:** `1`  
**Details:** 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`  
**Details:** Specifies the weight assigned to the distance to the target expansion factor.  
**Example:**  
```
TARGETXFACTORWEIGHT 10
```

### TASKDISTRIBUTIONPOLICY

**Format:** One of `DEFAULT`, `PACK`, `RR` (round-robin), or `LOCAL`  
**Default:** `---`  
**Details:** Specifies how job tasks should be mapped to allocated resources. See Task Distribution Overview for more information.  
**Example:**  
```
TASKDISTRIBUTIONPOLICY DEFAULT  
Moab should use standard task distribution algorithms.
```

### TOOLSDIR

**Format:** `<STRING>`  
**Default:** `Tools`  
**Details:** 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:** `---`  
**Details:** Specifies the functions to be trapped.  
**Example:**  
```
TRAPFUNCTION UpdateNodeUtilization|GetNodeSResTime
```

### TRAPJOB

**Format:** `<STRING>`  
**Default:** `---`
### TRAPJOB

**Details:** Specifies the jobs to be trapped.

**Example:** TRAPJOB pros23.0023.0

### TRAPNODE

**Format:** `<STRING>`

**Default:** ---

**Details:** Specifies the nodes to be trapped.

**Example:** TRAPNODE node001|node004|node005

### TRAPRES

**Format:** `<STRING>`

**Default:** ---

**Details:** Specifies the reservations to be trapped.

**Example:** TRAPRES interactive.0.1

### TRIGCHECKTIME

**Format:** `<INTEGER>` (milliseconds)

**Default:** 2000

**Details:** 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

**Details:** 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
Details: 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)
Details: 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.

UNSORTEDDEPENDENCIES
Format: Comma delimited string
Default: ---
Details: Specifies dependencies that are not supported and should not be accepted by job submissions. A maximum of 30 dependencies is supported.
Example: moab.cfg:
UNSORTEDDEPENDENCIES before,beforeok,beforenotok,on
Example:
> msub -l depend=before:105 cmd.sh
ERROR: cannot submit job - error in extension string

UPROCSWEIGHT
Format: <INTEGER>
Default: 0
Details: 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
**Details:** Specifies the weight assigned to per job processor second consumption.
**Example:** `USAGECONSUMEDWEIGHT 10`

### USAGEEXECUTIONTIMEWEIGHT

**Format:** `<INTEGER>`
**Default:** 0
**Details:** 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
**Details:** Specifies the weight assigned to total requested resources consumed.
**Example:** `USAGEPERCENTWEIGHT 5`

### USAGEREMAININGWEIGHT

**Format:** `<INTEGER>`
**Default:** 0
**Details:** Specifies the weight assigned to remaining usage.
**Example:** `USAGEREMAININGWEIGHT 10`

### USAGEWEIGHT

**Format:** `<INTEGER>`
**Default:** 1
**Details:** Specifies the weight assigned to the percent and total job usage subfactors.
**Example:** `USAGEWEIGHT 100`

### USEANYPARTITIONPRIO

**Format:** `<BOOLEAN>`
**Default:** FALSE
**Details:** 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.

**Example:**
```
USEANYPARTITIONPRIO TRUE
```

### USECPRSVNODELIST

**Format:** `<BOOLEAN>`

**Default:** `TRUE`

**Details:** 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:** `-`

**Details:** 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
```

### USEMACHINESPEED

**Format:** `<BOOLEAN>`

**Default:** `TRUE`

**Details:** Specifies whether or not job wallclock limits should be scaled by the machine speed of the node(s) they are running on.

**Example:**
```
USEMACHINESPEED TRUE
```

Job `<X>` specifying a wallclock limit of 1:00:00 would be given only 40 minutes to run if started on a node with a machine speed of 1.5.

### USEMOABTIME

**Format:** `<BOOLEAN>`

**Default:** `FALSE`

**Details:** 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
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:** \(<{\text{BOOLEAN}}\)\>

**Default:** **FALSE**

**Details:** Specifies whether to use the Moab job ID, or the resource manager's job ID.

**Example:**

```
USEMOABJOBID_TRUE
```

### USENEWXMLVARIABLES

**Format:** \(<{\text{BOOLEAN}}\)\>

**Default:** **FALSE**

**Details:** Reports variables as child elements in node XML.

**Example:**

```
USENEWXMLVARIABLES_TRUE
```

### USERCFG[<USERID>]

**Format:** List of zero or more space delimited \(<\text{ATTR}>={\text{VALUE}}\) pairs where \(<\text{ATTR}>\) is one of the following:

- General Credential Flags, \(\text{CDEF}\), \(\text{DEFAULT.TPN}\), \(\text{DEFAULT.WCLIMIT}\), \(\text{EMAILADDRESS}\), \(\text{ENABLEPROFILING}\), \(\text{FSCAP}\), \(\text{FSTARGET}\), \(\text{JOBFLAGS}\), \(\text{MAX.WCLIMIT}\), \(\text{QLIST}\), \(\text{QDEF}\), \(\text{EMAIL}\), \(\text{OVERRUN}\), \(\text{PDEF}\), \(\text{PLIST}\), \(\text{PREF}\), \(\text{PRIORITY}\), \(\text{TRIGGER}\), or a usage limit.

**VIEWPOINTPROXY** is an option available to those using Viewpoint as the front-end user interface. More information on **VIEWPOINTPROXY** is available in the Viewpoint documentation.

**Default:** ---

**Details:** 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:** \(<{\text{INTEGER}}\)\>

**Default:** -

---
### USERPRIOWEIGHT
**Format:** <INTEGER>
**Default:** 0
**Details:** 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
```

### USERWEIGHT
**Format:** <INTEGER>
**Default:** 1
**Details:** 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.

**Example:**
```
USERPRIOWEIGHT 10
```

### USESYSLOG
**Format:** <BOOLEAN>[<FACILITY>]
**Default:** FALSE:daemon
**Details:** 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
**Details:** 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 JOBPRIOACCURRALPOLICY parameter.

**Example:**
```
USESYSYSTEMQUEUETIME 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

**Details:** Enables searching of the user buffer using the user hash key instead of doing sequential searches of the user buffer.

**Example:** USEUSERHASH TRUE

**VCPROFILE**

**Format:** Comma delimited list of one or more of the following:
- ACL, DEFAULTRSVPROFILE, DESCRIPTION, NODESETLIST, OPRSVPROFILE, QUERYDATA, REQENDPAD, REQSETATTR or REQSTARTPAD

**Default:** ---

**Details:** Defines virtual cluster attributes (see VPC Overview).

**Example:** VCPROFILE[pkgA] REQENDPAD=0:10:00

All pkgA VPC’s will allocate an additional 10 minutes of time at the end to account for provisioning overhead.

**VMCALCULATELOADBYVMSUM**

**Format:** <BOOLEAN>

**Default:** False

**Details:** 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

**VMCREATETHROTTLE**

**Format:** <INTEGER>

**Default:** ---

**Details:** Sets the maximum allowable 'VM create' jobs at any given time.

**Example:** VMCREATETHROTTLE 25

Only 25 VM creation jobs are allowed in the system at any given time.

**VMMIGRATETHROTTLE**

**Format:** <INTEGER>

**Default:** ---
### VMMIGRATIONPOLICY

**Format:** `<STRING>`, values include RSV, GREEN, and OVERCOMMIT

**Default:** NONE

**Details:**
- **RSV** - If RSV flag is set, vacates VMs on a node for a reservation to run.
- **GREEN** - If GREEN flag is set, Moab consolidates VMs to allow nodes to go idle.
- **OVERCOMMIT** - The OVERCOMMIT flag must be set for the `VMOCTHRESHOLD` parameter to function.

**Example:**

```
VMMIGRATIONPOLICY GREEN,OVERCOMMIT
```

### VMMINOPDELAY

**Format:** `[HH[:MM[:SS]]`

**Default:** --

**Details:** 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>`

**Default:**

**Details:** Percentage threshold at which Moab begins to migrate virtual machines to other nodes. `VMMIGRATIONPOLICY` must be set to OVERCOMMIT for this to occur.

**Example:**

```
VMOCTHRESHOLD PROC:0.7,MEM:0.9
```

When a node surpasses .7 (70%) load of CPU or .9 (90%) of memory, Moab begins to migrate virtual machines to other nodes.

### VMProvisionStatusReadyValue

**Format:** `<INTEGER>`

**Default:** ---

**Details:** 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.

**Examples:**

```
VMProvisionStatusReadyValue 2
VMProvisionStatusReadyValue 1-4,6,16
```
### VMSARESTATIC

**Format:** `<BOOLEAN>`  
**Default:** `FALSE`  
**Details:** 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
```

### VMSTORAGEMOUNTDIR

**Format:** `<PATH>`  
**Default:** `---`  
**Details:** The specified path is used as the default location for storage mounts in all newly created VMs (created via the `mvmct` command). This parameter defines the default storage mount directory if one is not specified.  
**Example:**  
```
VMSTORAGEMOUNTDIR /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:** `<BOOLEAN>`  
**Default:** `FALSE`  
**Details:** When set to TRUE, VMTracking jobs are used to represent VMs in the queue.  
**Example:**  
```
VMTRACKING TRUE
```

### VPCFLAGS

**Format:** Comma delimited list of one or more of the following: `CREDS`, `JOBS`, `NODES`, or `STATS`  
**Default:** `---`  
**Details:** Virtual private cluster flags that specify the attributes to restrict to the users in a virtual private cluster.  
**Example:**  
```
VPCFLAGS=CREDS,JOBS,NODES
```
Moab will restrict the viewing of credentials, jobs and nodes to only within a virtual private cluster.

### WALLTIMECAP

**Format:** `<DOUBLE>`  
**Default:** `0 (NO CAP)`  
**Details:** 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

**Details:** 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)

**Details:** 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

**Details:** 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**

**Details:** 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 Resource Usage Limit Overview.

**Example:**

```
WCVIOLATIONACTION PREEMPT
```
Moab will requeue jobs which exceed their wallclock limit.

### WIKIEVENTS

**Format:** <BOOLEAN>  
**Default:** TRUE  
**Details:** When set to true, Moab events are set to native wiki format (ATTR=VALUE pairs) to facilitate easier readability.  
**Example:**
```
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=>= RDISKCMPCMP=>= RFEATURES=[NONE] SYSTEMQUEUE=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)  
**Details:** 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  
**Details:** 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)  
**Details:** Specifies the minimum job wallclock limit that will be considered in job expansion factor priority calculations.
<table>
<thead>
<tr>
<th>Example:</th>
<th>XFINWCLIMIT 0:01:00</th>
</tr>
</thead>
</table>

Jobs requesting less than one minute of wallclock time will be treated as if their wallclock limit was set to one minute when determining expansion factor for priority calculations.
### Appendix G: Commands Overview

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkjob</td>
<td>provide detailed status report for specified job</td>
</tr>
<tr>
<td>checknode</td>
<td>provide detailed status report for specified node</td>
</tr>
<tr>
<td>mcredctl</td>
<td>controls various aspects about the credential objects within Moab</td>
</tr>
<tr>
<td>mdiag</td>
<td>provide diagnostic reports for resources, workload, and scheduling</td>
</tr>
<tr>
<td>mjobctl</td>
<td>control and modify job</td>
</tr>
<tr>
<td>mnodectl</td>
<td>control and modify nodes</td>
</tr>
<tr>
<td>moab</td>
<td>control the Moab daemon</td>
</tr>
<tr>
<td>mrmctl</td>
<td>query and control resource managers</td>
</tr>
<tr>
<td>mrvctl</td>
<td>create, control and modify reservations</td>
</tr>
<tr>
<td>mschedctl</td>
<td>modify scheduler state and behavior</td>
</tr>
<tr>
<td>mshow</td>
<td>displays various diagnostic messages about the system and job queues</td>
</tr>
<tr>
<td>mshow -a</td>
<td>query and show available system resources</td>
</tr>
<tr>
<td>msub</td>
<td>scheduler job submission</td>
</tr>
<tr>
<td>mvmctl</td>
<td>create, control and modify VMs</td>
</tr>
<tr>
<td>resetstats</td>
<td>reset scheduler statistics</td>
</tr>
<tr>
<td>showbf</td>
<td>show current resource availability</td>
</tr>
<tr>
<td>showq</td>
<td>show queued jobs</td>
</tr>
<tr>
<td>showres</td>
<td>show existing reservations</td>
</tr>
<tr>
<td>showstart</td>
<td>show estimates of when job can/will start</td>
</tr>
<tr>
<td>showstate</td>
<td>show current state of resources</td>
</tr>
<tr>
<td>showstats</td>
<td>show usage statistics</td>
</tr>
<tr>
<td>showstats -f</td>
<td>show various tables of scheduling/system performance</td>
</tr>
</tbody>
</table>

### Commands Providing Maui Compatibility

⚠️ The following commands are deprecated. Click the link for respective deprecated commands to see the updated replacement command for each.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canceljob</td>
<td>cancel job</td>
</tr>
<tr>
<td>changeparam</td>
<td>change in memory parameter settings</td>
</tr>
<tr>
<td>diagnose</td>
<td>provide diagnostic report for various aspects of resources, workload, and scheduling</td>
</tr>
<tr>
<td>releasehold</td>
<td>release job defers and holds</td>
</tr>
<tr>
<td>releaseres</td>
<td>release reservations</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>runjob</td>
<td>force a job to run immediately</td>
</tr>
<tr>
<td>sethold</td>
<td>set job holds</td>
</tr>
<tr>
<td>setqos</td>
<td>modify job QOS settings</td>
</tr>
<tr>
<td>setres</td>
<td>set an admin/user reservation</td>
</tr>
<tr>
<td>setspri</td>
<td>adjust job/system priority of job</td>
</tr>
<tr>
<td>showconfig</td>
<td>show current scheduler configuration</td>
</tr>
</tbody>
</table>
**checkjob**

*(Check Job)*

**Synopsis**

```bash
```

**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.

**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**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Format</th>
<th>Default</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A</td>
<td>Attribute-Value pair</td>
<td>-</td>
<td>Provides output in the form of parsable Attribute-Value pairs.</td>
<td><code>&gt; checkjob -A 6235</code>&lt;br&gt;Moab will display job information in the following format: <code>&lt;ATTRIBUTE&gt;=&lt;VALUE&gt;;</code>.</td>
</tr>
<tr>
<td>--flags</td>
<td>-</td>
<td>-</td>
<td>Evaluates future eligibility of job (ignore current resource state and usage limitations).</td>
<td><code>&gt; checkjob --flags=future -v 6235</code>&lt;br&gt;Display reasons why idle job is blocked ignoring node state and current node utilization constraints.</td>
</tr>
<tr>
<td>-l</td>
<td>Policy level</td>
<td>-</td>
<td>Reports job start eligibility subject to specified throttling policy level.</td>
<td><code>&gt; checkjob -l SOFT 6235</code>&lt;br&gt;<code>&gt; checkjob -l HARD 6235</code></td>
</tr>
</tbody>
</table>
**-n (NodeID)**

**Format:** `<NODEID>`

**Default:** ---

**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>`

**Default:** ---

**Description:** Checks job access to specified QoS `<QOSID>`.

**Example:**
```
> checkjob -q special 6235
```

**-r (Reservation)**

**Format:** `<RSVID>`

**Default:** ---

**Description:** Checks job access to specified reservation `<RSVID>`.

**Example:**
```
> checkjob -r orion.1 6235
```

**-v [-v] (Verbose)**

**Format:**

**Default:** N/A

**Description:**

Sets verbose mode.

- `-v`
  - Shows why nodes are available or unavailable
  - Shows array information (see Example 2).
- `-v -v`
  - Shows timestamps to error messages
  - Shows job script
  - Shows priority calculation

**Example:**
```
> checkjob -v 6235
> checkjob -v -v 6326
```

**Details**

This command allows any Moab administrator to check the detailed status and resource requirements of a
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:

<table>
<thead>
<tr>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job has hold in place</td>
<td>one or more job holds are currently in place</td>
</tr>
<tr>
<td>insufficient idle procs</td>
<td>there are currently not adequate processor resources available to start the job</td>
</tr>
<tr>
<td>idle procs do not meet requirements</td>
<td>adequate idle processors are available but these do not meet job requirements</td>
</tr>
<tr>
<td>start date not reached</td>
<td>job has specified a minimum start date which is still in the future</td>
</tr>
<tr>
<td>expected state is not idle</td>
<td>job is in an unexpected state</td>
</tr>
<tr>
<td>state is not idle</td>
<td>job is not in the idle state</td>
</tr>
<tr>
<td>dependency is not met</td>
<td>job depends on another job reaching a certain state</td>
</tr>
<tr>
<td>rejected by policy</td>
<td>job start is prevented by a throttling policy</td>
</tr>
</tbody>
</table>

If a job cannot run on a particular node, one of the following 'per node' reasons will be given:

<table>
<thead>
<tr>
<th>Class</th>
<th>Node does not allow required job class/queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Node does not possess required processors</td>
</tr>
<tr>
<td>Disk</td>
<td>Node does not possess required local disk</td>
</tr>
<tr>
<td>Features</td>
<td>Node does not possess required node features</td>
</tr>
<tr>
<td>Memory</td>
<td>Node does not possess required real memory</td>
</tr>
<tr>
<td>Network</td>
<td>Node does not possess required network interface</td>
</tr>
<tr>
<td>State</td>
<td>Node is not Idle or Running</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account</td>
<td>&lt;STRING&gt;</td>
<td>Name of account associated with job</td>
</tr>
<tr>
<td><strong>Actual Run Time</strong></td>
<td>[[[DD:]HH:]MM:]SS</td>
<td>Length of time job actually ran.</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td><strong>Allocated Nodes</strong></td>
<td>Square bracket delimited list of node and processor ids</td>
<td>List of nodes and processors allocated to job</td>
</tr>
<tr>
<td><strong>Arch</strong></td>
<td>&lt;STRING&gt;</td>
<td>Node architecture required by job</td>
</tr>
<tr>
<td><strong>Attr</strong></td>
<td>Square bracket delimited list of job attributes</td>
<td>Job Attributes (i.e. [BACKFILL][BENCHMARK][PREEMPTEE])</td>
</tr>
<tr>
<td><strong>Average Utilized Procs</strong></td>
<td>&lt;FLOAT&gt;</td>
<td>Average load balance for a job</td>
</tr>
<tr>
<td><strong>Avg Util Resources Per Task</strong></td>
<td>&lt;FLOAT&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>Bypass</strong></td>
<td>&lt;INTEGER&gt;</td>
<td>Number of times a lower priority job with a later submit time ran before the job</td>
</tr>
<tr>
<td><strong>Class</strong></td>
<td>[&lt;CLASS NAME&gt; &lt;CLASS COUNT&gt;]</td>
<td>Name of class/queue required by job and number of class initiators required per task.</td>
</tr>
<tr>
<td><strong>Dedicated Resources Per Task</strong></td>
<td>&lt;INTEGER&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>Disk</strong></td>
<td>&lt;INTEGER&gt;</td>
<td>Amount of local disk required by job (in MB)</td>
</tr>
<tr>
<td><strong>Estimated Walltime</strong></td>
<td>[[[DD:]HH:]MM:]SS</td>
<td>The scheduler's estimated walltime. In simulation mode, it is the actual walltime.</td>
</tr>
<tr>
<td><strong>Exec Size</strong></td>
<td>&lt;INTEGER&gt;</td>
<td>Size of job executable (in MB)</td>
</tr>
<tr>
<td><strong>Executable</strong></td>
<td>&lt;STRING&gt;</td>
<td>Name of command to run</td>
</tr>
<tr>
<td><strong>Features</strong></td>
<td>Square bracket delimited list of &lt;STRING&gt;s</td>
<td>Node features required by job</td>
</tr>
<tr>
<td><strong>Flags</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Group</strong></td>
<td>&lt;STRING&gt;</td>
<td>Name of Unix group associated with job</td>
</tr>
<tr>
<td><strong>Holds</strong></td>
<td>Zero or more of User, System, and Batch</td>
<td>Types of job holds currently applied to job</td>
</tr>
<tr>
<td><strong>Image Size</strong></td>
<td>&lt;INTEGER&gt;</td>
<td>Size of job data (in MB)</td>
</tr>
<tr>
<td><strong>IWD (Initial Working Directory)</strong></td>
<td>&lt;DIR&gt;</td>
<td>Directory to run the executable in</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>&lt;INTEGER&gt;</td>
<td>Amount of real memory required per node (in MB)</td>
</tr>
<tr>
<td><strong>Max Util Resources Per Task</strong></td>
<td>&lt;FLOAT&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>Network</strong></td>
<td>&lt;STRING&gt;</td>
<td>Type of network adapter required by job</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td><strong>NodeAccess</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Nodecount</strong></td>
<td>&lt;INTEGER&gt;</td>
<td>Number of nodes required by job</td>
</tr>
<tr>
<td><strong>Opsys</strong></td>
<td>&lt;STRING&gt;</td>
<td>Node operating system required by job</td>
</tr>
<tr>
<td><strong>Partition Mask</strong></td>
<td>ALL or colon delimited list of partitions</td>
<td>List of partitions the job has access to</td>
</tr>
<tr>
<td><strong>PE</strong></td>
<td>&lt;FLOAT&gt;</td>
<td>Number of processor-equivalents requested by job</td>
</tr>
<tr>
<td><strong>QOS</strong></td>
<td>&lt;STRING&gt;</td>
<td>Quality of Service associated with job</td>
</tr>
<tr>
<td><strong>Reservation</strong></td>
<td>&lt;RSVID&gt;</td>
<td>RESID specifies the reservation id, TIME1 is the relative start time, TIME2 the relative end time, TIME3 the duration of the reservation</td>
</tr>
<tr>
<td><strong>Req</strong></td>
<td>[&lt;INTEGER&gt;] TaskCount: &lt;INTEGER&gt; Partition: &lt;partition&gt;</td>
<td>A job requirement for a single type of resource followed by the number of tasks instances required and the appropriate partition</td>
</tr>
<tr>
<td><strong>StartCount</strong></td>
<td>&lt;INTEGER&gt;</td>
<td>Number of times job has been started by Moab</td>
</tr>
<tr>
<td><strong>StartPriority</strong></td>
<td>&lt;INTEGER&gt;</td>
<td>Start priority of job</td>
</tr>
<tr>
<td><strong>StartTime</strong></td>
<td>&lt;TIME&gt;</td>
<td>Time job was started by the resource management system</td>
</tr>
<tr>
<td><strong>State</strong></td>
<td>One of Idle, Starting, Running, etc</td>
<td>Current Job State</td>
</tr>
<tr>
<td><strong>SubmitTime</strong></td>
<td>&lt;TIME&gt;</td>
<td>Time job was submitted to resource management system</td>
</tr>
<tr>
<td><strong>Swap</strong></td>
<td>&lt;INTEGER&gt;</td>
<td>Amount of swap disk required by job (in MB)</td>
</tr>
<tr>
<td><strong>Task Distribution</strong></td>
<td>Square bracket delimited list of nodes</td>
<td></td>
</tr>
<tr>
<td><strong>Time Queued</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total Nodes</strong></td>
<td>&lt;INTEGER&gt;</td>
<td>Number of nodes requested by job</td>
</tr>
<tr>
<td><strong>Total Tasks</strong></td>
<td>&lt;INTEGER&gt;</td>
<td>Number of tasks requested by job</td>
</tr>
<tr>
<td><strong>User</strong></td>
<td>&lt;STRING&gt;</td>
<td>Name of user submitting job</td>
</tr>
<tr>
<td><strong>Utilized Resources Per Task</strong></td>
<td>&lt;FLOAT&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>WallTime</strong></td>
<td>[[[DD:]HH:]MM:]SS of [[[DD:]HH:]MM:]SS</td>
<td>Length of time job has been running out of the specified limit</td>
</tr>
</tbody>
</table>

In the above table, fields marked with an asterisk (*) are only displayed when set or when the -v flag is specified.

**Example 1**

```
checkjob 717
> checkjob 717
job 717
State: Idle
```
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 2

```
checkjob -v
```

```
> checkjob -v medsec.1.1
job medsec.1.1 (RM job 'g01.1')

AName: medsec
    # This information is provided only when
Job Array Info:
    # the job is part of a job array.
Name: medsec.1.1
    # If not part of a job array, this information is not displayed.
1 : medsec.1.1 : Running
2 : medsec.1.2 : Running

Active: 2
Idle: 0
Migrated: 0
Complete: 0

State: Running
Creds: user:testuser1 group:testgroup1
WallTime: 00:21:53 of 00:05:00
SubmitTime: Tue Mar 1 11:54:50
    (Time Queued Total: 00:00:00 Eligible: 00:00:00)
Startime: Tue Mar 1 11:54:50
Total Requested Tasks: 1
Total Requested Nodes: 1

Req[0] TaskCount: 1 Partition: g01
Average Utilized Procs: 0.08
```
See Also

- 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
(Check Node)

Synopsis
checknode nodeID

Overview
This command shows detailed state information and statistics for nodes that run jobs.

The following information is returned by this command:

<table>
<thead>
<tr>
<th>Disk</th>
<th>Disk space available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>Memory available</td>
</tr>
<tr>
<td>Swap</td>
<td>Swap space available</td>
</tr>
<tr>
<td>State</td>
<td>Node state</td>
</tr>
<tr>
<td>Opsys</td>
<td>Operating system</td>
</tr>
<tr>
<td>Arch</td>
<td>Architecture</td>
</tr>
<tr>
<td>Adapters</td>
<td>Network adapters available</td>
</tr>
<tr>
<td>Features</td>
<td>Features available</td>
</tr>
<tr>
<td>Classes</td>
<td>Classes available</td>
</tr>
<tr>
<td>StateTime</td>
<td>Time node has been in current state in HH:MM:SS notation</td>
</tr>
<tr>
<td>Downtime</td>
<td>Displayed only if downtime is scheduled</td>
</tr>
<tr>
<td>Load</td>
<td>CPU Load (Berkley one-minute load average)</td>
</tr>
<tr>
<td>TotalTime</td>
<td>Total time node has been detected since statistics initialization expressed in HH:MM:SS notation</td>
</tr>
<tr>
<td>UpTime</td>
<td>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)</td>
</tr>
<tr>
<td>BusyTime</td>
<td>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)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>NODE</th>
<th>Node name you want to check.</th>
</tr>
</thead>
</table>

Flags

-h Help for this command.
-n Returns verbose output.
--xml Output in XML format. Same as mdiag -n --xml.
Example

```
> checknode P690-032
node P690-032

State:         Busy  (in current state for 11:31:10)
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 0:1]

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)
JobList:        13186
```

See Also

- **Moab Client Installation** - explains how to distribute this command to client nodes
- `mdiag -n`
- `showstate`
**mcredctl**

*(Moab Credential Control)*

**Synopsis**

```
mcredctl [-d credtype[:credid]]
           [-h credtype:credid] [-l credtype]
           [-q {role|limit|profile|accessfrom|accessto}
                credtype[:credid]] [--format=xml]
           [-r {stats|credits} credtype[:credid]]
```

**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).**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-d</code> — DESTROY</td>
<td>Purge a credential from moab.cfg (does not delete credential from memory).</td>
</tr>
<tr>
<td>Format: <code>&lt;TYPE&gt;:&lt;VAL&gt;</code></td>
<td>Default: <code>---</code></td>
</tr>
<tr>
<td>Example: <code>&gt; mcredctl -d user:john</code></td>
<td>All references to USERCFG[john] will be commented out of moab.cfg)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-l</code> — LIST</td>
<td>List the various sub-objects of the specified credential (format in XML).</td>
</tr>
<tr>
<td>Format: <code>&lt;TYPE&gt;</code></td>
<td>Default: <code>---</code></td>
</tr>
<tr>
<td>Example: <code>&gt; mcredctl -l user --format=xml</code></td>
<td>List all users within Moab in XML)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-q</code> — QUERY</td>
<td></td>
</tr>
<tr>
<td>Format: `{ role</td>
<td>accessfrom</td>
</tr>
<tr>
<td><code>limit &lt;TYPE&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>policies &lt;TYPE&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>
role USER:<USERID>
profile <TYPE>[;<VAL>]
accessfrom <TYPE>[:<VAL>]
accessto <TYPE>[:<VAL>]

Default: ---

Description: Display various aspects of a credential (formatted in XML).

Example:

View user bob's administrative role within Moab in XML)

> mcredctl -q role user:bob --format=xml

Display limits for all accounts in XML)

> mcredctl -q limit acct --format=xml

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 1

> mcredctl -d group:john
GROUPCFG[john] Successfully purged from config files

Example 2

> 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

> mcredctl -q role user:john --format=xml
<Data><user ID="test" role="admin5"></user></Data>

See Also

- Moab Client Installation - explains how to distribute this command to client nodes
**mdiag**

(Moab Diagnostics)

**Synopsis**

```markdown
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>
mdiag -f [-o user|group|acct|qos|class] [-v]
mdiag -g [groupid]
mdiag -G [Green]
mdiag -j [jobid] [-t <partition>] [-v]
mdiag -L  // diagnose usage limits
mdiag -m [rackid]
mdiag -n -A <creds> [t partition] [nodeid] [-v]
mdiag -p [-t partition] [-v]  // diagnose job priority
mdiag -q [_qosid]
mdiag -r [reservationid] [-w type=<type>] [-v]
mdiag -R [resourcemanagername] [-v]
mdiag -s [standingreservationid]
mdiag -T [triggerid] [-v]
mdiag -u [userid]
mdiag -x
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

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a [accountid]</td>
<td>display account information</td>
</tr>
<tr>
<td>-b</td>
<td>display information on jobs blocked by policies, holds, or other factors.</td>
</tr>
<tr>
<td>-c [classid]</td>
<td>display class information</td>
</tr>
<tr>
<td>-C [file]</td>
<td>analyze configuration file for errors including use of invalid parameters, deprecated parameters, and illegal values. (If you start Moab with the -e flag, Moab evaluates the configuration file at startup and quits if an error exists.)</td>
</tr>
<tr>
<td>-e</td>
<td>Moab will do a query for all events whose eventtime is between &lt;starttime&gt; and &lt;endtime&gt; and that match the search criteria. This works only when Moab is configured with ODBC. The syntax is: mdiag -e [-w &lt;starttime&gt;</td>
</tr>
<tr>
<td>-L</td>
<td>diagnose usage limits</td>
</tr>
<tr>
<td>-m [rackid]</td>
<td></td>
</tr>
<tr>
<td>-n -A &lt;creds&gt;</td>
<td></td>
</tr>
<tr>
<td>-p [-t partition]</td>
<td></td>
</tr>
<tr>
<td>-q [qosid]</td>
<td></td>
</tr>
<tr>
<td>-r [reservationid]</td>
<td></td>
</tr>
<tr>
<td>-R [resourcemanagername]</td>
<td></td>
</tr>
<tr>
<td>-s [standingreservationid]</td>
<td></td>
</tr>
<tr>
<td>-T [triggerid]</td>
<td></td>
</tr>
<tr>
<td>-u [userid]</td>
<td></td>
</tr>
<tr>
<td>-x</td>
<td></td>
</tr>
</tbody>
</table>

Starttime default is -
endtime default is INFINITY
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)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-f</td>
<td>display fairshare information</td>
</tr>
<tr>
<td>-g [groupid]</td>
<td>display group information</td>
</tr>
<tr>
<td>-G [Green]</td>
<td>display power management information</td>
</tr>
<tr>
<td>-j [jobid]</td>
<td>display job information</td>
</tr>
<tr>
<td>-L</td>
<td>display limits</td>
</tr>
<tr>
<td>-m rackid</td>
<td>display rack/frame information</td>
</tr>
<tr>
<td>-n [nodeid]</td>
<td>display nodes.</td>
</tr>
<tr>
<td>-o &lt;OTYPE&gt;[::&lt;OID&gt;]</td>
<td>organize information by specified object type</td>
</tr>
<tr>
<td>-p</td>
<td>display job priority.</td>
</tr>
<tr>
<td>-q [qosid]</td>
<td>display qos information</td>
</tr>
<tr>
<td>-r [reservationid]</td>
<td>display reservation information</td>
</tr>
<tr>
<td>-R [rmid]</td>
<td>display resource manager information</td>
</tr>
<tr>
<td>-s [srsv]</td>
<td>display standing reservation information</td>
</tr>
<tr>
<td>-S</td>
<td>display general scheduler information</td>
</tr>
<tr>
<td>-T [triggerid]</td>
<td>display trigger information</td>
</tr>
<tr>
<td>-u [userid]</td>
<td>display user information</td>
</tr>
<tr>
<td>-x</td>
<td>display advanced system information</td>
</tr>
<tr>
<td>--format=xml</td>
<td>display output in XML format</td>
</tr>
</tbody>
</table>

See Also
- Moab Client Installation - explains how to distribute this command to client nodes
- checkjob
- checknode
mdiag -a

(Moab Account Diagnostics)

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 1

```bash
> 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  FEWEIGHT=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   -  -
```

See Also

- Account credential
mdiat -b
(Moab Queues Diagnostics)

Synopsis
mdiat -b [-l policylevel] [-t partition]

Overview
The mdiag -b command returns information about blocked jobs.
**mdia -c**

( Moab Class Diagnostics )

**Synopsis**

mdia -c [-v] [classid] [--format=xml]

**Overview**

The `mdia -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 Output**

`mdia -c` information can be reported as XML as well. This is done with the command "mdia -c <CLASS_ID> --format=xml". 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>
```

In addition to the attributes listed below, `mdia -c`'s XML children describe its general statistics information (stats XML Element) and profile based statistics (Profile XML element).

**XML Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADEF</td>
<td>Accounts a class has access to.</td>
</tr>
<tr>
<td>CAPACITY</td>
<td>Number of procs available to the class.</td>
</tr>
<tr>
<td>DEFAULT.ATTR</td>
<td>Default attributes attached to a job.</td>
</tr>
<tr>
<td>DEFAULT.DISK</td>
<td>Default required disk attached to a job.</td>
</tr>
<tr>
<td>DEFAULT.FEATURES</td>
<td>Default required node features attached to a job.</td>
</tr>
<tr>
<td>DEFAULT.GRES</td>
<td>Default generic resources attached to a job.</td>
</tr>
<tr>
<td>DEFAULT.MEM</td>
<td>Default required memory attached to a job.</td>
</tr>
<tr>
<td>DEFAULT.NODESET</td>
<td>Default specified nodeset attached to a job.</td>
</tr>
<tr>
<td>DEFAULT.WCLIMIT</td>
<td>Default wallclock limit attached to a job.</td>
</tr>
<tr>
<td>EXCL.FEATURES</td>
<td>List of excluded (disallowed) node features.</td>
</tr>
<tr>
<td>EXCL.FLAGS</td>
<td>List of excluded (disallowed) job flags.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FSTARGET</td>
<td>The class' fairshare target.</td>
</tr>
<tr>
<td>HOLD</td>
<td>If TRUE this credential has a hold on it, FALSE otherwise.</td>
</tr>
<tr>
<td>HOSTLIST</td>
<td>The list of hosts in this class.</td>
</tr>
<tr>
<td>JOBEPILOG</td>
<td>Scheduler level job epilog to be run after job is completed by resource manager (script path).</td>
</tr>
<tr>
<td>JOBFLAGS</td>
<td>Default flags attached to jobs in the class.</td>
</tr>
<tr>
<td>JOBPROLOG</td>
<td>Scheduler level job prolog to be run before job is started by resource manager (script path).</td>
</tr>
<tr>
<td>ID</td>
<td>The unique ID of this class.</td>
</tr>
<tr>
<td>LOGLEVEL</td>
<td>The log level attached to jobs in the class.</td>
</tr>
<tr>
<td>MAX.PROC</td>
<td>The max processors per job in the class.</td>
</tr>
<tr>
<td>MAX.PS</td>
<td>The max processor-seconds per job in the class.</td>
</tr>
<tr>
<td>MAX.WCLIMIT</td>
<td>The max wallclock limit per job in the class.</td>
</tr>
<tr>
<td>MAXIJOB</td>
<td>The max idle jobs in the class.</td>
</tr>
<tr>
<td>MAXIPROC</td>
<td>The max idle processors in the class.</td>
</tr>
<tr>
<td>MAXJOBPERUSER</td>
<td>The max jobs per user.</td>
</tr>
<tr>
<td>MAXNODEPERJOB</td>
<td>The max nodes per job.</td>
</tr>
<tr>
<td>MAXNODEPERUSER</td>
<td>The max nodes per user.</td>
</tr>
<tr>
<td>MAXPROCPERJOB</td>
<td>The max processors per job.</td>
</tr>
<tr>
<td>MAXPROCERNODE</td>
<td>The max processors per node.</td>
</tr>
<tr>
<td>MAXPROCPERUSER</td>
<td>The max processors per user.</td>
</tr>
<tr>
<td>MIN.NODE</td>
<td>The minimum nodes per job in the class.</td>
</tr>
<tr>
<td>MIN.PROC</td>
<td>The minimum processors per job in the class.</td>
</tr>
<tr>
<td>MIN.WCLIMIT</td>
<td>The minimum wallclock limit per job in the class.</td>
</tr>
<tr>
<td>NODEACCESSPOLICY</td>
<td>The node access policy associated with jobs in the class.</td>
</tr>
<tr>
<td>OCDPROCFACTOR</td>
<td>Dedicated processor factor.</td>
</tr>
<tr>
<td>OCNODE</td>
<td>Overcommit node.</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>The class' associated priority.</td>
</tr>
<tr>
<td>PRIORITYF</td>
<td>Priority calculation function.</td>
</tr>
<tr>
<td>REQ.FEATURES</td>
<td>Required features for a job to be considered in the class.</td>
</tr>
<tr>
<td>REQ.FLAGS</td>
<td>Required flags for a job to be considered in the class.</td>
</tr>
<tr>
<td>REQ.IMAGE</td>
<td>Required image for a job to be considered in the class.</td>
</tr>
<tr>
<td>REQUIREDUSERLIST</td>
<td>The list of users who have access to the class.</td>
</tr>
<tr>
<td>RM</td>
<td>The resource manager reporting the class.</td>
</tr>
<tr>
<td>RMLIST</td>
<td>The list of resource managers who have access to the class.</td>
</tr>
</tbody>
</table>
### STATE
The class’ state.

### WCOVERRUN
Tolerated amount of time beyond the specified wall clock limit.

#### Example 1

```bash
> mdiag -c
```

Class/Queue Status

<table>
<thead>
<tr>
<th>ClassID</th>
<th>Priority</th>
<th>Flags</th>
<th>QDef</th>
<th>QOSList*</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0.00</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>[A][B]</td>
</tr>
<tr>
<td>batch</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>[A][B]</td>
</tr>
<tr>
<td>70.00</td>
<td>1</td>
<td>---</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>long</td>
<td>100</td>
<td>high</td>
<td>high</td>
<td>[B]</td>
</tr>
<tr>
<td>10.00</td>
<td>100</td>
<td>high</td>
<td>high</td>
<td>[B]</td>
</tr>
<tr>
<td>fast</td>
<td>100</td>
<td>high</td>
<td>high</td>
<td>[B]</td>
</tr>
<tr>
<td>bigmem</td>
<td>1</td>
<td>low,high</td>
<td>low</td>
<td>___</td>
</tr>
<tr>
<td>10.00</td>
<td>100</td>
<td>low,high</td>
<td>low</td>
<td>___</td>
</tr>
</tbody>
</table>

The **Limits** column will display limits in the following format:

<USAGE>:<HARDLIMIT>[,<SOFTLIMIT>]

In the example above, class `fast` has **MAXJOB** soft and hard limits of 100 and 150 respectively and is currently running 8 jobs.

#### See Also
- `showstats` command - display general statistics
**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 Example 2 below). **Note:** 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 1**

Standard Fairshare Output

```
> mdiag -f

FairShare Information
Depth: 6 intervals    Interval Length: 00:20:00    Decay Rate: 0.50
FS Policy: SDEDICATEDPES
System FS Settings: Target Usage: 0.00    Flags: 0

<table>
<thead>
<tr>
<th>FSInterval</th>
<th>%</th>
<th>Target</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FSWeight</td>
<td>---</td>
<td>--------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0.0625</td>
<td>0.0312</td>
<td></td>
<td>1.0000 0.5000 0.2500 0.1250 0.0625 0.0312</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TotalUsage</td>
<td>100.00</td>
<td></td>
<td>85.3 476.1 478.9 478.5 475.5 482.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mattp</td>
<td>2.51</td>
<td></td>
<td>2.20 2.69 2.21 2.65</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.65</td>
<td>3.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jsmith</td>
<td>12.82</td>
<td></td>
<td>12.66 15.36 10.96 8.74</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.15</td>
<td>13.85</td>
<td></td>
<td>3.44 3.93 2.78 4.36 3.11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kyliem</td>
<td>3.94</td>
<td></td>
<td>4.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.66</td>
<td>4.76</td>
<td></td>
<td>4.94 5.12 5.52 3.95</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tgh</td>
<td>1.22 1.60</td>
<td></td>
<td>1.51 1.15 1.05 1.61</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>walex</td>
<td>4.73</td>
<td></td>
<td>4.67 4.31 5.67 4.49</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jimf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Example 2**

Grouping User Output by Account

mdiag -f -o acct --flags=relative
> mdiag -f -o acct --flags=relative

FairShare Information

Depth: 6 intervals   Interval Length: 00:20:00   Decay Rate: 0.50

FS Policy: SDEDICATEDPES

System FS Settings:  Target Usage: 0.00   Flags: 0

<table>
<thead>
<tr>
<th>FSInterval</th>
<th>%</th>
<th>Target</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>-------</td>
<td>1.0000</td>
<td>0.5000</td>
<td>0.2500</td>
<td>0.1250</td>
<td>0.0625</td>
</tr>
<tr>
<td>0.0625</td>
<td>0.0312</td>
<td>0.0312</td>
<td>0.0312</td>
<td>0.0312</td>
<td>0.0312</td>
<td>0.0312</td>
<td>0.0312</td>
</tr>
<tr>
<td>TotalUsage</td>
<td>100.00</td>
<td>-------</td>
<td>23.8</td>
<td>476.1</td>
<td>478.9</td>
<td>478.5</td>
<td>475.5</td>
</tr>
<tr>
<td></td>
<td>482.8</td>
<td>-------</td>
<td>482.8</td>
<td>482.8</td>
<td>482.8</td>
<td>482.8</td>
<td>482.8</td>
</tr>
</tbody>
</table>

ACCOUNT

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>dallas</td>
<td>13.12</td>
<td>15.00</td>
<td>15.42</td>
<td>12.41</td>
<td>13.19</td>
<td>13.29</td>
</tr>
<tr>
<td>mattp</td>
<td>19.47</td>
<td>------</td>
<td>15.00</td>
<td>21.66</td>
<td>16.75</td>
<td>19.93</td>
</tr>
<tr>
<td>walex</td>
<td>9.93</td>
<td>------</td>
<td>20.91</td>
<td>9.28</td>
<td>7.97</td>
<td>12.14</td>
</tr>
<tr>
<td>stevei</td>
<td>12.19</td>
<td>------</td>
<td>9.09</td>
<td>10.78</td>
<td>15.85</td>
<td>5.64</td>
</tr>
<tr>
<td>anna</td>
<td>14.77</td>
<td>------</td>
<td>16.36</td>
<td>13.54</td>
<td>17.18</td>
<td>13.55</td>
</tr>
<tr>
<td>susieb</td>
<td>43.64</td>
<td>------</td>
<td>38.64</td>
<td>44.74</td>
<td>42.25</td>
<td>48.74</td>
</tr>
</tbody>
</table>

See Also:

- Fairshare Overview
mdiag -g

(Moab Group Diagnostics)

Synopsis

mdiag -g [groupid]

Overview

The `mdiag -g` command is used to present information about groups.
mdiag -j
(Moab Job Diagnostics)

Synopsis


Overview

Themdiag -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. 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 select only jobs associated with a given credential (user, acct, class, group, qos). For example:

```bash
mdiag -j -w user=david
```

Output from the preceding example displays only the user named David's jobs.

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:

```xml
<Data>
  <job ATTR="VALUE" ... > </job>
  ...
</Data>
```

For information about legal attributes, refer to the XML Attributes table.

See Also

- checkjob
- mdiag
**mdia**g **-n**  
(Moab Node Diagnostics)

**Synopsis**

mdia -n [-t partitionid] [-A creds] [-v] [--format=xml] [nodeid]

**Overview**

The `mdia -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**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-A]</td>
<td>{user</td>
<td>group</td>
</tr>
<tr>
<td>[-t]</td>
<td>&lt;partitionid&gt;</td>
<td>report only nodes from specified partition</td>
</tr>
<tr>
<td>[-v]</td>
<td>-</td>
<td>show verbose output (do not truncate columns and add columns for additional node attributes)</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Column</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>&lt;NODE NAME&gt;</td>
</tr>
<tr>
<td>State</td>
<td>&lt;NODE STATE&gt;</td>
</tr>
<tr>
<td>Procs</td>
<td>&lt;AVAILABLE PROCs&gt;:&lt;CONFIGURED PROCs&gt;</td>
</tr>
<tr>
<td>Memory</td>
<td>&lt;AVAILABLE MEMORY&gt;:&lt;CONFIGURED MEMORY&gt;</td>
</tr>
<tr>
<td>Disk</td>
<td>&lt;AVAILABLE DISK&gt;:&lt;CONFIGURED DISK&gt;</td>
</tr>
<tr>
<td>Swap</td>
<td>&lt;AVAILABLE SWAP&gt;:&lt;CONFIGURED SWAP&gt;</td>
</tr>
<tr>
<td>Speed</td>
<td>&lt;RELATIVE MACHINE SPEED&gt;</td>
</tr>
<tr>
<td>Opsys</td>
<td>&lt;NODE OPERATING SYSTEM&gt;</td>
</tr>
<tr>
<td>Arch</td>
<td>&lt;NODE HARDWARE ARCHITECTURE&gt;</td>
</tr>
<tr>
<td>Par</td>
<td>&lt;PARTITION NODE IS ASSIGNED TO&gt;</td>
</tr>
<tr>
<td>Load</td>
<td>&lt;CURRENT 1 MINUTE BSD LOAD&gt;</td>
</tr>
<tr>
<td>Rsv</td>
<td>&lt;NUMBER OF RESERVATIONS ON NODE&gt;</td>
</tr>
<tr>
<td>Classes</td>
<td>&lt;CLASS NAME&gt;&lt;CLASS INSTANCES AVAILABLE&gt;:&lt;CLASS INSTANCES CONFIGURED&gt;...</td>
</tr>
<tr>
<td>Network</td>
<td>&lt;NETWORK NAME&gt;...</td>
</tr>
<tr>
<td>Features</td>
<td>&lt;NODE FEATURE&gt;...</td>
</tr>
</tbody>
</table>
Example 1

> mdiag -n

```plaintext
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
```

**Note:** 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:

```
mdig -n --format=xml

<Data>
  <node> <ATTR>="<VAL>" ... </node>
  ...
</Data>
```

In addition to the attributes listed below, mdiag -n's node element XML has children that describe a node's messages (Messages XML element).

**XML Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGRES</td>
<td>Available generic resources</td>
</tr>
<tr>
<td>ALLOCRES</td>
<td>Special allocated resources (like vlans)</td>
</tr>
<tr>
<td>ARCH</td>
<td>The node's processor architecture.</td>
</tr>
<tr>
<td>AVLCLASS</td>
<td>Classes available on the node</td>
</tr>
<tr>
<td>AVLETIME</td>
<td>Time when the node will no longer be available (used in Utility centers)</td>
</tr>
<tr>
<td><strong>AVLSTIME</strong></td>
<td>Time when the node will be available (used in Utility centers)</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>CFGCLASS</strong></td>
<td>Classes configured on the node</td>
</tr>
<tr>
<td><strong>GRES</strong></td>
<td>generic resources on the node</td>
</tr>
<tr>
<td><strong>ENABLEPROFILING</strong></td>
<td>If true, a node's state and usage is tracked over time.</td>
</tr>
<tr>
<td><strong>FEATURES</strong></td>
<td>A list of comma separated custom features describing a node.</td>
</tr>
<tr>
<td><strong>GEMETRIC</strong></td>
<td>A list of comma separated consumable resources associated with a node.</td>
</tr>
<tr>
<td><strong>HOPCOUNT</strong></td>
<td>How many hops the node took to reach this Moab (used in hierarchical grids)</td>
</tr>
<tr>
<td><strong>ISDELETED</strong></td>
<td>Node has been deleted</td>
</tr>
<tr>
<td><strong>ISDYNAMIC</strong></td>
<td>Node is dynamic (used in Utility centers)</td>
</tr>
<tr>
<td><strong>JOBLIST</strong></td>
<td>The list of jobs currently running on a node.</td>
</tr>
<tr>
<td><strong>LOAD</strong></td>
<td>current load as reported by the resource manager</td>
</tr>
<tr>
<td><strong>LOADWEIGHT</strong></td>
<td>load weight used when calculating node priority</td>
</tr>
<tr>
<td><strong>MAXJOB</strong></td>
<td>See Node Policies for details.</td>
</tr>
<tr>
<td><strong>MAXJOBPERUSER</strong></td>
<td>See Node Policies for details.</td>
</tr>
<tr>
<td><strong>MAXLOAD</strong></td>
<td>See Node Policies for details.</td>
</tr>
<tr>
<td><strong>MAXPROC</strong></td>
<td>See Node Policies for details.</td>
</tr>
<tr>
<td><strong>MAXPROCPERUSER</strong></td>
<td>See Node Policies for details.</td>
</tr>
<tr>
<td><strong>NETWORK</strong></td>
<td>The ability to specify which networks are available to a given node is limited to only a few resource manager. 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.</td>
</tr>
<tr>
<td><strong>NODEID</strong></td>
<td>The unique identifier for a node.</td>
</tr>
<tr>
<td><strong>NODESTATE</strong></td>
<td>The state of a node.</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>A node's operating system.</td>
</tr>
<tr>
<td><strong>OSLIST</strong></td>
<td>Operating systems the node can run</td>
</tr>
<tr>
<td><strong>OSMODACTION</strong></td>
<td>URL for changing the operating system</td>
</tr>
<tr>
<td><strong>OWNER</strong></td>
<td>Credential type and name of owner</td>
</tr>
<tr>
<td><strong>PARTITION</strong></td>
<td>The partition a node belongs to. See Node Location for details.</td>
</tr>
<tr>
<td><strong>POWER</strong></td>
<td>The state of the node's power. Either ON or OFF.</td>
</tr>
<tr>
<td><strong>PRIORITY</strong></td>
<td>The fixed node priority relative to other nodes.</td>
</tr>
<tr>
<td><strong>PROCSPEED</strong></td>
<td>A node's processor speed information specified in MHz.</td>
</tr>
<tr>
<td><strong>RACK</strong></td>
<td>The rack associated with a node's physical location.</td>
</tr>
<tr>
<td><strong>RADISK</strong></td>
<td>The total available disk on a node.</td>
</tr>
<tr>
<td><strong>RAMEM</strong></td>
<td>The total available memory available on a node.</td>
</tr>
<tr>
<td><strong>RAPROC</strong></td>
<td>The total number of processors available on a node.</td>
</tr>
<tr>
<td><strong>Attribute</strong></td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>RASWAP</strong></td>
<td>The total available swap on a node.</td>
</tr>
<tr>
<td><strong>RCMEM</strong></td>
<td>The total configured memory on a node.</td>
</tr>
<tr>
<td><strong>RCPROC</strong></td>
<td>The total configured processors on a node.</td>
</tr>
<tr>
<td><strong>RCSWAP</strong></td>
<td>The total configured swap on a node.</td>
</tr>
<tr>
<td><strong>RESCOUNT</strong></td>
<td>Number of reservations on the node</td>
</tr>
<tr>
<td><strong>RSVLIST</strong></td>
<td>List of reservations on the node</td>
</tr>
<tr>
<td><strong>RESOURCES</strong></td>
<td>Deprecated (use GRES)</td>
</tr>
<tr>
<td><strong>RMACCESSLIST</strong></td>
<td>A comma separated list of resource managers who have access to a node.</td>
</tr>
<tr>
<td><strong>SIZE</strong></td>
<td>The number of slots or size units consumed by the node.</td>
</tr>
<tr>
<td><strong>SLOT</strong></td>
<td>The first slot in the rack associated with the node's physical location.</td>
</tr>
<tr>
<td><strong>SPEED</strong></td>
<td>A node's relative speed.</td>
</tr>
<tr>
<td><strong>SPEEDWEIGHT</strong></td>
<td>Speed weight used to calculate node's priority</td>
</tr>
<tr>
<td><strong>STATACTIVETIME</strong></td>
<td>Time node was active</td>
</tr>
<tr>
<td><strong>STATMODIFYTIME</strong></td>
<td>Time node's state was modified</td>
</tr>
<tr>
<td><strong>STATTOTALTIME</strong></td>
<td>Time node has been monitored</td>
</tr>
<tr>
<td><strong>STATUPTIME</strong></td>
<td>Time node has been up</td>
</tr>
<tr>
<td><strong>TASKCOUNT</strong></td>
<td>The number of tasks on a node.</td>
</tr>
</tbody>
</table>

**See Also**

- checknode
**mdia -p**

*(Moab Priority Diagnostics)*

**Synopsis**

`mdia -p [-t partition] [-v]`

**Overview**

The `mdia -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.

**Note:** 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**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Name</th>
<th>Format</th>
<th>Default</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-v</td>
<td>VERBOSE</td>
<td>---</td>
<td>---</td>
<td>display verbose priority information. If specified, display priority breakdown information for blocked, eligible, and active jobs. <strong>Note:</strong> By default, only information for eligible jobs is displayed. To view blocked jobs in addition to eligible, run <code>mdia -p -v -v</code>.</td>
<td><code>mdia -p</code> <code>mdia -p -v</code> <code>mdia -p -v</code></td>
</tr>
</tbody>
</table>

**Output**

<table>
<thead>
<tr>
<th>Priority Component</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td><code>&lt;PERCENT&gt;()</code></td>
<td>QOS: QOS associated with job QOSPRI: Priority assigned to the QOS target</td>
</tr>
<tr>
<td>QOS</td>
<td><code>&lt;PERCENT&gt;(QOS:&lt;QOSPRI&gt;)</code></td>
<td>USR: target fs usage - user fs GRP: group fs usage - group fs ACC: account fs usage - account QOS fs usage - QOS fs CLS: class fs usage - class fs</td>
</tr>
<tr>
<td>FairShare</td>
<td><code>&lt;PERCENT&gt;(USR:GRP:ACC:QOS:CLS)</code></td>
<td>QTime: job queue time which is applicable towards priority (in minutes)</td>
</tr>
</tbody>
</table>
Example 1

mdiag -p

diagnosing job priority information (partition: ALL)

<table>
<thead>
<tr>
<th>Job</th>
<th>PRIORITY*</th>
<th>Cred(QOS)</th>
<th>FS(Accnt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serv(QTime)</td>
<td>Weights</td>
<td>1( 1)</td>
<td>1( 1)</td>
</tr>
<tr>
<td>13678</td>
<td>1321*</td>
<td>7.6(100.0)</td>
<td>0.2( 2.7)</td>
</tr>
<tr>
<td>92.2(1218.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13698</td>
<td>235*</td>
<td>42.6(100.0)</td>
<td>1.1( 2.7)</td>
</tr>
<tr>
<td>56.3(132.3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13019</td>
<td>8699</td>
<td>0.6( 50.0)</td>
<td>0.3( 25.4)</td>
</tr>
<tr>
<td>99.1(8674.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13030</td>
<td>8699</td>
<td>0.6( 50.0)</td>
<td>0.3( 25.4)</td>
</tr>
<tr>
<td>99.1(8674.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13099</td>
<td>8537</td>
<td>0.6( 50.0)</td>
<td>0.3( 25.4)</td>
</tr>
<tr>
<td>13141</td>
<td>8438</td>
<td>0.6( 50.0)</td>
<td>0.2( 17.6)</td>
</tr>
<tr>
<td>13146</td>
<td>8428</td>
<td>0.6( 50.0)</td>
<td>0.2( 17.6)</td>
</tr>
<tr>
<td>99.2(8370.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13153</td>
<td>8360</td>
<td>0.0( 1.0)</td>
<td>0.1( 11.6)</td>
</tr>
<tr>
<td>13177</td>
<td>8261</td>
<td>0.0( 1.0)</td>
<td>0.1( 11.6)</td>
</tr>
<tr>
<td>99.8(8203.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13203</td>
<td>8127</td>
<td>0.6( 50.0)</td>
<td>0.3( 25.4)</td>
</tr>
<tr>
<td>99.1(8102.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13211</td>
<td>8098</td>
<td>0.0( 1.0)</td>
<td>0.1( 11.6)</td>
</tr>
<tr>
<td>99.8(8085.)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 user, group, and account fairshare weights are set to 5, 1, and 1 respectively.

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 \textit{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.

See Also

- Job Priority Overview
- Moab Cluster Manager - Priority Manager
**Synopsis**

```bash
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.

**Example 1: 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**

*(Moab Reservation Diagnostics)*

**Synopsis**

```plaintext
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, start time and end time, 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 `-w` flag filters the output according to the type of reservation. The allowable reservation types are, **None**, **Job**, **User**, and **Meta**. The **None** type means there is no filtering performed.

**Example 1**

```plaintext
> mdiag -r
Diagnosing Reservations
RsvID                      Type Par   StartTime     EndTime
-----                      ---- ---   ---------     -------     ---
----- ---- ---- ----
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: PROCES: [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: PROCES: [ALL]
Attributes (HostExp='fr3n01 fr3n03 fr3n05 fr3n07 fr3n09
fr3n11 fr3n13 fr3n15')
```
**Synopsis**

`mdiag -R [-v] [-V job] [resourcemanagerid]`

**Overview**

The `mdiag -R` command is used to present information about configured resource managers. The information presented includes name, host, port, state, type, performance statistics and failure notifications.

**Example 1**

```plaintext
> mdiag -R -v

RM[base]  Type: PBS  State: Active  ResourceType: COMPUTE
  Version:  '1.2.0p6-snap.1124480497'
  Nodes Reported:  4
  Flags:  executionServer,noTaskOrdering,typeIsExplicit
  Partition:  base
  Event Management:  EPORT=15004
  Note:  SSS protocol enabled
  Submit Policy:  NODECENTRIC
  DefaultClass:  batch
  Variables:  DefaultApp=MPICHG2,GridNet=Infiniband
  RM Performance:  AvgTime=0.00s  MaxTime=1.03s  (1330 samples)

RM[base] Failures:
  Mon May  3 09:15:16  clusterquery  'cannot get node info (rm is unavailable)'
  Mon May  3 10:25:46  workloadquery  'cannot get job info (request timed out)'

RM[Boeing]  Type: NATIVE  State: Active  ResourceType: LICENSE
  Cluster Query URL:  file://$HOME/lic.dat
  Licenses Reported:  3 types (3 of 6 available)
  Partition:  SHARED
  License Stats:  Avg License Avail:  0.00  (438 iterations)
  Iteration Summary:  Idle: 0.00  Active: 100.00  Busy: 0.00
  RM Performance:  AvgTime=0.00s  MaxTime=0.00s  (877 samples)

RM[GM]  Type: NATIVE  State: Active  ResourceType: COMPUTE
```
**mdiag -S**

*(Moab Scheduler Diagnostics)*

**Synopsis**

`mdiag -S [-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**

**Example 1**

```
> 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)
```
**mddiag -t**  
*(Moab Partition Diagnostics)*

**Synopsis**

```bash
mddiag -t [-v] [-v] [partitionid]
```

**Overview**

The `mddiag -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.

**Example 1: Standard Partition Diagnostics**

```
> mdiag -t
Partition Status
...
```
**mdiag -T**

(Moab Trigger Diagnostics)

**Synopsis**

`mdiag -T [triggerid]`

**Overview**

The `mdiag -T` command is used to present information about each Trigger. The information presented includes Name, State, Action, Event Time.

**Example 1**

```
> mdiag -T
[test@node01 moab]$ mdiag -T
<table>
<thead>
<tr>
<th>TrigID</th>
<th>Event</th>
<th>MFire</th>
<th>Offset</th>
<th>Thresh</th>
<th>AType</th>
<th>ActionDate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>exec</td>
<td>-00:00:02</td>
</tr>
<tr>
<td>Active</td>
<td>-00:00:01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object:</td>
<td>rsv:test.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PID:</td>
<td>23786</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timeout:</td>
<td>00:01:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action Data:</td>
<td>$HOME/bin/1.sh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sets on Success:</td>
<td>Initialized</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output Buffer:</td>
<td>/opt/moab/spool/1.sh.oWo5APo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Buffer:</td>
<td>/opt/moab/spool/1.sh.epbq65C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>start</td>
<td>FALSE</td>
<td></td>
<td>-</td>
<td>exec</td>
<td>-00:00:02</td>
</tr>
<tr>
<td>Inactive</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object:</td>
<td>rsv:test.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action Data:</td>
<td>$HOME/bin/2.sh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sets on Success:</td>
<td>COLOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requires:</td>
<td>Initialized</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>cancel</td>
<td>FALSE</td>
<td></td>
<td>-</td>
<td>exec</td>
<td>-</td>
</tr>
<tr>
<td>Inactive</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object:</td>
<td>rsv:test.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action Data:</td>
<td>$HOME/bin/cancel.email $USERNAME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requires:</td>
<td>$USERNAME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
**mdiau -u**

*(Moab User Diagnostics)*

**Synopsis**

`mdiau -u [userid]`

**Overview**

The `mdiau -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.

**Example 1**

```plaintext
> mdiau -u

evaluating user information

<table>
<thead>
<tr>
<th>Name</th>
<th>Priority</th>
<th>Flags</th>
<th>QDef</th>
<th>QOSList*</th>
</tr>
</thead>
<tbody>
<tr>
<td>jvella</td>
<td>0</td>
<td>[NONE]</td>
<td>[NONE]</td>
<td>[NONE]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ALIST=Engineering</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Message: profiling enabled (597 of 3000 samples/00:15:00 interval)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[NONE]</td>
<td>[NONE]</td>
<td>[NONE]</td>
</tr>
<tr>
<td>reynolds</td>
<td>0</td>
<td>[NONE]</td>
<td>[NONE]</td>
<td>[NONE]</td>
</tr>
<tr>
<td>ALIST=Administration</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message: profiling enabled (597 of 3000 samples/00:15:00 interval)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mshaw</td>
<td>0</td>
<td>[NONE]</td>
<td>[NONE]</td>
<td>[NONE]</td>
</tr>
<tr>
<td>ALIST=Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message: profiling enabled (584 of 3000 samples/00:15:00 interval)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kforbes</td>
<td>0</td>
<td>[NONE]</td>
<td>[NONE]</td>
<td>[NONE]</td>
</tr>
<tr>
<td>ALIST=Shared</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message: profiling enabled (597 of 3000 samples/00:15:00 interval)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gastor</td>
<td>0</td>
<td>[NONE]</td>
<td>[NONE]</td>
<td>[NONE]</td>
</tr>
<tr>
<td>ALIST=Engineering</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message: profiling enabled (597 of 3000 samples/00:15:00 interval)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Note that only users which have jobs which are currently queued or have been queued since Moab was most recently started are listed.

**See Also**

- `showstats` command (display user statistics)
mjobctl
(Moab Job Control)

Synopsis

mjobctl -c jobexp
mjobctl -c -w attr=val
mjobctl -c jobexp
mjobctl -h [User|System|Batch|Defer|All] jobexp
mjobctl -m attr{=|=|=}val jobexp
mjobctl -N [<SIGNO>] jobexp
mjobctl -n <JOBNAME>
mjobctl -p <PRIORIT Y> jobexp
mjobctl -q {diag|starttime|hostlist} jobexp
mjobctl -r jobexp
mjobctl -R jobexp
mjobctl -s jobexp
mjobctl -u [User|System|Batch|Defer|All] jobexp
mjobctl -w attr{=|=|=}val jobexp
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 11.0 General Job Administration for more details on jobs and their attributes.

Format

-c — Cancel

| Format: | JOBEXP |
| Default: | --- |
| 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> |
| Default: | --- |
| 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.)

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.
### -C — Checkpoint

**Format:**

\[ \text{JOBEXP} \]

**Default:** ---

**Description:** Checkpoint a job

**Example:**

> mjobctl -C job1045

Checkpoint job job1045.

### -h — Hold

**Format:**

\[ <\text{HOLDTYPE}> \text{JOBEXP} \]

\[ <\text{HOLDTYPE}> = \{ \text{user} | \text{batch} | \text{system} | \text{defer} | \text{ALL} \} \]

**Default:** user

**Description:** Set or release a job hold

See Section 11.1, Job Holds 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:**

\[ <\text{ATTR}> \{ += | = | -= \} <\text{VAL}> \text{JOBEXP} \]

\[ <\text{ATTR}> = \{ \text{account} | \text{awduration} | \text{class} | \text{deadline} | \text{depend} | \text{eeduration} | \text{env} | \text{features} | \text{feature} | \text{flags} | \text{gres} | \text{group} | \text{hold} | \text{hostlist} | \text{jobdisk} | \text{jobmem} | \text{jobname} | \text{jobswap} | \text{loglevel} | \text{messages} | \text{minstarttime} | \text{nodecount} | \text{notificationaddress} | \text{partition} | \text{priority} | \text{proccount} | \text{queue} | \text{qos} | \text{reqreservation} | \text{rmxstring} | \text{reqawduration} | \text{sysprio} | \text{trig} | \text{trigvar} | \text{userprio} | \text{var} | \text{wclimit} \} \]

**Default:** ---

**Description:**

Modify a specific job attribute.

For 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`, `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.)

**Example:**

```
> mjobctl -m reqawduration+=600 1664
```

Add 10 minutes to the job walltime.

```
> mjobctl -m eeduration=-1 1664
```

Reset job’s effective queue time.

```
> 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`.

---

### -N — Notify

**Format:** `[signal=]<SIGID> JOBEXP`

**Default:** `---`

**Description:** Send a signal to all jobs matching the job expression.

**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:** `---`

**Default:** `---`

**Description:** Select jobs by job name.

**Example:**

```
---
```

---

### -p — Priority

**Format:** `[++|+=|-=]<VAL> JOBEXP [--flags=relative]`

---
**Default:** ---

**Description:** Modify a job's system priority.

**Example:** The use of different operands with this command produces different results. Using the format `<VAL> <JOBID>` or `+<VAL> <JOBID>` will produce the same results: the '+' does not increase priority. Additionally, `+<VAL> <JOBID>` and `<VAL> <JOBID> --flags=relative` will produce the same results of relatively increasing the job's priority (not the system's). Using the format `-<VAL> <JOBID>` sets the job priority to 0, and will not change based on `<VAL>` (it will not decrease priority by that number).

```bash
> mjobctl -p +1000 job1045
```

Adds 1000 points to the max system priority, ensuring that this job will be higher priority than all normal jobs. The new priority of job1045 is 1000001000. The system priority has not changed.

```bash
> mjobctl -p 1000 job1045 --flags=relative
```

Adds 1000 points to what the priority of the job would be from normal calculation. The new priority for job1045 is 1250. The system priority has been set to 1000.

---

**-q — Query**

**Format:** `[ diag | hostlist | starttime ] JOBEXP

**Default:** ---

**Description:** Query a job.

**Example:**

```bash
> mjobctl -q diag job1045
```

Query job1045.

```bash
> mjobctl -q starttime job1045
```

Query starttime of job job1045.

```bash
> mjobctl -q wiki <jobName>
```

Query a job with the output displayed in a WIKI string. The job’s name may be replaced with ALL.

---

**-r — Resume**

**Format:** JOBEXP

**Default:** ---

**Description:** Resume a job.

**Example:**

```bash
> mjobctl -r job1045
```

Resume job job1045.

---

**-R — Requeue**
<table>
<thead>
<tr>
<th>Format:</th>
<th>JOBEXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>Requeue a job.</td>
</tr>
<tr>
<td>Example:</td>
<td>\texttt{\textgreater{} mjobctl -R job1045} \hfill Requeue job job1045.</td>
</tr>
</tbody>
</table>

**-s — Suspend**

<table>
<thead>
<tr>
<th>Format:</th>
<th>JOBEXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>Suspend a job.</td>
</tr>
<tr>
<td>Example:</td>
<td>\texttt{\textgreater{} mjobctl -s job1045} \hfill Suspend job job1045.</td>
</tr>
</tbody>
</table>

**-S — Submit**

<table>
<thead>
<tr>
<th>Format:</th>
<th>JOBEXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>Submit a job.</td>
</tr>
<tr>
<td>Example:</td>
<td>\texttt{\textgreater{} mjobctl -S job1045} \hfill Submit job job1045.</td>
</tr>
</tbody>
</table>

**-u — Unhold**

<table>
<thead>
<tr>
<th>Format:</th>
<th>[\textsc{&lt;TYPE&gt;}[,\textsc{&lt;TYPE&gt;}] ] JOBEXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textsc{&lt;TYPE&gt;}</td>
<td>= [ \textsc{user}</td>
</tr>
<tr>
<td>Default:</td>
<td>ALL</td>
</tr>
<tr>
<td>Description:</td>
<td>Release a hold on a job</td>
</tr>
<tr>
<td>Example:</td>
<td>\texttt{\textgreater{} mjobctl -u user,system scrib.1045} \hfill Release user and system holds on job scrib.1045.</td>
</tr>
</tbody>
</table>

**-w — Where**

| Format: | [CompletionTime | StartTime]|\textless{} | \textless{} | \textgreater{} | \textgreater{}|<EPOCH\_TIME> |
|---------|-------------|---------|-------|-------------|
| Default: | --- |
| 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: | \texttt{\textgreater{} mjobctl -q diag ALL --flags=COMPLETED --format=xml} } |
Prints all completed jobs still in memory that completed between July 1, 2009 and October 1, 2009.

-x — Execute

Format: JOBEXP

Default: ---

Description: Execute a job. The -w option allows flags to be set for the job. Allowable flags are, ignorepolicies, 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>

Default: ---

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.

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, either specify the expression using a designated expression or wildcard character (one of '[^]*?^$') 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, treat 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 (i.e. "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 -c job[1-2],job4 or mjobctl -c job1,job2,job4.

Example:

> mjobctl -c "80.*"
job '802' cancelled
Cancel all jobs starting with '80'.

```bash
> mjobctl -m priority+=200 "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>". In addition to the attributes listed below, `mjobctl`'s XML children describe a job's requirements (`req` XML element) and messages (`Messages` XML element).

### XML Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account</td>
<td>the account assigned to the job</td>
</tr>
<tr>
<td>Alloc NodeList</td>
<td>the nodes allocated to the job</td>
</tr>
<tr>
<td>Args</td>
<td>the job's executable arguments</td>
</tr>
<tr>
<td>AWDuration</td>
<td>the active wall time consumed</td>
</tr>
<tr>
<td>BlockReason</td>
<td>the block message index for the reason the job is not eligible</td>
</tr>
<tr>
<td>Bypass</td>
<td>Number of times the job has been bypassed by other jobs</td>
</tr>
<tr>
<td>Calendar</td>
<td>the job's timeframe constraint calendar</td>
</tr>
<tr>
<td>Class</td>
<td>the class assigned to the job</td>
</tr>
<tr>
<td>CmdFile</td>
<td>the command file path</td>
</tr>
<tr>
<td>Completion Code</td>
<td>the return code of the job as extracted from the RM</td>
</tr>
<tr>
<td>Completion Time</td>
<td>the time of the job's completion</td>
</tr>
<tr>
<td>Cost</td>
<td>the cost of executing the job relative to an allocation manager</td>
</tr>
<tr>
<td>CPULimit</td>
<td>the CPU limit for the job</td>
</tr>
<tr>
<td><strong>Depend</strong></td>
<td>any dependencies on the status of other jobs</td>
</tr>
<tr>
<td><strong>DRM</strong></td>
<td>the master destination RM</td>
</tr>
<tr>
<td><strong>DRMJID</strong></td>
<td>the master destination RM job ID</td>
</tr>
<tr>
<td><strong>EEDuration</strong></td>
<td>the duration of time the job has been eligible for scheduling</td>
</tr>
<tr>
<td><strong>EFile</strong></td>
<td>the stderr file</td>
</tr>
<tr>
<td><strong>Env</strong></td>
<td>the job's environment variables set for execution</td>
</tr>
<tr>
<td><strong>EnvOverride</strong></td>
<td>the job's overriding environment variables set for execution</td>
</tr>
<tr>
<td><strong>EState</strong></td>
<td>the expected state of the job</td>
</tr>
<tr>
<td><strong>EstHistStartTime</strong></td>
<td>the estimated historical start time</td>
</tr>
<tr>
<td><strong>EstPrioStartTime</strong></td>
<td>the estimated priority start time</td>
</tr>
<tr>
<td><strong>EstRsvStartTime</strong></td>
<td>the estimated reservation start time</td>
</tr>
<tr>
<td><strong>EstWCTime</strong></td>
<td>the estimated walltime the job will execute</td>
</tr>
<tr>
<td><strong>ExcHList</strong></td>
<td>the excluded host list</td>
</tr>
<tr>
<td><strong>Flags</strong></td>
<td>Command delimited list of Moab flags on the job</td>
</tr>
<tr>
<td><strong>GAttr</strong></td>
<td>the requested generic attributes</td>
</tr>
<tr>
<td><strong>GJID</strong></td>
<td>the global job ID</td>
</tr>
<tr>
<td><strong>Group</strong></td>
<td>the group assigned to the job</td>
</tr>
<tr>
<td><strong>Hold</strong></td>
<td>the hold list</td>
</tr>
<tr>
<td><strong>Holdtime</strong></td>
<td>the time the job was put on hold</td>
</tr>
<tr>
<td><strong>HopCount</strong></td>
<td>the hop count between the job's peers</td>
</tr>
<tr>
<td><strong>HostList</strong></td>
<td>the requested host list</td>
</tr>
<tr>
<td><strong>IFlags</strong></td>
<td>the internal flags for the job</td>
</tr>
<tr>
<td><strong>IsInteractive</strong></td>
<td>if set, the job is interactive</td>
</tr>
<tr>
<td><strong>IsRestartable</strong></td>
<td>if set, the job is restartable</td>
</tr>
<tr>
<td><strong>IsSuspendable</strong></td>
<td>if set, the job is suspendable</td>
</tr>
<tr>
<td><strong>IWD</strong></td>
<td>the directory where the job is executed</td>
</tr>
<tr>
<td><strong>JobID</strong></td>
<td>the job's batch ID.</td>
</tr>
<tr>
<td><strong>JobName</strong></td>
<td>the user-specified name for the job</td>
</tr>
<tr>
<td><strong>JobGroup</strong></td>
<td>the job ID relative to its group</td>
</tr>
<tr>
<td><strong>LogLevel</strong></td>
<td>the individual log level for the job</td>
</tr>
<tr>
<td><strong>MasterHost</strong></td>
<td>the specified host to run primary tasks on</td>
</tr>
<tr>
<td><strong>Messages</strong></td>
<td>any messages reported by Moab regarding the job</td>
</tr>
<tr>
<td><strong>MinPreemptTime</strong></td>
<td>the minimum amount of time the job must run before being eligible for preemption</td>
</tr>
<tr>
<td><strong>Notification</strong></td>
<td>any events generated to notify the job's user</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OFile</td>
<td>the stdout file</td>
</tr>
<tr>
<td>OldMessages</td>
<td>any messages reported by Moab in the old message style regarding the job</td>
</tr>
<tr>
<td>OWCLimit</td>
<td>the original wallclock limit</td>
</tr>
<tr>
<td>PAL</td>
<td>the partition access list relative to the job</td>
</tr>
<tr>
<td>QueueStatus</td>
<td>the job's queue status as generated this iteration</td>
</tr>
<tr>
<td>QOS</td>
<td>the QOS assigned to the job</td>
</tr>
<tr>
<td>QOSReq</td>
<td>the requested QOS for the job</td>
</tr>
<tr>
<td>ReqAWDuration</td>
<td>the requested active walltime duration</td>
</tr>
<tr>
<td>ReqCMaxTime</td>
<td>the requested latest allowed completion time</td>
</tr>
<tr>
<td>ReqMem</td>
<td>the total memory requested/dedicated to the job</td>
</tr>
<tr>
<td>ReqNodes</td>
<td>the number of requested nodes for the job</td>
</tr>
<tr>
<td>ReqProcs</td>
<td>the number of requested procs for the job</td>
</tr>
<tr>
<td>ReqReservation</td>
<td>the required reservation for the job</td>
</tr>
<tr>
<td>ReqRMType</td>
<td>the required RM type</td>
</tr>
<tr>
<td>ReqSMinTime</td>
<td>the requested earliest start time</td>
</tr>
<tr>
<td>RM</td>
<td>the master source resource manager</td>
</tr>
<tr>
<td>RMXString</td>
<td>the resource manager extension string</td>
</tr>
<tr>
<td>RsvAccess</td>
<td>the list of reservations accessible by the job</td>
</tr>
<tr>
<td>RsvStartTime</td>
<td>the reservation start time</td>
</tr>
<tr>
<td>RunPriority</td>
<td>the effective job priority</td>
</tr>
<tr>
<td>Shell</td>
<td>the execution shell's output</td>
</tr>
<tr>
<td>SID</td>
<td>the job's system ID (parent cluster)</td>
</tr>
<tr>
<td>Size</td>
<td>the job's computational size</td>
</tr>
<tr>
<td>STotCPU</td>
<td>the average CPU load tracked across all nodes</td>
</tr>
<tr>
<td>SMaxCPU</td>
<td>the max CPU load tracked across all nodes</td>
</tr>
<tr>
<td>STotMem</td>
<td>the average memory usage tracked across all nodes</td>
</tr>
<tr>
<td>SMaxMem</td>
<td>the max memory usage tracked across all nodes</td>
</tr>
<tr>
<td>SRMJID</td>
<td>the source RM's ID for the job</td>
</tr>
<tr>
<td>StartCount</td>
<td>the number of the times the job has tried to start</td>
</tr>
<tr>
<td>StartPriority</td>
<td>the effective job priority</td>
</tr>
<tr>
<td>StartTime</td>
<td>the most recent time the job started executing</td>
</tr>
<tr>
<td>State</td>
<td>the state of the job as reported by Moab</td>
</tr>
<tr>
<td>StatMSUtl</td>
<td>the total number of memory seconds utilized</td>
</tr>
<tr>
<td>StatPSDed</td>
<td>the total number of processor seconds dedicated to the job</td>
</tr>
</tbody>
</table>
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
UtlProcs - the number of utilized processors by the job
Variable
VWCTime - the virtual wallclock limit

Example 1

```plaintext
> 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"
```
See Also

- **Moab Client Installation** - explains how to distribute this command to client nodes
- `setspri`
- `canceljob`
- `runjob`
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

Two days, three hours and fifty-seven seconds in the future:
+02:03:0:57

Three weeks in the future:
+21:0:0:0

Thirty 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]]] ie 14:30_06/20)

Examples:

1 PM, March 1 (this year)
13:00_03/01
**mnodectl**  
*(Moab Node Control)*

**Synopsis**

```bash
mnodectl -d message:<index>  nodeexp
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 level 1 and 2 Moab administrators (see `ADMINCFG`).

**Format**

### -d — Destroy

**Format:**

```plaintext
message:<INDEX>
```

**Default:** N/A

**Description:**

With `message:<INDEX>` argument: Remove the message with the given index from the node.

**Example:**

```bash
> mnodectl node001 -d message:admin2
```

The admin2 message on node001 is removed.

### -m — Modify

**Format:**

```plaintext
<ATTR>=<VAL>
```

Where `<ATTR>` is one of the following:

- GEVENT
- GMETRIC
- MESSAGE
- OS
- POWER
- STATE
- VARIABLE

**Default:** ---

**Description:**

Modify the state or attribute of specified node(s)

**Example:**

```bash
> 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=Drain node1
> mnodectl -m variable=IP=10.10.10.100,Location=R1S2 node1
```
### -q — Query

**Format:** `{cat | diag | profile | wiki}`

**Default:** ---

**Description:** Query node categories or node profile information (see ENABLEPROFILING for nodes).

**Example:**

```bash
> 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%
```

```bash
> mnodectl -v -q profile
...
```

```bash
> mnodectl -q wiki <nodeName>
```

Query a node with the output displayed in a WIKI string. The node's name may be replaced with ALL.

### Parameters

#### GEVENT

**Format:** `<EVENT>:<MESSAGE>`

**Default:** ---

**Description:** Creates a generic event on the node to which Moab may respond. (see Enabling Generic Events)

**Example:**

```bash
mnodectl -m gevent=powerfail:'power has failed' node1
```

#### GMETRIC

**Format:** `<ATTR>:<VALUE>`

**Default:** ---
<table>
<thead>
<tr>
<th><strong>Description:</strong> Sets the value for a generic metric on the node. (see Enabling Generic Metrics)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example:</strong></td>
</tr>
</tbody>
</table>
| ```bash
mnodectl -m gmetric=temp:120 node1
``` |

| **MESSAGE** |
| **Format:** '<MESSAGE>' |
| **Default:** --- |
| **Description:** Sets a message to be displayed on the node. |
| **Example:** |
| ```bash
mnodectl -m message='powerfailure: power has failed' node1
``` |

| **NODEEXP** |
| **Format:** <STRING> Where <NODEEXP> is a node name, regex or "ALL" |
| **Default:** --- |
| **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> |
| **Default:** --- |
| **Description:** Operating System (see Resource Provisioning) |
| **Example:** |
| ```bash
mnodectl node1 -m OS=RHELAS30
``` |

| **POWER** |
| **Format:** {off|on} |
| **Default:** --- |
| **Description:** Set the power state of a node. Action will NOT be taken if the node is already in the specified state. |
| **Example:** |
| ```bash
> mnodectl node1 -m power=off
``` |

| **STATE** |
| **Format:** <NODE_STATE> Where <NODE_STATE> is one of the following:  
Busy,  
Down,  
Drain,  
Drained,  
Flush,  
```
### Default:
---

**Description:** Sets the state of the specified node(s).

⚠️ Not all resource managers support all of these states.

**Example:**
```
> mnodectl node1 -m state=Drain
```

---

### VARIABLE

**Format:** `<name>[=value],<name][=value]...`

**Default:** ---

**Description:** Set a list of variables for a node.

**Example:**
```
> mnodectl node1 -m variable=IP=10.10.10.100,Location=R1S2
```

---

### See Also

- [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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--about</td>
<td>Displays build environment and version information.</td>
</tr>
<tr>
<td>--loglevel</td>
<td>Sets the server loglevel to the specified value.</td>
</tr>
<tr>
<td>--version</td>
<td>Displays version information.</td>
</tr>
<tr>
<td>-c</td>
<td>Configuration file the server should use.</td>
</tr>
<tr>
<td>-C</td>
<td>Clears checkpoint files (.moab.ck, .moab.ck.1).</td>
</tr>
<tr>
<td>-d</td>
<td>Debug mode (does not background itself).</td>
</tr>
<tr>
<td>-e</td>
<td>Forces Moab to exit if there are any errors in the configuration file, or if it can't find these directories:</td>
</tr>
<tr>
<td></td>
<td>• statdir</td>
</tr>
<tr>
<td></td>
<td>• logdir</td>
</tr>
<tr>
<td></td>
<td>• spooldir</td>
</tr>
<tr>
<td></td>
<td>• toolsdir</td>
</tr>
<tr>
<td>-P</td>
<td>Starts Moab in a paused state for the duration specified.</td>
</tr>
<tr>
<td>-R</td>
<td>Causes Moab to automatically recycle every time the specified duration transpires.</td>
</tr>
<tr>
<td>-s</td>
<td>Starts Moab in the state that was most recently checkpointed.</td>
</tr>
<tr>
<td>-S</td>
<td>Suspends/stops scheduling at specified iteration (or at startup if no iteration is specified).</td>
</tr>
<tr>
<td>-v</td>
<td>Same as --version.</td>
</tr>
</tbody>
</table>
# mrmctl

**Moab Resource Manager Control**

## Synopsis

```
mrmctl -f [fobject] {rmName|am:{amid}}
mrmctl -l [rmid|am:{amid}]
mrmctl -m <attr>=<value> {rmid|am:{amid}} [-w <subobjtype>=<value>]
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

<table>
<thead>
<tr>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[fobject] where <strong>fobject</strong> is optional and one of <strong>messages</strong> or <strong>stats</strong>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>If no <strong>fobject</strong> 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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clears resource manager statistics. If <strong>messages</strong> is specified, then reported failures, performance data, and messages will be flushed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example:</th>
</tr>
</thead>
</table>
| ```
> mrmctl -f base
``` |

Moab will clear the statistics for RM *base*.

### -l – List

<table>
<thead>
<tr>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All RMs and AMs (when no RM/AM is specified)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>List Resource and Allocation Manager(s)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example:</th>
</tr>
</thead>
</table>
| ```
> mrmctl -l
``` |

Moab will list all resource and allocation managers.

### -m – Modify

<table>
<thead>
<tr>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All RMs and AMs (when no RM/AM is specified).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modify Resource and Allocation Manager(s).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example:</th>
</tr>
</thead>
</table>
| ```
> mrmctl -m state=disabled peer13
``` |
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
```

See Also
- **Moab Client Installation** - explains how to distribute this command to client nodes
- `mdiag -R`
- `mdiag -c`
mrsvctl
(Moab Reservation Control)

Synopsis
[-E] [-f features] [-F flags] [-g rsvgroup] [-h hostexp]
[-I (cancel|end|failure|start)] [-n name] [-0 owner]
[-p partition] [-P profile] [-R resources] [-S starttime]
[-t tasks] [-T trigger] [-V variable] [-x joblist]

mrsvctl -l {reservationid | -i index}

mrsvctl -C {-g standing_reservationid}

mrsvctl -m {reservationid | -i index} [-d duration] [-e endtime] [-h hostexp]

mrsvctl -q {reservationid | -i index}

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

<table>
<thead>
<tr>
<th>-a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: ACL</td>
</tr>
<tr>
<td>Format: <code>&lt;TYPE&gt;=&lt;VAL&gt;[,&lt;TYPE&gt;=&lt;VAL&gt;...]</code></td>
</tr>
<tr>
<td>Default: ---</td>
</tr>
<tr>
<td>Description: List of credentials with access to the reserved resources (See also: ACL Modifiers)</td>
</tr>
<tr>
<td>Example: &gt; 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</td>
</tr>
</tbody>
</table>
Moab will remove user john from the system.1 reservation.

Notes:

- There are three different assignment operators that can be used for modifying 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.
- 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.
- You can verify the ACL of a reservation using the "mdiag -r" command.

```
> mrsvctl -m -a USER==john system.1
Moab will remove user john from the system.1 reservation.
```

```
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.
```

```
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".
```

---

<table>
<thead>
<tr>
<th>-b</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name:</strong></td>
</tr>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
</tr>
</tbody>
</table>

Moab will associate the reserved nodes with the node category swmain.

---

<table>
<thead>
<tr>
<th>-c</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name:</strong></td>
</tr>
</tbody>
</table>

### Create a Reservation

**Format:** `<ARGUMENTS>`  
**Default:** ---  
**Description:** Creates a reservation.  
Note: 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.

**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.

### Clear Standing Reservation

**Name:** CLEAR  
**Format:** `<RSVID>` | `-g <SRSVID>`  
**Default:** ---  
**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**.

### Set Duration

**Name:** DURATION  
**Format:** `[[[DD:]HH:]MM:]SS`  
**Default:** INFINITY  
**Description:** Duration of the reservation (not needed if ENDTIME is specified).

**Example:**  
```
> mrsvctl -c -h node01 -d 5:00:00
```
Moab will create a reservation on node01 lasting 5 hours.

### Add Description

**Name:** DESCRIPTION  
**Format:** `<STRING>`  
**Default:** ---  
**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**

<table>
<thead>
<tr>
<th>Name</th>
<th>ENDTIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>[HH:[MM:[SS]]][_MO[/DD[/YY]]] or +[[[DD:]HH:]MM:]SS</td>
</tr>
<tr>
<td>Default</td>
<td>INFINITY</td>
</tr>
<tr>
<td>Description:</td>
<td>Absolute or relative time reservation will end (not required if Duration specified). ENDTIME also supports an epoch timestamp.</td>
</tr>
<tr>
<td>Example:</td>
<td>&gt; mrsvctl -c -h node01 -e +3:00:00</td>
</tr>
</tbody>
</table>

Moab will create a reservation on node01 ending in 3 hours.

**-E**

<table>
<thead>
<tr>
<th>Name</th>
<th>EXCLUSIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>N/A</td>
</tr>
<tr>
<td>Default</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>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.</td>
</tr>
<tr>
<td>Example:</td>
<td>&gt; mrsvctl -c -h node01 -E</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>FEATURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>&lt;STRING&gt;[:&lt;STRING&gt;]...</td>
</tr>
<tr>
<td>Default</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>List of node features which must be possessed by the reserved resources</td>
</tr>
<tr>
<td>Example:</td>
<td>&gt; mrsvctl -c -h node[0-9] -f fast</td>
</tr>
</tbody>
</table>

Moab will create a reservation on nodes matching the expression and which also have the feature fast.

**-F**

<table>
<thead>
<tr>
<th>Name</th>
<th>FLAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>&lt;flag&gt;[[,&lt;flag&gt;]...]</td>
</tr>
<tr>
<td>Default</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>Comma-delimited list of flags to set for the reservation (see Managing Reservations for flags).</td>
</tr>
</tbody>
</table>
### -g

<table>
<thead>
<tr>
<th>Name</th>
<th>RSVGROUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>&lt;STRING&gt;</td>
</tr>
<tr>
<td>Default</td>
<td>---</td>
</tr>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
<tr>
<td>Example</td>
<td>Moab will create a reservation on nodes matching the expression given and assign it to the reservation group staff.</td>
</tr>
</tbody>
</table>

#### Example:

```bash
mrsvctl -c -g staff -h 'node0[1-9]'  
```

### -h

<table>
<thead>
<tr>
<th>Name</th>
<th>HOSTLIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>&lt;STRING&gt;</td>
</tr>
<tr>
<td>Default</td>
<td>ALL</td>
</tr>
<tr>
<td>Description:</td>
<td>Host list (comma delimited), host regular expression, host range, or class mapping indicating the nodes which the reservation will allocate.</td>
</tr>
<tr>
<td>Example</td>
<td>Moab will create a reservation on nodes matching the expression given.</td>
</tr>
</tbody>
</table>

#### Example:

```bash
mrsvctl -c -h 'node0[1-9]'  
```

### -i

<table>
<thead>
<tr>
<th>Name</th>
<th>INDEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>&lt;STRING&gt;</td>
</tr>
<tr>
<td>Default</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>Use the reservation index instead of full reservation ID.</td>
</tr>
<tr>
<td>Example</td>
<td>Moab will modify the reservation with the index of 1 to start in 5 minutes.</td>
</tr>
</tbody>
</table>

#### Example:

```bash
mrsvctl -m -i 1 starttime=+5:00  
```
### SIGNAL

**Name:** SIGNAL  
**Format:** {cancel|end|failure|start}  
**Default:** ---  
**Description:** Send signals under the specified event conditions.  
**Example:**  
```bash  
> mrsvctl -m -I start starttime=+5:00  
```
Moab will send a signal when the start event occurs.

### LIST

**Name:** LIST  
**Format:** <RSV_ID> or ALL  
RSV_ID can be the name of a reservation or a regular expression.  
**Default:** ALL  
**Description:** List reservation(s).  
**Example:**  
```bash  
> mrsvctl -l system*  
```
Moab will list all of the reservations whose names start with `system`.

### MODIFY

**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>]  
- **label** label=<LABEL>  
- **modify** variable{+=key1=val1|-|=key_to_remove}  
- **reqtaskcount** reqtaskcount{+=|-|=}<TASKCOUNT>  
- **rsvgroup**  
- **starttime** starttime{+=|-|=}<RELTIME> or starttime=<ABSTIME>  
**Default:** ---  
**Description:** Modify aspects of a reservation.  
**Example:**  
```bash  
> 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.
Moab advances the starttime of system.1 five hours from its current starttime (without modifying the duration of the reservation).

```
> mrsvctl -m starttime+=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 endtime-=5:00:00 system.1
```

Moab moves the starttime of reservation system.1 ahead five hours.

```
> mrsvctl -m -s 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 -s -=5:00:00 system.1
```

Moab moves the starttime of reservation system.1 ahead five hours.

```
> mrsvctl -m +d +=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.

```
> mrsvctl -m variable+key1=val1 system.1
```

Moab adds the variable key1 with the value val1 to system.1.

```
> mrsvctl -m variable+=key1=val1 -m 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.

---

**Notes:**

- When the label is assigned for a reservation, the reservation can then be referenced by that label as well as by the reservation name. The reservation name cannot be modified.
- The starttime of a reservation can be modified by using `starttime` or `-s`. 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.
- The endtime of a reservation can be modified by using `endtime` or `-e`. 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.
- The duration can be changed by using `duration` or `-d`. 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). The + and - operators can be used on -s, and + can be used on -e as well.

- 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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mrsvctl -m -s 8:00:00 3/1/07 system.1</code></td>
<td>Moab moves system.1's starttime to 8:00 a.m., March 1.</td>
</tr>
<tr>
<td><code>mrsvctl -m -s 8:00:00 system.1</code></td>
<td>Moab moves system.1's starttime to 8:00 a.m., March 2.</td>
</tr>
<tr>
<td><code>mrsvctl -m -e 7:00:00 system.1</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>mrsvctl -m -e 7:00:00_3/2/07 system.1</code></td>
<td>Moab will return an error because the endtime cannot be before the starttime.</td>
</tr>
</tbody>
</table>

### -n

<table>
<thead>
<tr>
<th>Name:</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format:</td>
<td>&lt;STRING&gt;</td>
</tr>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>Name for new reservation.</td>
</tr>
</tbody>
</table>

- 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.

<table>
<thead>
<tr>
<th>Example:</th>
<th>Moab will create a reservation on node01 with the name John.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mrsvctl -c -h node01 -n John</code></td>
<td></td>
</tr>
</tbody>
</table>

### -o

<table>
<thead>
<tr>
<th>Name:</th>
<th>OWNER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format:</td>
<td>&lt;STRING&gt;</td>
</tr>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>Specifies the owner of a reservation. See Reservation Ownership for more information.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example:</th>
<th>Moab creates a reservation on node01 owned by user1.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mrsvctl -c -h node01 -o user1</code></td>
<td></td>
</tr>
</tbody>
</table>

### -p

<table>
<thead>
<tr>
<th>Command:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mrsvctl -c -h node01</code></td>
<td></td>
</tr>
</tbody>
</table>

- 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.

<table>
<thead>
<tr>
<th>Example:</th>
<th>Moab will create a reservation on node01 with the name John.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mrsvctl -c -h node01 -n John</code></td>
<td></td>
</tr>
</tbody>
</table>
###PARTITION

**Name:** PARTITION  
**Format:** <STRING>  
**Default:** ---  
**Description:** Only allocate resources from the specified partition  
**Example:**  

```
mrsvctl -c -p switchB -t 14
```

Moab will allocate 14 tasks from the switchB partition.

###PROFILE

**Name:** PROFILE  
**Format:** <STRING>  
**Default:** ---  
**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.

###QUERY

**Name:** QUERY  
**Format:** <RSV_ID> - The -q option accepts x: node regular expressions and r: node range expressions (asterisks (*) are supported wildcards as well), or ALL --flags=COMPLETED  
**Default:** ---  
**Description:** Get diagnostic information or list all completed reservations.  
**Example:**  

```
mrsvctl -q ALL --flags=COMPLETED
```

Moab will query completed reservations.  

```
mrsvctl -q system.1
```

Moab will query the reservation system.1.

###RELEASE

**Name:** RELEASE  
**Format:** <RSV_ID> - The -r option accepts x: node regular expressions and r: node range expressions (asterisks (*) are supported wildcards as well).  
**Default:** ---  
**Description:** Releases the specified reservation.  
**Example:**  

```
> mrsvctl -r system.1
```

Moab will release reservation system.1.
<table>
<thead>
<tr>
<th>Name</th>
<th>RESOURCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>&lt;tid&gt; or &lt;RES&gt;=&lt;VAL&gt;[:&lt;RES&gt;=&lt;VAL&gt;]... &lt;RES&gt; is one of the following:</td>
</tr>
<tr>
<td></td>
<td>PROCES, MEM, DISK, SWAP, GRES</td>
</tr>
<tr>
<td>Default</td>
<td>PROCS=-1</td>
</tr>
<tr>
<td>Description</td>
<td>Specifies the resources to be reserved per task. (-1 indicates all resources on node)</td>
</tr>
<tr>
<td>Example</td>
<td>&gt; mrsvct1 -c -R MEM=100;PROCS=2 -t 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>STARTTIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>[HH[:MM[:SS]]][_MO[/DD[/YY]]] or +[[[DD:]HH:]MM:]SS</td>
</tr>
<tr>
<td>Default</td>
<td>[NOW]</td>
</tr>
<tr>
<td>Description</td>
<td>Absolute or relative time reservation will start. STARTTIME also supports an epoch timestamp.</td>
</tr>
<tr>
<td>Example</td>
<td>&gt; mrsvct1 -c -t ALL -s 3:00:00_4/4/04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>SET ATTRIBUTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>&lt;ATTR&gt;=&lt;VALUE&gt; where &lt;ATTR&gt; is one of aaccount, agroup, aqos, auser, reqarch, reqmemory, reqnetwork, reqos</td>
</tr>
<tr>
<td>Description</td>
<td>Specifies the resources to be reserved per task. (-1 indicates all resources on node)</td>
</tr>
<tr>
<td>Example</td>
<td>&gt; mrsvct1 -c -h node01 -s +5:00</td>
</tr>
</tbody>
</table>
**rsvaccesslist** (comma delimited list of reservations or reservation groups which can be accessed by this reservation request)

Default: ---

Description: Specifies a reservation attribute will be used to create this reservation

Example: > mrvctl -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>]`
  **Default:** ---

  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:
  
  > mrvctl -c -t ALL
  Moab will create a reservation on all resources.

  > mrvctl -c -t 3
  Moab will create a reservation for three tasks.

  > mrvctl -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
  **Format:** `<STRING>`
  **Default:** N/A

  Description: Comma-delimited reservation trigger list following format described in the trigger format section of the reservation configuration overview. See Trigger Creation for more information.

  Example:
  
  > mrvctl -c -h node01 -T
  offset=200,etype=start,atype=exec,action=/opt/moab/tools/support.diag.p
  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>]]...`
  **Default:** N/A
Description: Semicolon-delimited list of variables that will be set when the reservation is created (see 19.5 Trigger Variables). 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>

Default: N/A

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

<table>
<thead>
<tr>
<th>RESERVATION ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format:</td>
</tr>
<tr>
<td>Default:</td>
</tr>
<tr>
<td>Description:</td>
</tr>
<tr>
<td>Example:</td>
</tr>
</tbody>
</table>

Specifies all reservations starting with 'system'.

Resource Allocation Details

When allocating resources, the following rules apply:

- When specifying tasks, a 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
dertime 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
dertime 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
```
The following increases the duration of a reservation by setting the duration to an absolute time:

```bash
$> 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
```

```bash
$> mrsvctl -m duration=48:00:00 system.1
duration for rsv 'system.1' changed
```

```bash
$> 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:

```bash
$> 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
```

```bash
$> mrsvctl -m endtime-=24:00:00 system.1
dertime for rsv 'system.1' changed
```

```bash
$> 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:

```bash
$ 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
```
Shortening a reservation by modifying the duration

The following modifies the duration of a reservation using the "-=" tag:

```bash
$> 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
```

The following modifies the duration of a reservation by setting the duration to an absolute time:

```bash
$> 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
```

Modifying the starttime of a reservation

The following increases the starttime of a reservation using the "+=" tag:

```bash
$> 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
```
The following decreases the starttime of a reservation using the ":=" tag:

```bash
$> 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 modifies the starttime of a reservation using an absolute time:

```bash
$> 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:

```bash
$> 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
```
The following modifies the start time of a reservation using an absolute time:

```bash
$> 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**

- **Example 1**: Basic Reservation
  - Reserve two nodes for use by users john and mary for a period of 8 hours starting in 24 hours.

  ```bash
  > mrsvctl -c -a USER= john,USER=mary -s +24:00:00 -d 8:00:00 -t 2
  reservation 'system.1' created
  ```

- **Example 2**: 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.

  ```bash
  % mrsvctl -c -s 8:00:00_06/20 -e 17:00:00_06/22 -h ALL
  reservation 'system.1' created
  ```

- **Example 3**: Explicit Task Description
  - Reserve one processor and 512 MB of memory on nodes node003 through node 006 for members of the group staff and jobs in the interactive class.

  ```bash
  > mrsvctl -c -R PROCS=1,MEM=512 -a GROUP=staff,CLASS=interactive -h 'node00[3-6]'  
  ```

- **Example 4**: Dynamic Reservation Modification
- **Example 5**: Adding a Reservation Trigger
- **Example 6**: Index-based Reservation Release
- **Example 7**: Reservation Modification
- **Example 8**: Allocating Reserved Resources
- **Example 9**: Modifying an Existing Reservation
**Example 4: Dynamic Reservation Modification**

Modify reservation john.1 to start in 2 hours, run for 2 hours, and include node02 in the hostlist.

```plaintext
> mrsvctl -m starttime=+2:00:00,duration=2:00:00,HostExp+=node02
Note:  hosts added to rsv system.3
```

**Example 5: Adding a Reservation Trigger**

Add a trigger to reservation system.1

```plaintext
> mrsvctl -m TRIGGER=X
Note:  trigger added to rsv system.1
```

**Example 6: Index-based Reservation Release**

Release reservation system.1 using its index.

```plaintext
> mrsvctl -r -i 1
reservation 'system.1' successfully released
```

**Example 7: Reservation Modification**

Remove user John's access to reservation system.1

```plaintext
> mrsvctl -m -a USER=John system.1 --flags=unset
successfully changed ACL for rsv system.1
```

**Example 8: Allocating Reserved Resources**

Allocate resources for group `dev` which are exclusive except for resources found within reservations myrinet.3 or john.6

```plaintext
> 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

```plaintext
> 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

```plaintext
> mrsvctl -c -E -d 2:00:00 -a group=dev -t 64 -S rsvaccesslist=system.3,netowrk
```
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 9: 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

**See Also**

- 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
(Moab Scheduler Control)

Synopsis

mschedctl -A '<MESSAGE>'
mschedctl -c message messagestring [-o type:val]
mschedctl -c trigger triggerid -o type:val
mschedctl -d trigger triggerid
mschedctl -d vpc:vpclid
mschedctl -d message:index
mschedctl -f {all|estimates|fairshare|usage}
mschedctl -k
mschedctl -l {config|message|trigger|trans|vpc|vpcprofile} [--flags=verbose] [--xml]
mschedctl -m config string [-e] [--flags=persistent]
mschedctl -m trigger triggerid attr=val[,attr=val...]
mschedctl -m vpc vpclid attr=val[,attr=val...]
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

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;STRING&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>Report the specified parameter modification to the event log and annotate it with the specified message.</td>
</tr>
<tr>
<td>Example:</td>
<td>&gt; mschedctl --flags=pers -A 'increase logging' -m 'LOGLEVEL 6'</td>
</tr>
</tbody>
</table>

Adjust the LOGLEVEL parameter and record an associated message.

-c — CREATE

<table>
<thead>
<tr>
<th>Format:</th>
<th>One of:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• message &lt;STRING&gt; [-o &lt;TYPE&gt;:&lt;VAL&gt;]</td>
</tr>
<tr>
<td></td>
<td>• trigger &lt;TRIGSPEC&gt; -o &lt;OBJECTTYPE&gt;:&lt;OBJECTID&gt;</td>
</tr>
<tr>
<td></td>
<td>• vpc [-a &lt;ATTR&gt;=&lt;VAL&gt;...]</td>
</tr>
</tbody>
</table>

where <ATTR> is one of account, duration, messages, profile, reqresources, resources, rsvprofile, starttime, user, or variables

<table>
<thead>
<tr>
<th>Default:</th>
<th>---</th>
</tr>
</thead>
</table>

Format

-ANOTATE

Format: <STRING>

Default: ---

Description: Report the specified parameter modification to the event log and annotate it with the specified message.

Example:

> mschedctl --flags=pers -A 'increase logging' -m 'LOGLEVEL 6'

Adjust the LOGLEVEL parameter and record an associated message.
Description: Create a message or trigger and attach it to the specified object, or create a Virtual Private Cluster (VPC). To create a trigger on a default object, use the Moab configuration file (moab.cfg) rather than the `mschedctl` command.

Example:

```bash
mschedctl -c message tell the admin to be nice
```
Create a message on the system table.

```bash
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.

```bash
mschedctl -c vpc -a resources=6,7,8 -a profile=packageA
```
Create a vpc using TID's 6, 7, and 8 and based on profile packageA

**VPC commands (such as mschedctl -c vpc) are only enabled on hosting center builds.**

Additional VPC attributes: `-a`

---

### -d — DESTROY

**Format:** One of:

- `trigger <TRIGID>`
- `message:<INDEX>`
- `vpc:<VPCID>[:{soft|hard}]`

**Default:** ---

**Description:** Delete a trigger, message, or VPC.

**Example:**

```bash
mschedctl -d trigger 3
```
Delete trigger 3.

```bash
mschedctl -d message:5
```
Delete message with index 5.

```bash
mschedctl -d vpc:vpc.5:soft
```
Delete vpc.5. Soft destruction causes Moab to launch the end triggers attached to the reservations before deleting the VPC. Hard destruction causes Moab to launch the cancel triggers.

```bash
mschedctl -d vpc:ALL
```
Delete all VPCs.
### -f — FLUSH

**Format:** `{all|estimates|fairshare|usage}`

**Default:** --

**Description:** Flush (clear out) specified statistics

**Example:**
```
mschedctl -f usage
Flush usage statistics.
```

### -k — KILL

**Format:** --

**Default:** --

**Description:** Stop scheduling and exit the scheduler

**Example:**
```
mschedctl -k
Kill the scheduler.
```

### -l — LIST

**Format:** `{'config' | 'gres' | 'message' | 'trans' | 'trigger' | 'vpc' | 'vpcprofile'}` [--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 resources, scheduler configuration, system messages, triggers, transactions, virtual private clusters or VPC profiles.

**Example:**
```
mschedctl -l config
List system parameters.

mschedctl -l gres
List all configured generic resources.

mschedctl -l trans 1
List transaction id 1.

mschedctl -l trigger
List triggers.

mschedctl -l vpc
List VPCs.

mschedctl -l vpc:vpc.1
List VPC vpc.1.
```
**-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`, `etyp`, `iscomplete`, `oid`, `otype`, `offset`, or `threshold`

- `vpc`: `<VPCID> <ATTR>=<VAL>`

  where `<ATTR>` is one of `variables`, or `<user>|<group>|<owner>|<qos>|<account>`

  Changing the VPC user does not change the VPC owner, and vice versa. Additionally, the credential you want to change to must already exist in order to make the change. In other words, you cannot set `user` to a user that Moab doesn't know about.

**Default:** ---

**Description:** Modify a system parameter, trigger, or VPC.

**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.

> mschedctl -m vpc:packageA.1 variables=blue=dog
Change aspects of vpc packageA.1.

> mschedctl -m vpc:vpc.10 user=craig
vpc USER set to craig
```
Changes the user of vpc.10 to 'craig'

- **-p — PAUSE**

  **Format:**
  ```
  ---
  ```

  **Default:**
  ```
  ---
  ```

  **Description:** Disable scheduling but allow the scheduler to update its cluster and workload state information.

  **Example:**
  ```
  > mschedctl -p
  ```

- **-q — QUERY**

  **Format:**
  ```
  Comma delimited list of one or more of the following:
  acl, ALL, am, cjob, class, cluster, cp, depend, event, fs, fsc, green, gres, group, hybrid, image, job, limits, node, nodefeature, pactions, par, priority, qos, queue, rack, range, req, rm, rsv, sched, sim, srsrv, stats, sys, task, tasksperjob, tjob, tnode, trans, trig, tx, user, vm, vpc
  ```

  **Default:**
  ```
  ---
  ```

  **Description:** Query scheduler attributes

  **Example:**
  ```
  > mschedctl -q pactions --format=xml
  ```

- **-R — RECYCLE**

  **Format:**
  ```
  ---
  ```

  **Default:**
  ```
  ---
  ```

  **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:**
  ```
  <INTEGER>
  ```

  **Default:**
  ```
  0
  ```

  **Description:** Resume scheduling at the specified time (or immediately if none is specified).

  **Example:**
  ```
  > mschedctl -r
  ```
Resume scheduling immediately.

**-s — STOP**

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;INTEGER&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>Description:</td>
<td>Suspend/stop scheduling at specified iteration (or at the end of the current iteration if none is specified). If the letter 'I' follows <code>&lt;ITERATION&gt;</code>, Moab will not process client requests until this iteration is reached.</td>
</tr>
<tr>
<td>Example:</td>
<td><code>&gt; mschedctl -s 100I</code></td>
</tr>
<tr>
<td></td>
<td>Stop scheduling at iteration 100 and ignore all client requests until then.</td>
</tr>
</tbody>
</table>

**-S — STEP**

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;INTEGER&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>Description:</td>
<td>Step the specified number of iterations (or to the next iteration if none is specified) and suspend scheduling. If the letter 'I' follows <code>&lt;ITERATION&gt;</code>, Moab will not process client requests until this iteration is reached.</td>
</tr>
<tr>
<td>Example:</td>
<td><code>&gt; mschedctl -S</code></td>
</tr>
<tr>
<td></td>
<td>Step to the next iteration and stop scheduling.</td>
</tr>
</tbody>
</table>

**Example 1**

Shutting down the Scheduler:

```
> mschedctl -k
scheduler will be shutdown immediately
```

**Example 2**

Creating a virtual private cluster:

```
> mschedctl -c vpc -a resources=14332 -a variables=os=rhel3
vpc.98
```

**See Also**

- Moab Client Installation - explains how to distribute this command to client nodes
**mshow**

*(Moab Show)*

**Synopsis**

mshow [-a] [-q jobqueue]

**Overview**

The mshow command displays various diagnostic messages about the system and job queues.

**Arguments**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>AVAILABLE RESOURCES</td>
</tr>
<tr>
<td>-q</td>
<td>JOB QUEUE</td>
</tr>
</tbody>
</table>

**Format**

**AVAILABLE RESOURCES**

| Format: | Can be combined with --flags=[tid|verbose|future] --format=xml and/or -w |
|---------|------------------------------------------------------------------|
| Default: | --- |
| 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**

<table>
<thead>
<tr>
<th>Format:</th>
<th>---</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>---</td>
</tr>
<tr>
<td>Description:</td>
<td>Show the job queue.</td>
</tr>
<tr>
<td>Example:</td>
<td></td>
</tr>
</tbody>
</table>

> mshow -q

[information about all queues]

**See Also**

- Moab Client Installation - explains how to distribute this command to client nodes
- mshow -a command to show available resources
mshow -a

(Moab Show Available Resources)

Synopsis


Overview

The mshow -a command allows for querying of available system resources.

Arguments

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-i]</td>
<td>INTERSECTION</td>
</tr>
<tr>
<td>[-o]</td>
<td>NO AGGREGATE</td>
</tr>
<tr>
<td>[-p]</td>
<td>PROFILE</td>
</tr>
<tr>
<td>[-T]</td>
<td>TIMELOCK</td>
</tr>
<tr>
<td>[-w]</td>
<td>WHERE</td>
</tr>
<tr>
<td>[-x]</td>
<td>EXCLUSIVE</td>
</tr>
</tbody>
</table>

Table 1: Argument Format

<table>
<thead>
<tr>
<th>--flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: Flags</td>
</tr>
<tr>
<td>Format: --flags=[ future</td>
</tr>
<tr>
<td>Description: future will return resources available immediately and available in the future. policy 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. timeflex allows the reservation to move in time (but not space) in order to start at an earlier time, if able. verbose will return diagnostic information.</td>
</tr>
<tr>
<td>Example: &gt; mshow -a -w user=john --flags=tid --xml Show resources available to john in XML format with a transaction ID.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>--xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: XML</td>
</tr>
<tr>
<td>Format: --xml</td>
</tr>
<tr>
<td>Description: Report results in XML format.</td>
</tr>
<tr>
<td>Example: &gt; mshow -a -w user=john --flags=tid --xml Show resources available to john in XML format with a transaction ID.</td>
</tr>
</tbody>
</table>
### INTERSECTION

- **Name:** INTERSECTION
- **Format:** --
- **Description:** Specifies that an intersection should be performed during an mshow -a command with multiple requirements.
- **Example:**

### NO AGGREGATE

- **Name:** NO AGGREGATE
- **Format:** --
- **Description:** Specifies that the results of the command mshow -a with multiple requirements should not be aggregated together.
- **Example:**

### PROFILE

- **Name:** PROFILE
- **Format:** <VPCPROFILEID>
- **Description:** Specifies which virtual private cluster profile should be used to adjust the explicit constraints of the request.
- **Example:**

### TIMELOCK

- **Name:** TIMELOCK
- **Format:** --
- **Description:** Specifies that the multiple requirements of an mshow -a command should be timelocked.
- **Example:**

### WHERE

- **Name:** WHERE
- **Format:** <ATTR>=<VAL>,<ATTR>=<VAL>
- **Description:** Add a "Where" clause to the current command (currently supports up to six co-allocation clauses).
- **Example:**

```bash
> mshow -a -w minprocs=2,duration=1:00:00 -w nodemem=512,duration=1:00:00
```
**Name:** EXCLUSIVE

**Format:** ---

**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 2: Request Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>the account credential of the requestor</td>
</tr>
<tr>
<td>acl</td>
<td>ACL to attach to the reservation associated with the VPC</td>
</tr>
</tbody>
</table>

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) |

![Warning]

The number of tasks requested in each Where request must be equal whether this taskcount is specified via **minprocs**, **mintasks**, or **gres**.

<p>| count    | the number of profiles to apply to the resource request |
| displaymode | Possible value is <strong>future</strong>. (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 |
| gres     | select only nodes which possess the specified generic resource |
| group    | the group credential of the requestor |
| hostlist | select only the specified resources |
| 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. |</p>
<table>
<thead>
<tr>
<th>label</th>
<th>associate the specified label with all results matching this request</th>
</tr>
</thead>
<tbody>
<tr>
<td>minnodes</td>
<td>return only results with at least the number of nodes specified. If used with TID's, return only solutions with exactly minnodes nodes available</td>
</tr>
<tr>
<td>minprocs</td>
<td>return only results with at least the number of processors specified. If used with TID's, return only solutions with exactly minprocs processors available</td>
</tr>
<tr>
<td>mintasks</td>
<td>FORMAT: &lt;TASKCOUNT&gt;[[@&lt;RESTYPE&gt;:&lt;COUNT&gt;[+&lt;RESTYPE&gt;:&lt;COUNT&gt;]]... where &lt;RESTYPE&gt; 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</td>
</tr>
<tr>
<td>nodedisk</td>
<td>select only nodes with at least nodedisk MB of local disk configured</td>
</tr>
<tr>
<td>nodefeature</td>
<td>select only nodes with all specified features present using format &lt;feature&gt;[:&lt;feature&gt;][;...</td>
</tr>
<tr>
<td>nodemem</td>
<td>select only nodes with at least nodemem MB of memory configured</td>
</tr>
<tr>
<td>offset</td>
<td>select only resources which can be co-allocated with the specified time offset where offset is specified in the format [[DD:]HH:]MM:]SS</td>
</tr>
<tr>
<td>os</td>
<td>select only nodes with have, or can be provisioned to have, the specified operating system</td>
</tr>
<tr>
<td>partition</td>
<td>the partition in which the resources must be located</td>
</tr>
<tr>
<td>policylevel</td>
<td>enable policy enforcement at the specified policy constraint level</td>
</tr>
<tr>
<td>qos</td>
<td>the qos credential of the requestor</td>
</tr>
<tr>
<td>rsvprofile</td>
<td>use the specified profile if committing a resulting transaction id directly to a reservation</td>
</tr>
<tr>
<td>starttime</td>
<td>constrain the timeframe for the returned results by specifying one or more ranges using the format &lt;STIME&gt;[-&lt;ENDTIME&gt;][:&lt;STIME&gt;[-&lt;ENDTIME&gt;]] 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 &lt;EPOCHTIME&gt;]. 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.</td>
</tr>
<tr>
<td>taskmem</td>
<td>require taskmem MB of memory per task located</td>
</tr>
<tr>
<td>tpn</td>
<td>require exactly tpn tasks per node on all discovered resources</td>
</tr>
<tr>
<td>user</td>
<td>the user credential of the requestor</td>
</tr>
<tr>
<td>var</td>
<td>use associated variables in generating per transaction charging quotes</td>
</tr>
<tr>
<td>variables</td>
<td>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 &lt;tid&gt;.</td>
</tr>
</tbody>
</table>

**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) or vpc (using mschedctl -c vpc -R). This allows system administrators to grab and reserve available resources for whatever reason, without conflicting with jobs or reservations that may 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.

**Example 1: Basic Compute Node Query and Reservation**

```bash
> mshow -a -w duration=10:00:00, minprocs=1, os=AIX53, jobfeature=shared --flags=tid, future

<table>
<thead>
<tr>
<th>Partition</th>
<th>Tasks</th>
<th>Nodes</th>
<th>Duration</th>
<th>StartOffset</th>
<th>StartDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>1</td>
<td>1</td>
<td>10:00:00</td>
<td>00:00:00</td>
<td>13:28:09_04/27</td>
</tr>
<tr>
<td>TID=4</td>
<td>ReqID=0</td>
<td>1</td>
<td>10:00:00</td>
<td>10:00:00</td>
<td>17:14:48_04/28</td>
</tr>
<tr>
<td>TID=5</td>
<td>ReqID=0</td>
<td>1</td>
<td>10:00:00</td>
<td>20:00:00</td>
<td>21:01:27_04/29</td>
</tr>
</tbody>
</table>

> mrsvctl -c -R 4

Note: reservation system.2 created
```

**Example 2: 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.

```bash
> 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

<table>
<thead>
<tr>
<th>Partition</th>
<th>Tasks</th>
<th>Nodes</th>
<th>Duration</th>
<th>StartOffset</th>
<th>StartDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>1</td>
<td>1</td>
<td>10:00:00</td>
<td>00:00:00</td>
<td>13:28:09_04/27</td>
</tr>
<tr>
<td>TID=4</td>
<td>ReqID=0</td>
<td>1</td>
<td>10:00:00</td>
<td>10:00:00</td>
<td>17:14:48_04/28</td>
</tr>
<tr>
<td>TID=5</td>
<td>ReqID=0</td>
<td>1</td>
<td>10:00:00</td>
<td>20:00:00</td>
<td>21:01:27_04/29</td>
</tr>
</tbody>
</table>

> mrsvctl -c -R 4

Note: reservation system.2 created
Note: reservation system.3 created
Note: reservation system.4 created
```

**Example 3: Request for Generic Resources**

Query for a generic resource on a specific host (no processors, only a generic resource).
Example 4: 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 `mshow` requests 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 -i -x -o -w gres=dvd,duration=10:00,hostlist=node03 --
flags=tid,future

Partition     Tasks  Nodes   StartOffset      Duration
StartDate     -----  -----  ------------  ------------  ---------
----------
----------
-----
11:33:25_07/27  TID=16  ReqID=0
11:43:25_07/27  TID=17  ReqID=0
11:53:25_07/27  TID=18  ReqID=0

> mrsvctl -c -R 16
Note:  reservation system.6 created

> mdiag -r system.6
Diagnosing Reservations

<table>
<thead>
<tr>
<th>RsvID</th>
<th>Duration</th>
<th>Node</th>
<th>Task</th>
<th>Proc</th>
<th>StartTime</th>
<th>EndTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>system.6</td>
<td>-00:01:02</td>
<td>loc</td>
<td>-</td>
<td>0</td>
<td>-00:01:02</td>
<td>00:08:35</td>
</tr>
</tbody>
</table>

Flags: ISCLOSED
ACL:   RSV==system.6=
CL:    RSV==system.6

> mshow -a -w
duration=10:00:00,minprocs=1,os=AIX53,jobflags=ADVRES,jobfeature=shared --flags=tid

Partition     Tasks  Nodes      Duration   StartOffset
StartDate     -----  -----  ------------  ------------  ---------
----------
----------
-----
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     -----  -----  ------------  ------------  ---------
----------
----------
-----
13:20:23_04/27  TID=1  ReqID=0

```

```
Example 5: 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.

```bash
> mshow -a -w class=grid, minprocs=8, duration=20:00 --flags=future, verbose
<Data>
  <Object>cluster</Object>
  <job User="john" time="1162407604"></job>
  <par Name="template">
    <range duration="Duration" nodecount="Nodes" proccount="Procs" starttime="Startime"></range>
  </par>
  <par Name="ALL" feasibleNodeCount="131" feasibleTaskCount="163">
  </par>
</Data>
```

This command reports the original query, and the timeframe, resource size, and hostlist associated with each possible time slot.

Example 6: Create a Virtual Private Cluster

Request an exclusive five-node virtual private cluster using the Apache profile.

```bash
> mshow -a -i -x -o --flags=summary,tid,future,timeflex -p apache -w
> duration=3000,minnodes=5
```

```
---
<table>
<thead>
<tr>
<th>Partition</th>
<th>Tasks</th>
<th>Nodes</th>
<th>StartOffset</th>
<th>Duration</th>
<th>StartDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>5</td>
<td>5</td>
<td>00:00:00</td>
<td>00:50:00</td>
<td>09:59:07_07/03</td>
</tr>
<tr>
<td>TID=265</td>
<td>5</td>
<td>5</td>
<td>00:50:00</td>
<td>00:50:00</td>
<td>10:49:07_07/03</td>
</tr>
<tr>
<td>ALL</td>
<td>5</td>
<td>5</td>
<td>1:40:00</td>
<td>00:50:00</td>
<td>11:39:07_07/03</td>
</tr>
<tr>
<td>TID=267</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

> mschedctl -c vpc -a resources=265
See Also

- [Moab Client Installation](#) - explains how to distribute this command to client nodes
- [mshow in a hosting environment](#)
mshow -a
(Moab Show Available Resources)

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 ...
```

<table>
<thead>
<tr>
<th>Partition</th>
<th>Procs</th>
<th>Nodes</th>
<th>StartOffset</th>
<th>Duration</th>
<th>StartDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>16</td>
<td>8</td>
<td>00:00:00</td>
<td>1:00:00</td>
<td>13:00:18_08/25</td>
</tr>
<tr>
<td>ReqID=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>2</td>
<td>1</td>
<td>00:00:00</td>
<td>1:00:00</td>
<td>13:00:18_08/25</td>
</tr>
<tr>
<td>ReqID=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th>Partition</th>
<th>Procs</th>
<th>Nodes</th>
<th>StartOffset</th>
<th>Duration</th>
<th>StartDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>16</td>
<td>16</td>
<td>00:00:00</td>
<td>2:00:00</td>
<td>13:00:18_08/25</td>
</tr>
<tr>
<td>TID=26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReqID=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:
Where \(A_1\) is the hostlist, \(A_2\) is the duration, \(A_3\) is the starttime, \(A_4\) are any flags, and \(A_5\) 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.

```plaintext
-----
# moab.cfg
-----
RSVPROFILE[test1]
TRIGGER=Sets=$Var1.$Var2.$Var3.!Net,EType=start,AType=exec,Action=/tmp/

RSVPROFILE[test1]
TRIGGER=Requires=$Var1.$Var2.$Var3,Sets=$Var4.$Var5,EType=start,AType=e

RSVPROFILE[test1]

RSVPROFILE[test1] TRIGGER=Requires=NOOSini,AType=cancel,EType=start
RSVPROFILE[test1]
TRIGGER=EType=start,Requires=OSinit,AType=exec,Action=/tmp/host/trigger...
-----
```

To create a reservation with this profile the `mrsvctl -c -P` command is used:

```plaintext
> 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.

```plaintext
> 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 `mdiaq -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.

**See Also**
- mshow
msub
(Moab Job Submit)

Synopsis

msub [-a datetime][-A account][-c interval][-C directive_prefix][-d path]
[-e path][-E][-h][-I][-j join][-k keep][-l resourcelist][-m mailoptions][-M user_list][-N name][-p priority]
[-q destination][-r][-S pathlist][-u userlist][-t jobarrays][-v variablelist][-W additionalattributes][-x][-z]

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.

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
Default: ---
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>
Default: ---
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>]

**Default:** ---

**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:**

```bash
> 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)

**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:**

```bash
> 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:**

```bash
> msub -d /home/test/job12 cmd.pbs
```

The job will begin execution in the /home/test/job12 directory.

---

### -e

**Name:** Error Path

**Description:**

```bash
> msub -c c=12 cmd.pbs
```

The job will be checkpointed every 12 minutes.
**-E**

**Name:** Environment Variables

**Format:** N/A

**Default:** ---

**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_JOBID**: Job ID. If submitted from the grid, grid jobid.
- **MOAB_JOBNAME**: Job name.
- **MOAB_MACHINE**: Name of the machine (ie. Destination RM) that the job is running on.
- **MOAB_NODECOUNT**: Number of nodes allocated to job.
- **MOAB_NODENAME**: 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 NODENAME). With TORQUE/PBS, the variables are added just before the job is started.

This feature only works with SLURM and TORQUE/PBS.

**Example:**

```bash
> msub -E mySim.cmd
```

The job *mySim* will be submitted with extra environment variables.

**-h**

**Name:** Hold

**Format:** N/A

**Default:** ---

**Description:** Specifies that a user hold be applied to the job at submission time.

**Example:**

```bash
> msub -e test12/stderr.txt
> msub -E mySim.cmd
```
The job will be submitted with a user hold on it.

**-I**

**Name:** Interactive  
**Format:** N/A  
**Default:** ---  
**Description:** Declares the job is to be run interactively.  
**Example:**  
```bash  
> msub -I job117.sh  
```

The job will be submitted in interactive mode.

**-j**

**Name:** Join  
**Format:** [oe|n]  
**Default:** n (not merged)  
**Description:** Declares if the standard error stream of the job will be merged with the standard output stream of the job. If "oe" is specified, the error and output streams will be merged into the output stream.  
**Example:**  
```bash  
> 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). It is only for PBS resource managers.  
**Example:**  
```bash  
> 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  
**Default:** ---  
**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

```bash  
> msub -h cmd.ll  
> msub -I job117.sh  
> msub -j oe cmd.sh  
> msub -k oe myjob.sh  
```
frequent queries. Running ten jobs with the -K option creates an additional fifty queries per minute for the scheduler.

**Example:**

```bash
> msub -K newjob.sh
3
Job 3 completed*
```

*Only shows up after job completion.

---

### -l

**Name:** Resource List  
**Format:** `<STRING>` (either standard PBS/TORQUE options or resource manager extensions)  
**Default:** ---  
**Description:** Defines the resources that are required by the job and establishes a limit to the amount of resource that can be consumed. Either resources native to the resource manager (see PBS/TORQUE resources) or scheduler resource manager extensions may be specified. Note that resource lists are dependent on the resource manager in use.

**Example:**

```bash
> 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.

---

### -m

**Name:** Mail Options  
**Format:** `<STRING>` (either `n` or one or more of the characters `a`, `b`, and `e`)  
**Default:** ---  
**Description:** Defines the set of conditions (abort, begin, end) when the server will send a mail message about the job to the user.

**Example:**

```bash
> 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  
**Description:** Specifies the list of users to whom mail is sent by the execution server.

**Example:**

```bash
> msub -M jon@node01,bill@node01,jill@node02 cmd.sh
```
Mail will be sent to the specified users if the job is aborted.

<table>
<thead>
<tr>
<th>Option</th>
<th>Name</th>
<th>Format</th>
<th>Default</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-N</td>
<td>Name</td>
<td>&lt;STRING&gt;</td>
<td>STDIN or name of job script</td>
<td>Specifies the user-specified job name attribute.</td>
<td><code>&gt; msub -N chemjob3 cmd.sh</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Job will be associated with the name chemjob3.</td>
<td></td>
</tr>
<tr>
<td>-o</td>
<td>Output Path</td>
<td>[&lt;hostname&gt;:]&lt;path&gt;</td>
<td>$SUBMISSIONDIR/$JOBNAME.o$JOBID</td>
<td>Defines the path to be used for the standard output stream of the batch job.</td>
<td><code>&gt; msub -o test12/stdout.txt</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The STDOUT stream of the job will be placed in the relative (to execution) directory specified.</td>
<td></td>
</tr>
<tr>
<td>-p</td>
<td>Priority</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Defines the priority of the job.</td>
<td><code>&gt; msub -p 25 cmd.sh</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(between -1024 and 0)</td>
<td></td>
<td>To enable priority range from -1024 to +1023, see ENABLEPOSUSERPRIORITY.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The job will have a user priority of 25.</td>
<td></td>
</tr>
<tr>
<td>-q</td>
<td>Destination Queue (Class)</td>
<td>[&lt;queue&gt;][@&lt;server&gt;]</td>
<td>[DEFAULT]</td>
<td>Defines the destination of the job.</td>
<td><code>&gt; msub -q priority cmd.sh</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The job will be submitted to the priority queue.</td>
<td></td>
</tr>
<tr>
<td>-r</td>
<td>Rerunable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 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 Path
**Format:** <path>[@<host>],<path>[@<host>],...
**Default:** $SHELL
**Description:** Declares the shell that interprets the job script.

**Example:**
```
> msub -S /bin/bash
```
The job script will be interpreted by the /bin/bash shell.

### -t

**Name:** Job Arrays
**Format:** <name>.[<indexlist>]%<limit>
**Default:** --
**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.

**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
**Format:** <string>,<string>,...
**Default:** --
**Description:** Expands the list the environment variables that are exported to the job (taken from the msub command environment).
### -V
**Name:** All Variables  
**Format:** N/A  
**Default:** N/A  
**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>  
**Default:**  
**Description:** Allows the for the 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".

### -x
**Name:** ---  
**Format:** <script> or <command>  
**Default:**  
**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  
**Format:** N/A  
**Default:** N/A
Description: The job's identifier will not be printed to stdout upon submission.

Example:  

```bash
> msub -z cmd.sh
```
No job identifier will be printout the stdout upon successful submission.

## Job Script

The `msub` command supports job scripts written in any one of the following languages:

<table>
<thead>
<tr>
<th>Language</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBS/TORQUE Job Submission Language</td>
<td>---</td>
</tr>
<tr>
<td>LoadLeveler Job Submission Language</td>
<td>Use the INSTANTSTAGE parameter as only a subset of the command file keywords are interpreted by Moab.</td>
</tr>
<tr>
<td>SSS XML Job Object Specification</td>
<td>---</td>
</tr>
<tr>
<td>LSF Job Submission Language</td>
<td>enabled in Moab 4.2.4 and higher</td>
</tr>
</tbody>
</table>

### /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`:

```bash
-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.
Example 1

```
> msub -l nodes=3:ppn=2,walltime=1:00:00,pmem=100kb script2.pbs.cmd
4364.orion
```

Example 2

Example 2 is the XML-formatted version of Example 1. 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>
```

See Also

- [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 using msub to submit a job by specifying a job script, msub processes that script and then sends an XML representation of the job to the Moab scheduler. It is possible to change the job XML before it is sent to Moab via an msub submission filter.

The filter gives administrators the ability to customize the submission process. Customization may be helpful if jobs should have certain defaults assigned to them, if an administrator wants to keep detailed submission statistics, or if an administrator wants to change job requests based on custom needs.

The submit filter, which must be written by an administrator, is a simple executable or script that receives XML via its standard input and then returns the modified XML in its standard output. To see the schema for job submission XML, please refer to Submitting Jobs via msub in XML.

Sample Submit Filter Script

```perl
#!/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;

SUBMITFILTER /home/submitfilter/filter.pl
```

The script is executed by the user running msub.

To configure msub to use the submit filter, each submission host must have access to the submit filter script. Also, you must add a SUBMITFILTER parameter to the Moab configuration file (moab.cfg) on each submission host. The following exemplifies 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, which causes msub to print verbose log messages to the terminal.
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:

```xml
<job> **job attribute children** </job>
```

An example of a simple job element with all the required children for a job submission is as follows:

```xml
<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:

```xml
<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: 

```
```

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:

```xml
<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:

```xml
<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:

```xml
<Hold>User</Hold>
```

InitialWorkingDirectory: Specifies in which directory the job should begin executing. This is equivalent to the -d flag in the msub command.

```xml
<InitialWorkingDirectory>/home/user/directory</InitialWorkingDirectory>
```

Interactive: Specifies that the job is to be interactive. This is the equivalent of the -I flag in msub.

```xml
<Interactive>TRUE</Interactive>
```

JobName: Specifies the user-specified job name attribute. This is the equivalent to the -N flag in msub.

```xml
<JobName>Job</JobName>
```

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 -n and -N options in msub.

```xml
<NotificationList URI=user1:user2>JobFail,JobStart,JobEnd</NotificationList>
```

In this example, the command msub -m abe -N 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:

```xml
<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.

```xml
<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:

```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 19.0 Triggers 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:

```xml
<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;Action="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 host=list` msub directive.

```xml
<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`:

```xml
<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:

```xml
<Requested> <Processors>6</Processors> <TPN>2</TPN> </Requested>
```

**WallclockDuration**: The requested wall clock duration of the job. This attribute is specified in the Requested element.

```xml
<Requested> <WallclockDuration>3600</WallclockDuration> </Requested>
```

### See Also

- Applying the msub Submit Filter
- SUBMITFILTER parameter
**mvmctl**

*(Moab Virtual Machine Control)*

**Synopsis**

```
mvmctl -c [options] <vmid>
mvmctl -d <vmid>
mvmctl -m [options] <vmid>
mvmctl -M dsthost=<newhost> <vmid>
mvmctl -q <vmid> [--xml]
```

**Overview**

*mvmctl* controls the creation, modification, querying, migration, and destruction of virtual machines (VMs).

**Format**

<table>
<thead>
<tr>
<th>Name</th>
<th>Create</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format:</td>
<td><code>[&lt;options&gt;] [&lt;vmid&gt;]</code></td>
</tr>
<tr>
<td>Description:</td>
<td>Creates a VM.</td>
</tr>
</tbody>
</table>

Where the `<attr>` attribute is one of the following:

- **image** - The VM type/image to use for the new VM.
- **hypervisor** - The node on which the new VM resides.
- **disk** - Disk space for the VM OS.
- **mem** - Amount of memory to use.
- **procs** - Number of processors for the VM.
- **sovereign** - Boolean.
- **template** - Job template for the VM.
- **variable** - User-defined VM variables (of the form `<varname>=<value>[+<varname>=<value>]` — must be a name/value pair).
- **storage** - Storage request (not OS disk storage). Only viable at create time; cannot be modified. Storage is a percent sign-delimited list of storage requests with the following form:

```
<type>:<size>[@<mountPoint>]
```

- **ttl** - Time-to-live (walltime) of the VM. Can be in seconds or a time string (125:00:00:00 for 125 days, and so forth). Only viable at create time; cannot be modified.
- **trigger** - A trigger to attach to a job. To implement multiple triggers, input `mvmctl -c trigger=x&y,trigger=a&b`. The syntax is like other command line trigger definitions. Note that some escaping may be required on the command line.

Note that the `<vmid>` must be unique; Moab returns an error if the name is already taken.

**Example:**

```
> mvmctl -c image=stateful,hypervisor=node03,mem=512,procs=2 myNewVM
```

```
> mvmctl -c image=rhel51,hypervisor=n4,storage=gold:3%silver:5@/home/jason/silver%g
```
### -d

**Name:** Destroy  
**Format:** `<vmid>`  
**Default:** ---  
**Description:** Destroys the specified VM.  
**Example:**

```bash
> mvmctl -d oldVM
```

### -m

**Name:** Modify  
**Format:** `[<options>] <vmid>`  
```
The `<options>` parameter is a comma-separated list of `<attr>=<value>` pairs.
```
**Default:**  
**Description:** Modifies the VM.  
**Example:**

```bash
> 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 trigger=etype=start\&atype=exec\&action='trig.py $OID $HOSTLIST' myNewVM

The modify variable uses the same syntax as Create.

> mvmctl -m variable=user:bob+purpose:myVM myNewVM
```

**Notes:**  
- The variable option is a set-only operation. Previous variables will be over-written.
### M

**Name:** Migrate  

**Format:**  

dsthost=<newhost> <vmid>  

**Default:**  

**Description:** Migrate the given VM to the destination host.  

**Example:**  

```
> mvmctl -M dsthost=node05 myNewVM
```

### q

**Name:** Query  

**Format:**  

<vmid> [--xml]  

**Default:**  

**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.  

**Example:**  

```
< mvmctl -q myNewVM

< mvmctl -q ALL --xml
```
resetstats
(Reset Stats)

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.

Example 1

```bash
> resetstats
```

See Also

- Moab Client Installation - explains how to distribute this command to client nodes
**showbf**  
*(Show Available Resources)*

**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]
```

**Overview**

Shows what resources are available for immediate use.

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNT</td>
<td>Account name.</td>
</tr>
<tr>
<td>CLASS</td>
<td>Class/queue required.</td>
</tr>
<tr>
<td>DURATION</td>
<td>Time duration specified as the number of seconds or in [DD:]HH:MM:SS notation.</td>
</tr>
<tr>
<td>FEATURELIST</td>
<td>Colon separated list of node features required.</td>
</tr>
<tr>
<td>GROUP</td>
<td>Specify particular group.</td>
</tr>
<tr>
<td>MEMCMP</td>
<td>Memory comparison used with the -m flag. Valid signs are &gt;, &gt;=, ==, &lt;=, and &lt;.</td>
</tr>
<tr>
<td>MEMORY</td>
<td>Specifies the amount of required real memory configured on the node, (in MB), used with the -m flag.</td>
</tr>
<tr>
<td>NODECOUNT</td>
<td>Specify number of nodes for inquiry with -n flag.</td>
</tr>
<tr>
<td>PARTITION</td>
<td>Specify partition to check with -p flag.</td>
</tr>
<tr>
<td>QOS</td>
<td>Specify QOS to check with -q flag.</td>
</tr>
<tr>
<td>USER</td>
<td>Specify particular user to check with -u flag.</td>
</tr>
</tbody>
</table>

**Flags**
<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A</td>
<td>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.</td>
</tr>
<tr>
<td>-a</td>
<td>Show resource availability information only for specified account.</td>
</tr>
<tr>
<td>-d</td>
<td>Show resource availability information for specified duration.</td>
</tr>
<tr>
<td>-D</td>
<td>Display current and future resource availability notation.</td>
</tr>
<tr>
<td>-g</td>
<td>Show resource availability information only for specified group.</td>
</tr>
<tr>
<td>-h</td>
<td>Help for this command.</td>
</tr>
<tr>
<td>-L</td>
<td>Enforce Hard limits when showing available resources.</td>
</tr>
<tr>
<td>-m</td>
<td>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.</td>
</tr>
<tr>
<td>-n</td>
<td>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.</td>
</tr>
<tr>
<td>-p</td>
<td>Show resource availability information for the specified partition.</td>
</tr>
<tr>
<td>-q</td>
<td>Show information for the specified QOS.</td>
</tr>
<tr>
<td>-u</td>
<td>Show resource availability information only for specified user.</td>
</tr>
</tbody>
</table>

**Example 1**

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
```

<table>
<thead>
<tr>
<th>Partition</th>
<th>Tasks</th>
<th>Nodes</th>
<th>StartOffset</th>
<th>Duration</th>
<th>StartDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>3</td>
<td>3</td>
<td>00:00:00</td>
<td>INFINITY</td>
<td>11:32:38_08/19</td>
</tr>
<tr>
<td>ReqID=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClusterA</td>
<td>1</td>
<td>1</td>
<td>00:00:00</td>
<td>INFINITY</td>
<td>11:32:38_08/19</td>
</tr>
<tr>
<td>ReqID=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClusterB</td>
<td>2</td>
<td>2</td>
<td>00:00:00</td>
<td>INFINITY</td>
<td>11:32:38_08/19</td>
</tr>
<tr>
<td>ReqID=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example 2**

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
```

backFill window (user: 'john' group: 'staff' partition: ALL) Mon Feb 16 08:28:54

partition ALL:
33 procs available with no time limit
Example 3

In this example, a resource availability window is requested for processors located only on nodes with at least 512 MB of memory. In the example above, the command output reports that no processors are available for immediate use which meet this constraint.

```
> showbf -m ' =512'

backfill window (user: 'john' group: 'staff' partition: ALL) Thu Jun 18 16:03:04

no procs available
```

See Also

- [Moab Client Installation](#) - explains how to distribute this command to client nodes
- `showq`
- `mdiag -t`
**showq**  
(*Show Queue*)

**Synopsis**

```
```

**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**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-b</code></td>
<td>display blocked jobs only</td>
</tr>
<tr>
<td><code>-c</code></td>
<td>display details about recently completed jobs (see example, JOBCPURGETIME).</td>
</tr>
<tr>
<td><code>-g</code></td>
<td>display grid job and system id's for all jobs.</td>
</tr>
<tr>
<td><code>-i</code></td>
<td>display extended details about idle jobs. (see example)</td>
</tr>
<tr>
<td><code>-l</code></td>
<td>display local/remote view. For use in a Grid environment, displays job usage of both local and remote compute resources.</td>
</tr>
<tr>
<td><code>-n</code></td>
<td>displays normal showq output, but lists job names under JOBID</td>
</tr>
<tr>
<td><code>-p</code></td>
<td>display only jobs assigned to the specified partition.</td>
</tr>
<tr>
<td><code>-r</code></td>
<td>display extended details about active (running) jobs. (see example)</td>
</tr>
<tr>
<td><code>-R</code></td>
<td>display only jobs which overlap the specified reservation.</td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Display local and full resource manager job IDs as well as partitions. If specified with the <code>-i</code> option, will display job reservation time.</td>
</tr>
<tr>
<td><code>-w</code></td>
<td>display only jobs associated with the specified constraint. Valid constraints include user, group, acct, class, and qos. (see showq -w example.)</td>
</tr>
</tbody>
</table>

**Details**

Beyond job information, the `showq` command 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**

- **Example 1**: Default Report
- **Example 2**: Detailed Active/Running Job Report
- **Example 3**: Detailed Eligible/Idle Job Report
- **Example 4**: Detailed Completed Job Report
- **Example 5**: Filtered Job Report
Example 1: Default Report

The output of this command is divided into three parts, Active Jobs, Eligible Jobs, and Blocked Jobs.

> showq

```
active jobs------------------------

<table>
<thead>
<tr>
<th>JOBID</th>
<th>STARTTIME</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROC</th>
<th>REMAINING/WCLIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>12941</td>
<td>15:02:50</td>
<td>sartois</td>
<td>Running</td>
<td>25</td>
<td>2:44:11</td>
</tr>
<tr>
<td>12954</td>
<td>15:02:52</td>
<td>tgates</td>
<td>Running</td>
<td>4</td>
<td>2:57:33</td>
</tr>
<tr>
<td>12944</td>
<td>15:02:50</td>
<td>eval1</td>
<td>Running</td>
<td>16</td>
<td>6:37:31</td>
</tr>
<tr>
<td>12946</td>
<td>15:02:50</td>
<td>tgates</td>
<td>Running</td>
<td>2</td>
<td>1:05:57:31</td>
</tr>
</tbody>
</table>

4 active jobs  47 of 48 processors active (97.92%)
32 of 32 nodes active (100.00%)

eligible jobs----------------------

<table>
<thead>
<tr>
<th>JOBID</th>
<th>QUEUETIME</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROC</th>
<th>WCLIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>12956</td>
<td>15:02:50</td>
<td>cfosdyke</td>
<td>Idle</td>
<td>32</td>
<td>6:40:00</td>
</tr>
<tr>
<td>12969</td>
<td>15:03:23</td>
<td>cfosdyke</td>
<td>Idle</td>
<td>4</td>
<td>6:40:00</td>
</tr>
<tr>
<td>12939</td>
<td>15:02:50</td>
<td>eval1</td>
<td>Idle</td>
<td>16</td>
<td>3:00:00</td>
</tr>
<tr>
<td>12940</td>
<td>15:02:50</td>
<td>mwillis</td>
<td>Idle</td>
<td>2</td>
<td>3:00:00</td>
</tr>
</tbody>
</table>
```

The fields are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOBID</td>
<td>job identifier.</td>
</tr>
<tr>
<td>USERNAME</td>
<td>User owning job.</td>
</tr>
<tr>
<td>STATE</td>
<td>Job State. Current batch state of the job.</td>
</tr>
<tr>
<td>PROC</td>
<td>Number of processors being used by the job.</td>
</tr>
<tr>
<td>REMAINING/WCLIMIT</td>
<td>For active jobs, the time the job has until it has reached its wall clock limit or for idle/blocked jobs, the amount of time requested by the job. Time specified in [DD:]HH:MM:SS notation.</td>
</tr>
<tr>
<td>STARTTIME</td>
<td>Time job started running.</td>
</tr>
</tbody>
</table>

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 id's may be marked with a single character to to specify the following conditions:
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 wall clock 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.

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:

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Job violates a fairness policy. Use diagnose -q for more information.</td>
</tr>
<tr>
<td>UserHold</td>
<td>A user hold is in place.</td>
</tr>
<tr>
<td>SystemHold</td>
<td>An administrative or system hold is in place.</td>
</tr>
<tr>
<td>BatchHold</td>
<td>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).</td>
</tr>
<tr>
<td>Deferred</td>
<td>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).</td>
</tr>
<tr>
<td>NotQueued</td>
<td>Job is in the resource manager state NQ (indicating the job’s controlling scheduling daemon is unavailable).</td>
</tr>
</tbody>
</table>

A summary of the job queue's status is provided at the end of the output.

Example 2: Detailed Active/Running Job Report

```
> showq -r

active jobs------------------------
JOBID        MHOST        PROCS REMAINING           S CODE PAR EFFIC XFACTOR Q         USER    GROUP
12941        G5-014      25  2:43:31  Thu Sep 1 15:02:50 R  -   3 100.00      1.0  -    sartois   Arches
12954        G5-016      4   2:56:54  Thu Sep 1 15:02:52 R  -   3 100.00      1.0 Hi    tgates   Arches
12944        P690-016   16  6:36:51  Thu Sep 1 15:02:50 R  -   3 100.00      1.0 De     eval1  RedRock
12946        G5-014      25  2:43:31  Thu Sep 1 15:02:50 R  -   3 100.00      1.0 -    tgates   Arches
```
The fields are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOBID</td>
<td>Name of active job.</td>
</tr>
<tr>
<td>S</td>
<td><strong>Job State.</strong> Either &quot;R&quot; for Running or &quot;S&quot; for Starting.</td>
</tr>
<tr>
<td>PAR</td>
<td>Partition in which job is running.</td>
</tr>
<tr>
<td>EFFIC</td>
<td>CPU efficiency of job.</td>
</tr>
<tr>
<td>XFACTOR</td>
<td>Current expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit</td>
</tr>
<tr>
<td>Q</td>
<td>Quality Of Service specified for job.</td>
</tr>
<tr>
<td>USERNAME</td>
<td>User owning job.</td>
</tr>
<tr>
<td>GROUP</td>
<td>Primary group of job owner.</td>
</tr>
<tr>
<td>MHOST</td>
<td>Master Host running primary task of job.</td>
</tr>
<tr>
<td>PROC</td>
<td>Number of processors being used by the job.</td>
</tr>
<tr>
<td>REMAINING</td>
<td>Time the job has until it has reached its wall clock limit. Time specified in HH:MM:SS notation.</td>
</tr>
<tr>
<td>STARTTIME</td>
<td>Time job started running.</td>
</tr>
</tbody>
</table>

After displaying the running jobs, a summary is provided indicating the number of jobs, the number of allocated processors, and the system utilization.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobName</td>
<td>Name of active job.</td>
</tr>
<tr>
<td>S</td>
<td><strong>Job State.</strong> Either &quot;R&quot; for Running or &quot;S&quot; for Starting.</td>
</tr>
<tr>
<td>CCode</td>
<td>Completion Code. The return/completion code given when a job completes. (Only applicable to completed jobs.)</td>
</tr>
<tr>
<td>Par</td>
<td>Partition in which job is running.</td>
</tr>
<tr>
<td>Effic</td>
<td>CPU efficiency of job.</td>
</tr>
<tr>
<td>XFactor</td>
<td>Current expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit</td>
</tr>
<tr>
<td>Q</td>
<td>Quality Of Service specified for job.</td>
</tr>
<tr>
<td>User</td>
<td>User owning job.</td>
</tr>
<tr>
<td>Group</td>
<td>Primary group of job owner.</td>
</tr>
<tr>
<td>Nodes</td>
<td>Number of processors being used by the job.</td>
</tr>
<tr>
<td>Remaining</td>
<td>Time the job has until it has reached its wall clock limit. Time specified in HH:MM:SS notation.</td>
</tr>
</tbody>
</table>
Start Time: Time job started running.

`> showq -i`

eligible jobs----------------------

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PRIORITY</th>
<th>XFACTOR</th>
<th>Q</th>
<th>USER</th>
<th>GROUP</th>
<th>PROCS</th>
<th>WCLIMIT</th>
<th>CLASS</th>
<th>SYSTEMQUEUETIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>12956*</td>
<td>20</td>
<td>1.0</td>
<td>-</td>
<td>cfosdyke</td>
<td>RedRock</td>
<td>32</td>
<td>6:40:00</td>
<td>batch</td>
<td>Thu Sep 1 15:02:50</td>
</tr>
<tr>
<td>12969*</td>
<td>19</td>
<td>1.0</td>
<td>-</td>
<td>cfosdyke</td>
<td>RedRock</td>
<td>4</td>
<td>6:40:00</td>
<td>batch</td>
<td>Thu Sep 1 15:03:23</td>
</tr>
<tr>
<td>12939</td>
<td>16</td>
<td>1.0</td>
<td>-</td>
<td>eval1</td>
<td>RedRock</td>
<td>16</td>
<td>3:00:00</td>
<td>batch</td>
<td>Thu Sep 1 15:02:50</td>
</tr>
<tr>
<td>12940</td>
<td>16</td>
<td>1.0</td>
<td>-</td>
<td>mwillis</td>
<td>Arches</td>
<td>2</td>
<td>3:00:00</td>
<td>batch</td>
<td>Thu Sep 1 15:02:50</td>
</tr>
<tr>
<td>12947</td>
<td>16</td>
<td>1.0</td>
<td>-</td>
<td>mwillis</td>
<td>Arches</td>
<td>2</td>
<td>3:00:00</td>
<td>batch</td>
<td>Thu Sep 1 15:02:50</td>
</tr>
<tr>
<td>12949</td>
<td>16</td>
<td>1.0</td>
<td>-</td>
<td>eval1</td>
<td>RedRock</td>
<td>2</td>
<td>3:00:00</td>
<td>batch</td>
<td>Thu Sep 1 15:02:50</td>
</tr>
<tr>
<td>12953</td>
<td>16</td>
<td>1.0</td>
<td>-</td>
<td>t gates</td>
<td>Arches</td>
<td>10</td>
<td>4:26:40</td>
<td>batch</td>
<td>Thu Sep 1 15:02:50</td>
</tr>
<tr>
<td>12955</td>
<td>16</td>
<td>1.0</td>
<td>-</td>
<td>eval1</td>
<td>RedRock</td>
<td>2</td>
<td>4:26:40</td>
<td>batch</td>
<td>Thu Sep 1 15:02:50</td>
</tr>
<tr>
<td>12957</td>
<td>16</td>
<td>1.0</td>
<td>-</td>
<td>t gates</td>
<td>Arches</td>
<td>16</td>
<td>3:00:00</td>
<td>batch</td>
<td>Thu Sep 1 15:02:50</td>
</tr>
<tr>
<td>12963</td>
<td>16</td>
<td>1.0</td>
<td>-</td>
<td>eval1</td>
<td>RedRock</td>
<td>16</td>
<td>1:06:00:00</td>
<td>batch</td>
<td>Thu Sep 1 15:02:52</td>
</tr>
<tr>
<td>12964</td>
<td>16</td>
<td>1.0</td>
<td>-</td>
<td>t gates</td>
<td>Arches</td>
<td>16</td>
<td>1:00:00:00</td>
<td>batch</td>
<td>Thu Sep 1 15:02:52</td>
</tr>
<tr>
<td>12937</td>
<td>1</td>
<td>1.0</td>
<td>-</td>
<td>allendr</td>
<td>RedRock</td>
<td>9</td>
<td>3:00:00</td>
<td>batch</td>
<td>Thu Sep 1 15:02:50</td>
</tr>
</tbody>
</table>

The fields are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOBID</td>
<td>Name of job.</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>Calculated job priority.</td>
</tr>
<tr>
<td>XFACTOR</td>
<td>Current expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit</td>
</tr>
<tr>
<td>Q</td>
<td>Quality Of Service specified for job.</td>
</tr>
<tr>
<td>USER</td>
<td>User owning job.</td>
</tr>
<tr>
<td>GROUP</td>
<td>Primary group of job owner.</td>
</tr>
<tr>
<td>PROCS</td>
<td>Minimum number of processors required to run job.</td>
</tr>
<tr>
<td>WCLIMIT</td>
<td>Wall clock limit specified for job. Time specified in HH:MM:SS notation.</td>
</tr>
<tr>
<td>CLASS</td>
<td>Class requested by job.</td>
</tr>
<tr>
<td>SYSTEMQUEUETIME</td>
<td>Time job was admitted into the system queue.</td>
</tr>
</tbody>
</table>

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 4: Detailed Completed Job Report**
The fields are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOBID</td>
<td>job id for completed job.</td>
</tr>
<tr>
<td>S</td>
<td>Job State. Either “C” for Completed or “V” for Vacated.</td>
</tr>
<tr>
<td>CCODE</td>
<td>Completion code reported by the job.</td>
</tr>
<tr>
<td>PAR</td>
<td>Partition in which job ran.</td>
</tr>
<tr>
<td>EFFIC</td>
<td>CPU efficiency of job.</td>
</tr>
<tr>
<td>XFACTOR</td>
<td>Expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit</td>
</tr>
<tr>
<td>Q</td>
<td>Quality of Service specified for job.</td>
</tr>
<tr>
<td>USERNAME</td>
<td>User owning job.</td>
</tr>
<tr>
<td>GROUP</td>
<td>Primary group of job owner.</td>
</tr>
<tr>
<td>MHOST</td>
<td>Master Host which ran the primary task of job.</td>
</tr>
<tr>
<td>PROC</td>
<td>Number of processors being used by the job.</td>
</tr>
<tr>
<td>WALLTIME</td>
<td>Wallclock time used by the job. Time specified in [DD:]HH:MM:SS notation.</td>
</tr>
<tr>
<td>STARTTIME</td>
<td>Time job started running.</td>
</tr>
</tbody>
</table>

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 5: Filtered Job Report

Show only jobs associated with user john and class benchmark

```bash
> showq -w class=benchmark -w user=john
...
```

See Also
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
showres

(Show Reservation)

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, or by any valid user if the parameter RSVCTLPOLICY is set to ANY.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-f</td>
<td>show free (unreserved) resources rather than reserved resources. The '-f' flag cannot be used in conjunction with the any other flag</td>
</tr>
<tr>
<td>-g</td>
<td>when used with the '-n' flag, shows 'grep'-able output with nodename on every line</td>
</tr>
<tr>
<td>-n</td>
<td>display information regarding all nodes reserved by &lt;RSVID&gt;</td>
</tr>
<tr>
<td>-o</td>
<td>display all reservations which overlap &lt;RSVID&gt; (in time and space) Note: not supported with '-n' flag</td>
</tr>
<tr>
<td>-r</td>
<td>display reservation timeframes in relative time mode</td>
</tr>
<tr>
<td>-v</td>
<td>show verbose output. If used with the '-n' flag, the command will display all reservations found on nodes contained in &lt;RSVID&gt;. Otherwise, it will show long reservation start dates including the reservation year.</td>
</tr>
</tbody>
</table>

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSVID</td>
<td>ID of reservation of interest - optional</td>
</tr>
</tbody>
</table>

Example 1

```
> showres

<table>
<thead>
<tr>
<th>ReservationID</th>
<th>Type</th>
<th>Start</th>
<th>End</th>
<th>Duration</th>
<th>N/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>12941</td>
<td>Job R</td>
<td>-00:05:01</td>
<td>2:41:39</td>
<td>2:46:40</td>
<td>13/25</td>
</tr>
<tr>
<td>Thu Sep 15:02:50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12944</td>
<td>Job R</td>
<td>-00:05:01</td>
<td>6:34:59</td>
<td>6:40:00</td>
<td>16/16</td>
</tr>
<tr>
<td>Thu Sep 15:02:50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12946</td>
<td>Job R</td>
<td>-00:05:01</td>
<td>1:05:54:59</td>
<td>1:06:00</td>
<td>1/2</td>
</tr>
<tr>
<td>Thu Sep 15:02:50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12954</td>
<td>Job R</td>
<td>-00:04:59</td>
<td>2:55:01</td>
<td>3:00:00</td>
<td>2/4</td>
</tr>
<tr>
<td>Thu Sep 15:02:52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12956</td>
<td>Job I</td>
<td>1:05:54:59</td>
<td>1:12:34:59</td>
<td>6:40:00</td>
<td>16/32</td>
</tr>
<tr>
<td>Fri Sep 21:02:50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12969</td>
<td>Job I</td>
<td>6:34:59</td>
<td>13:14:59</td>
<td>6:40:00</td>
<td>4/4</td>
</tr>
<tr>
<td>Thu Sep 21:42:50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6 reservations located
```
The above example shows all reservations on the system. The fields are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Reservation Type. This will be one of the following: Job, User, Group, Account, or System.</td>
</tr>
<tr>
<td>ReservationID</td>
<td>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.</td>
</tr>
<tr>
<td>S</td>
<td>State. This field is valid only for job reservations. It indicates whether the job is (S)tarting, (R)unning, or (I)dle.</td>
</tr>
<tr>
<td>Start</td>
<td>Relative start time of the reservation. Time is displayed in HH:MM:SS notation and is relative to the present time.</td>
</tr>
<tr>
<td>End</td>
<td>Relative end time of the reservation. Time is displayed in HH:MM:SS notation and is relative to the present time. Reservation that will not complete in 1,000 hours are marked with the keyword INFINITY.</td>
</tr>
<tr>
<td>Duration</td>
<td>Duration of the reservation in HH:MM:SS notation. Reservations lasting more than 1,000 hours are marked with the keyword INFINITY.</td>
</tr>
<tr>
<td>Nodes</td>
<td>Number of nodes involved in reservation.</td>
</tr>
<tr>
<td>StartTime</td>
<td>Time Reservation became active.</td>
</tr>
</tbody>
</table>

### Example 2

```
> showres -n
reservations on Thu Sep  1 16:49:59

<table>
<thead>
<tr>
<th>NodeName</th>
<th>Type</th>
<th>ReservationID</th>
<th>JobState</th>
<th>Task</th>
<th>StartTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>G5-001</td>
<td>Job</td>
<td>12946</td>
<td>Running</td>
<td>2</td>
<td>-1:47:09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1:06:00:00:00</td>
<td></td>
<td></td>
<td>Thu Sep 15:02:50</td>
</tr>
<tr>
<td>G5-001</td>
<td>Job</td>
<td>12956</td>
<td>Idle</td>
<td>2</td>
<td>1:04:12:51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6:40:00</td>
<td></td>
<td></td>
<td>Fri Sep 21:02:50</td>
</tr>
<tr>
<td>G5-002</td>
<td>Job</td>
<td>12956</td>
<td>Idle</td>
<td>2</td>
<td>1:04:12:51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6:40:00</td>
<td></td>
<td></td>
<td>Fri Sep 21:02:50</td>
</tr>
<tr>
<td>G5-002</td>
<td>Job</td>
<td>12953</td>
<td>Running</td>
<td>2</td>
<td>-00:29:37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4:26:40</td>
<td></td>
<td></td>
<td>Thu Sep 16:20:22</td>
</tr>
<tr>
<td>G5-003</td>
<td>Job</td>
<td>12956</td>
<td>Idle</td>
<td>2</td>
<td>1:04:12:51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6:40:00</td>
<td></td>
<td></td>
<td>Fri Sep 21:02:50</td>
</tr>
<tr>
<td>G5-003</td>
<td>Job</td>
<td>12953</td>
<td>Running</td>
<td>2</td>
<td>-00:29:37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4:26:40</td>
<td></td>
<td></td>
<td>Thu Sep 16:20:22</td>
</tr>
<tr>
<td>G5-004</td>
<td>Job</td>
<td>12956</td>
<td>Idle</td>
<td>2</td>
<td>1:04:12:51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6:40:00</td>
<td></td>
<td></td>
<td>Fri Sep 21:02:50</td>
</tr>
<tr>
<td>G5-004</td>
<td>Job</td>
<td>12953</td>
<td>Running</td>
<td>2</td>
<td>-00:29:37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4:26:40</td>
<td></td>
<td></td>
<td>Thu Sep 16:20:22</td>
</tr>
<tr>
<td>G5-005</td>
<td>Job</td>
<td>12956</td>
<td>Idle</td>
<td>2</td>
<td>1:04:12:51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6:40:00</td>
<td></td>
<td></td>
<td>Fri Sep 21:02:50</td>
</tr>
<tr>
<td>G5-005</td>
<td>Job</td>
<td>12953</td>
<td>Running</td>
<td>2</td>
<td>-00:29:37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4:26:40</td>
<td></td>
<td></td>
<td>Thu Sep 16:20:22</td>
</tr>
<tr>
<td>G5-006</td>
<td>Job</td>
<td>12956</td>
<td>Idle</td>
<td>2</td>
<td>1:04:12:51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6:40:00</td>
<td></td>
<td></td>
<td>Fri Sep 21:02:50</td>
</tr>
<tr>
<td>G5-006</td>
<td>Job</td>
<td>12953</td>
<td>Running</td>
<td>2</td>
<td>-00:29:37</td>
</tr>
</tbody>
</table>
```

This example shows reservations for nodes. The fields are as follows:
<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NodeName</strong></td>
<td>Node on which reservation is placed.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Reservation Type. This will be one of the following: Job, User, Group, Account, or System.</td>
</tr>
<tr>
<td><strong>ReservationID</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>JobState</strong></td>
<td>This field is valid only for job reservations. It indicates the state of the job associated with the reservation.</td>
</tr>
<tr>
<td><strong>Start</strong></td>
<td>Relative start time of the reservation. Time is displayed in HH:MM:SS notation and is relative to the present time.</td>
</tr>
<tr>
<td><strong>Duration</strong></td>
<td>Duration of the reservation in HH:MM:SS notation. Reservations lasting more than 1000 hours are marked with the keyword INFINITY.</td>
</tr>
<tr>
<td><strong>StartTime</strong></td>
<td>Time Reservation became active.</td>
</tr>
</tbody>
</table>

**Example 3**

```
> 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.

**See Also:**

- [Moab Client Installation](#) - explains how to distribute this command to client nodes
- `mrsvctl -c` - create new reservations.
- `mrsvctl -r` - release existing reservations.
- `mddiag -r` - diagnose/view the state of existing reservations.
- [Reservation Overview](#) - description of reservations and their use.
showstart

(Show Start Estimate)

Synopsis

showstart {jobid|proccount[@duration]|s3jobspec} [-e {all|hist|prio|rsv}]
[-f] [-g [peer]] [-l qos=<QOS>] [--format=xml]

Overview

This command displays the estimated start time of a job based on 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.

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DURATION</td>
<td>duration of pseudo-job to be checked in format [[DD:]HH:]MM:SS (default duration is 1 second)</td>
</tr>
<tr>
<td>-e</td>
<td>estimate method. By default, Moab will use the reservation based estimation method.</td>
</tr>
<tr>
<td>-f</td>
<td>use feedback. If specified, Moab will apply historical accuracy information to improve the quality of the estimate. See ENABLESTARTESTIMATESTATS for more information.</td>
</tr>
<tr>
<td>-g</td>
<td>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 form the remote Moab where the job was migrated to. All resource managers can be queried by using the keyword &quot;all&quot; which returns all information in a table.</td>
</tr>
<tr>
<td>-l qos=&lt;QOS&gt;</td>
<td>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</td>
</tr>
</tbody>
</table>
hypothetical jobs by using the proccount[@duration] syntax.

<table>
<thead>
<tr>
<th>JOBID</th>
<th>job to be checked</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCCOUNT</td>
<td>number of processors in pseudo-job to be checked</td>
</tr>
<tr>
<td>S3JOBSPEC</td>
<td>XML describing the job according to the Dept. of Energy Scalable Systems Software/S3 job specification.</td>
</tr>
</tbody>
</table>

Example 1

```bash
> 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 2

```bash
> 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).

See Also

- Moab Client Installation - explains how to distribute this command to client nodes
- checkjob
- showres
- showstats -f eststarttime
- showstats -f avgqtime
- Job Start Estimates
showstate

(Show State)

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.

Example 1

> showstate

cluster state summary for Wed Nov 23 12:00:21

<table>
<thead>
<tr>
<th>JobID</th>
<th>S</th>
<th>User</th>
<th>Group</th>
<th>Procs</th>
<th>Remaining</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartTime</td>
<td>-----</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>(A) fr17n11.942.0</td>
<td>R</td>
<td>johns</td>
<td>staff</td>
<td>16</td>
<td>13:21:15</td>
</tr>
<tr>
<td>Nov 22 12:00:21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(B) fr17n11.942.0</td>
<td>S</td>
<td>johns</td>
<td>staff</td>
<td>32</td>
<td>13:07:11</td>
</tr>
<tr>
<td>Nov 22 12:00:21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(C) fr17n11.942.0</td>
<td>R</td>
<td>johns</td>
<td>staff</td>
<td>8</td>
<td>11:22:25</td>
</tr>
<tr>
<td>Nov 22 12:00:21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(D) fr17n11.942.0</td>
<td>S</td>
<td>johns</td>
<td>staff</td>
<td>8</td>
<td>10:43:43</td>
</tr>
<tr>
<td>Nov 22 12:00:21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E) fr17n11.942.0</td>
<td>S</td>
<td>johns</td>
<td>staff</td>
<td>8</td>
<td>9:19:25</td>
</tr>
<tr>
<td>Nov 22 12:00:21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F) fr17n11.942.0</td>
<td>R</td>
<td>johns</td>
<td>staff</td>
<td>8</td>
<td>9:01:16</td>
</tr>
<tr>
<td>Nov 22 12:00:21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(G) fr17n11.942.0</td>
<td>R</td>
<td>johns</td>
<td>staff</td>
<td>1</td>
<td>7:28:25</td>
</tr>
<tr>
<td>Nov 22 12:00:21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(H) fr17n11.942.0</td>
<td>R</td>
<td>johns</td>
<td>staff</td>
<td>1</td>
<td>3:05:17</td>
</tr>
<tr>
<td>Nov 22 12:00:21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(I) fr17n11.942.0</td>
<td>S</td>
<td>johns</td>
<td>staff</td>
<td>24</td>
<td>0:54:38</td>
</tr>
<tr>
<td>Nov 22 12:00:22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Usage Summary: 9 Active Jobs 106 Active Nodes

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, fr3n06...

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 fr1n08 is in state BUSY but has no job assigned to it (it possibly has a runaway job running on it).
See Also

- Moab Client Installation - explains how to distribute this command to client nodes
- Specifying Node Rack/Slot Location
showstats

(Show Statistics)

Synopsis

showstats
showstats -a [accountid] [-v] [-t <TIMESPEC>]
showstats -c [classid] [-v] [-t <TIMESPEC>]
showstats -f [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 resetstats command.

![It is not recommended to query for timeframes larger than one month if Moab is configured without a database. Doing so may cause Moab to slow down significantly or stop responding. For large queries, configure Moab with a database connection.]

Access

By default, this command can be run by any Moab level 1, 2, or 3 Administrator.

Parameters

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a [ACCOUNTID]</td>
<td>display account statistics</td>
</tr>
<tr>
<td>-c [CLASSID]</td>
<td>display class statistics</td>
</tr>
<tr>
<td>-f</td>
<td>display full matrix statistics (see showstats -f for full details)</td>
</tr>
<tr>
<td>-g [GROUPID]</td>
<td>display group statistics</td>
</tr>
<tr>
<td>-j [JOBTEMPLATE]</td>
<td>display template statistics</td>
</tr>
<tr>
<td>-n [NODEID]</td>
<td>display node statistics (ENABLEPROFILING must be set)</td>
</tr>
<tr>
<td>-q [QOSID]</td>
<td>display QoS statistics</td>
</tr>
<tr>
<td>-s</td>
<td>display general scheduler statistics</td>
</tr>
<tr>
<td>-t</td>
<td>display statistical information from the specified timeframe:</td>
</tr>
<tr>
<td></td>
<td>&lt;TIME&gt;[,&lt;TIME&gt;]</td>
</tr>
<tr>
<td></td>
<td>(ABSTIME: [HH[:MM[:SS]][_MO[/DD[/YY]]] ie 14:30_06/20)</td>
</tr>
<tr>
<td></td>
<td>(RELTIME: +[[[DD:]HH:]MM:]SS)</td>
</tr>
</tbody>
</table>

![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.]


Example 1

> showstats -a

Account Statistics Initialized Tue Aug 26 14:32:39

|------ Running ------|-------------------------------|
|----- Completed ----------------------------------|

<table>
<thead>
<tr>
<th>Account</th>
<th>Jobs</th>
<th>Procs</th>
<th>ProcHours</th>
<th>Jobs</th>
<th>%</th>
<th>PHReq</th>
<th>%</th>
<th>PHDed</th>
</tr>
</thead>
<tbody>
<tr>
<td>137651</td>
<td>16</td>
<td>92</td>
<td>1394.52</td>
<td>229</td>
<td>39.15</td>
<td>18486</td>
<td>45.26</td>
<td>7003.5</td>
</tr>
<tr>
<td>41.54</td>
<td>40.00</td>
<td>8.15</td>
<td>5.21</td>
<td>90.70</td>
<td>34.69</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>462212</td>
<td>11</td>
<td>63</td>
<td>855.27</td>
<td>43</td>
<td>7.35</td>
<td>6028</td>
<td>14.76</td>
<td>3448.4</td>
</tr>
<tr>
<td>20.45</td>
<td>6.25</td>
<td>5.40</td>
<td>3.14</td>
<td>98.64</td>
<td>40.83</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>462213</td>
<td>6</td>
<td>72</td>
<td>728.12</td>
<td>90</td>
<td>15.38</td>
<td>5974</td>
<td>14.63</td>
<td>3170.7</td>
</tr>
<tr>
<td>18.81</td>
<td>6.25</td>
<td>4.88</td>
<td>0.52</td>
<td>82.01</td>
<td>24.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>005810</td>
<td>3</td>
<td>24</td>
<td>220.72</td>
<td>77</td>
<td>13.16</td>
<td>2537</td>
<td>6.21</td>
<td>1526.6</td>
</tr>
<tr>
<td>9.06</td>
<td>-----</td>
<td>1.53</td>
<td>0.42</td>
<td>98.73</td>
<td>28.40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>462213</td>
<td>6</td>
<td>72</td>
<td>728.12</td>
<td>90</td>
<td>15.38</td>
<td>5974</td>
<td>14.63</td>
<td>3170.7</td>
</tr>
<tr>
<td>18.81</td>
<td>6.25</td>
<td>4.88</td>
<td>0.52</td>
<td>82.01</td>
<td>24.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>005810</td>
<td>3</td>
<td>24</td>
<td>220.72</td>
<td>77</td>
<td>13.16</td>
<td>2537</td>
<td>6.21</td>
<td>1526.6</td>
</tr>
<tr>
<td>9.06</td>
<td>-----</td>
<td>1.53</td>
<td>0.42</td>
<td>98.73</td>
<td>28.40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>462213</td>
<td>6</td>
<td>72</td>
<td>728.12</td>
<td>90</td>
<td>15.38</td>
<td>5974</td>
<td>14.63</td>
<td>3170.7</td>
</tr>
<tr>
<td>18.81</td>
<td>6.25</td>
<td>4.88</td>
<td>0.52</td>
<td>82.01</td>
<td>24.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>005810</td>
<td>3</td>
<td>24</td>
<td>220.72</td>
<td>77</td>
<td>13.16</td>
<td>2537</td>
<td>6.21</td>
<td>1526.6</td>
</tr>
<tr>
<td>9.06</td>
<td>-----</td>
<td>1.53</td>
<td>0.42</td>
<td>98.73</td>
<td>28.40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>462213</td>
<td>6</td>
<td>72</td>
<td>728.12</td>
<td>90</td>
<td>15.38</td>
<td>5974</td>
<td>14.63</td>
<td>3170.7</td>
</tr>
<tr>
<td>18.81</td>
<td>6.25</td>
<td>4.88</td>
<td>0.52</td>
<td>82.01</td>
<td>24.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>005810</td>
<td>3</td>
<td>24</td>
<td>220.72</td>
<td>77</td>
<td>13.16</td>
<td>2537</td>
<td>6.21</td>
<td>1526.6</td>
</tr>
<tr>
<td>9.06</td>
<td>-----</td>
<td>1.53</td>
<td>0.42</td>
<td>98.73</td>
<td>28.40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>462213</td>
<td>6</td>
<td>72</td>
<td>728.12</td>
<td>90</td>
<td>15.38</td>
<td>5974</td>
<td>14.63</td>
<td>3170.7</td>
</tr>
<tr>
<td>18.81</td>
<td>6.25</td>
<td>4.88</td>
<td>0.52</td>
<td>82.01</td>
<td>24.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>005810</td>
<td>3</td>
<td>24</td>
<td>220.72</td>
<td>77</td>
<td>13.16</td>
<td>2537</td>
<td>6.21</td>
<td>1526.6</td>
</tr>
<tr>
<td>9.06</td>
<td>-----</td>
<td>1.53</td>
<td>0.42</td>
<td>98.73</td>
<td>28.40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>462213</td>
<td>6</td>
<td>72</td>
<td>728.12</td>
<td>90</td>
<td>15.38</td>
<td>5974</td>
<td>14.63</td>
<td>3170.7</td>
</tr>
<tr>
<td>18.81</td>
<td>6.25</td>
<td>4.88</td>
<td>0.52</td>
<td>82.01</td>
<td>24.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>005810</td>
<td>3</td>
<td>24</td>
<td>220.72</td>
<td>77</td>
<td>13.16</td>
<td>2537</td>
<td>6.21</td>
<td>1526.6</td>
</tr>
<tr>
<td>9.06</td>
<td>-----</td>
<td>1.53</td>
<td>0.42</td>
<td>98.73</td>
<td>28.40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>462213</td>
<td>6</td>
<td>72</td>
<td>728.12</td>
<td>90</td>
<td>15.38</td>
<td>5974</td>
<td>14.63</td>
<td>3170.7</td>
</tr>
<tr>
<td>18.81</td>
<td>6.25</td>
<td>4.88</td>
<td>0.52</td>
<td>82.01</td>
<td>24.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>005810</td>
<td>3</td>
<td>24</td>
<td>220.72</td>
<td>77</td>
<td>13.16</td>
<td>2537</td>
<td>6.21</td>
<td>1526.6</td>
</tr>
<tr>
<td>9.06</td>
<td>-----</td>
<td>1.53</td>
<td>0.42</td>
<td>98.73</td>
<td>28.40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>462213</td>
<td>6</td>
<td>72</td>
<td>728.12</td>
<td>90</td>
<td>15.38</td>
<td>5974</td>
<td>14.63</td>
<td>3170.7</td>
</tr>
<tr>
<td>18.81</td>
<td>6.25</td>
<td>4.88</td>
<td>0.52</td>
<td>82.01</td>
<td>24.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>005810</td>
<td>3</td>
<td>24</td>
<td>220.72</td>
<td>77</td>
<td>13.16</td>
<td>2537</td>
<td>6.21</td>
<td>1526.6</td>
</tr>
<tr>
<td>9.06</td>
<td>-----</td>
<td>1.53</td>
<td>0.42</td>
<td>98.73</td>
<td>28.40</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account</td>
<td>Account Number.</td>
</tr>
<tr>
<td>Jobs</td>
<td>Number of running jobs.</td>
</tr>
<tr>
<td>Procs</td>
<td>Number of processors allocated to running jobs.</td>
</tr>
<tr>
<td>ProcHours</td>
<td>Number of proc-hours required to complete running jobs.</td>
</tr>
<tr>
<td>Jobs*</td>
<td>Number of jobs completed.</td>
</tr>
<tr>
<td>%</td>
<td>Percentage of total jobs that were completed by account.</td>
</tr>
<tr>
<td>PHReq*</td>
<td>Total proc-hours requested by completed jobs.</td>
</tr>
<tr>
<td>%</td>
<td>Percentage of total proc-hours requested by completed jobs that were requested by account.</td>
</tr>
<tr>
<td>PHDed</td>
<td>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.</td>
</tr>
</tbody>
</table>
**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: \((\text{QueuedTime} + \text{RunTime}) / \text{WallClockLimit}\).

**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 wall clock accuracy for jobs completed. Wall clock accuracy is calculated by dividing a job's actual run time by its specified wall clock 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 2

```
> showstats -g

Group Statistics Initialized Tue Aug 26 14:32:39

|----- Running ------|-------------------------------|
--- Completed ----------------------------------|

<table>
<thead>
<tr>
<th>GroupName</th>
<th>GID</th>
<th>Jobs</th>
<th>Procs</th>
<th>ProcHours</th>
<th>% PHReq</th>
<th>% PHDed</th>
</tr>
</thead>
<tbody>
<tr>
<td>univ</td>
<td>214</td>
<td>16</td>
<td>92</td>
<td>1394.52</td>
<td>39.15</td>
<td>18486</td>
</tr>
<tr>
<td></td>
<td>41.54</td>
<td>40.00</td>
<td>0.77</td>
<td>8.15</td>
<td>5.21</td>
<td>90.70</td>
</tr>
<tr>
<td>daf</td>
<td>204</td>
<td>11</td>
<td>63</td>
<td>855.27</td>
<td>43</td>
<td>7.35</td>
</tr>
<tr>
<td></td>
<td>20.45</td>
<td>6.25</td>
<td>0.71</td>
<td>5.40</td>
<td>3.14</td>
<td>98.64</td>
</tr>
<tr>
<td>dnavy</td>
<td>207</td>
<td>6</td>
<td>72</td>
<td>728.12</td>
<td>90</td>
<td>15.38</td>
</tr>
<tr>
<td></td>
<td>18.81</td>
<td>6.25</td>
<td>0.37</td>
<td>4.88</td>
<td>0.52</td>
<td>82.01</td>
</tr>
<tr>
<td>govt</td>
<td>232</td>
<td>3</td>
<td>24</td>
<td>220.72</td>
<td>77</td>
<td>13.16</td>
</tr>
<tr>
<td></td>
<td>59.06</td>
<td>1.53</td>
<td>4.81</td>
<td>0.42</td>
<td>98.73</td>
<td>28.40</td>
</tr>
<tr>
<td>asp</td>
<td>227</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>12</td>
<td>2.05</td>
</tr>
<tr>
<td></td>
<td>5.69</td>
<td>2.50</td>
<td>1.78</td>
<td>8.61</td>
<td>5.60</td>
<td>83.64</td>
</tr>
<tr>
<td>derim</td>
<td>229</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>74</td>
<td>12.65</td>
</tr>
<tr>
<td></td>
<td>2.09</td>
<td>0.50</td>
<td>1.93</td>
<td>0.51</td>
<td>96.03</td>
<td>32.60</td>
</tr>
<tr>
<td>dchall</td>
<td>274</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>3</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>25.00</td>
<td>0.52</td>
<td>0.88</td>
<td>2.49</td>
<td>95.82</td>
</tr>
<tr>
<td>nih</td>
<td>239</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>17</td>
<td>2.91</td>
</tr>
<tr>
<td></td>
<td>0.88</td>
<td>0.95</td>
<td>1.83</td>
<td>0.14</td>
<td>97.59</td>
<td>84.31</td>
</tr>
<tr>
<td>darmy</td>
<td>205</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>31</td>
<td>5.30</td>
</tr>
<tr>
<td></td>
<td>0.32</td>
<td>6.25</td>
<td>0.14</td>
<td>0.59</td>
<td>0.07</td>
<td>81.33</td>
</tr>
<tr>
<td>systems</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>6</td>
<td>1.03</td>
</tr>
<tr>
<td></td>
<td>0.13</td>
<td>4.07</td>
<td>8.49</td>
<td>1.23</td>
<td>28.68</td>
<td>37.34</td>
</tr>
<tr>
<td>pdc</td>
<td>252</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>1</td>
<td>0.17</td>
</tr>
</tbody>
</table>
```

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:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupName</td>
<td>Name of group.</td>
</tr>
<tr>
<td>GID</td>
<td>Group ID of group.</td>
</tr>
<tr>
<td>Jobs</td>
<td>Number of running jobs.</td>
</tr>
<tr>
<td>Procs</td>
<td>Number of procs allocated to running jobs.</td>
</tr>
<tr>
<td>Prochours</td>
<td>Number of proc-hours required to complete running jobs.</td>
</tr>
<tr>
<td>Jobs*</td>
<td>Number of jobs completed.</td>
</tr>
<tr>
<td>%</td>
<td>Percentage of total jobs that were completed by group.</td>
</tr>
<tr>
<td>PHReq*</td>
<td>Total proc-hours requested by completed jobs.</td>
</tr>
<tr>
<td>%</td>
<td>Percentage of total proc-hours requested by completed jobs that were requested by group.</td>
</tr>
<tr>
<td>PHDed</td>
<td>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.</td>
</tr>
<tr>
<td>%</td>
<td>Percentage of total proc-hours that were dedicated by group.</td>
</tr>
<tr>
<td>FSTgt</td>
<td>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.</td>
</tr>
<tr>
<td>AvgXF*</td>
<td>Average expansion factor for jobs completed. A job’s XFactor (expansion factor) is calculated by the following formula: (QueuedTime + RunTime) / WallClockLimit.</td>
</tr>
<tr>
<td>MaxXF*</td>
<td>Highest expansion factor received by jobs completed.</td>
</tr>
<tr>
<td>AvgQH*</td>
<td>Average queue time (in hours) of jobs.</td>
</tr>
<tr>
<td>Effic</td>
<td>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.</td>
</tr>
<tr>
<td>WCAcc*</td>
<td>Average wall clock accuracy for jobs completed. Wall clock accuracy is calculated by dividing a job’s actual run time by its specified wall clock limit.</td>
</tr>
</tbody>
</table>

*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**

```bash
> 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
```
Example 4

> 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:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Jobs</td>
<td>Number of jobs currently active (Running or Starting).</td>
</tr>
<tr>
<td>Eligible Jobs</td>
<td>Number of jobs in the system queue (jobs that are considered when scheduling).</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Idle Jobs</td>
<td>Number of jobs both in and out of the system queue that are in the LoadLeveler Idle state.</td>
</tr>
<tr>
<td>Completed Jobs</td>
<td>Number of jobs completed since statistics were initialized.</td>
</tr>
<tr>
<td>Successful Jobs</td>
<td>Jobs that completed successfully without abnormal termination.</td>
</tr>
<tr>
<td>XFactor</td>
<td>Average expansion factor of all completed jobs.</td>
</tr>
<tr>
<td>Max XFactor</td>
<td>Maximum expansion factor of completed jobs.</td>
</tr>
<tr>
<td>Max Bypass</td>
<td>Maximum bypass of completed jobs.</td>
</tr>
<tr>
<td>Available ProcHours</td>
<td>Total proc-hours available to the scheduler.</td>
</tr>
<tr>
<td>Dedicated ProcHours</td>
<td>Total proc-hours made available to jobs.</td>
</tr>
<tr>
<td>Effic</td>
<td>Scheduling efficiency (DedicatedProcHours / Available ProcHours).</td>
</tr>
<tr>
<td>Min Efficiency</td>
<td>Minimum scheduling efficiency obtained since scheduler was started.</td>
</tr>
<tr>
<td>Iteration</td>
<td>Iteration on which the minimum scheduling efficiency occurred.</td>
</tr>
<tr>
<td>Available Procs</td>
<td>Number of procs currently available.</td>
</tr>
<tr>
<td>Busy Procs</td>
<td>Number of procs currently busy.</td>
</tr>
<tr>
<td>Effic</td>
<td>Current system efficiency (BusyProcs/AvailableProcs).</td>
</tr>
<tr>
<td>WallClock Accuracy</td>
<td>Average wall clock accuracy of completed jobs (job-weighted average).</td>
</tr>
<tr>
<td>Job Efficiency</td>
<td>Average job efficiency (UtilizedTime / DedicatedTime).</td>
</tr>
<tr>
<td>Est Backlog</td>
<td>Estimated backlog of queued work in hours.</td>
</tr>
<tr>
<td>Avg Backlog</td>
<td>Average backlog of queued work in hours.</td>
</tr>
</tbody>
</table>

**Example 5**

```
> showstats -u

User Statistics Initialized Tue Aug 26 14:32:39

<p>|----- Running ------|-------------------------------|
|-----|-------|-------------------------------|</p>
<table>
<thead>
<tr>
<th>UserName</th>
<th>UID</th>
<th>Jobs</th>
<th>Procs</th>
<th>ProcHours</th>
<th>Jobs</th>
<th>% PHReq</th>
<th>% PHDed</th>
</tr>
</thead>
<tbody>
<tr>
<td>moorejt</td>
<td>2617</td>
<td>1</td>
<td>16</td>
<td>58.80</td>
<td>2</td>
<td>0.34</td>
<td>221</td>
</tr>
<tr>
<td>zhong</td>
<td>1767</td>
<td>3</td>
<td>24</td>
<td>220.72</td>
<td>20</td>
<td>3.42</td>
<td>2306</td>
</tr>
<tr>
<td>lui</td>
<td>2467</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>16</td>
<td>2.74</td>
<td>1970</td>
</tr>
<tr>
<td>evans</td>
<td>3092</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>62</td>
<td>10.60</td>
<td>4960</td>
</tr>
<tr>
<td>wengel</td>
<td>2430</td>
<td>2</td>
<td>64</td>
<td>824.90</td>
<td>1</td>
<td>0.17</td>
<td>767</td>
</tr>
<tr>
<td>mukho</td>
<td>2961</td>
<td>2</td>
<td>16</td>
<td>71.06</td>
<td>6</td>
<td>1.03</td>
<td>776</td>
</tr>
<tr>
<td>jimenez</td>
<td>1449</td>
<td>1</td>
<td>16</td>
<td>302.29</td>
<td>2</td>
<td>0.34</td>
<td>768</td>
</tr>
</tbody>
</table>
```

---

**Example 5**

```
> showstats -u

User Statistics Initialized Tue Aug 26 14:32:39

<p>|----- Running ------|-------------------------------|
|-----|-------|-------------------------------|</p>
<table>
<thead>
<tr>
<th>UserName</th>
<th>UID</th>
<th>Jobs</th>
<th>Procs</th>
<th>ProcHours</th>
<th>Jobs</th>
<th>% PHReq</th>
<th>% PHDed</th>
</tr>
</thead>
<tbody>
<tr>
<td>moorejt</td>
<td>2617</td>
<td>1</td>
<td>16</td>
<td>58.80</td>
<td>2</td>
<td>0.34</td>
<td>221</td>
</tr>
<tr>
<td>zhong</td>
<td>1767</td>
<td>3</td>
<td>24</td>
<td>220.72</td>
<td>20</td>
<td>3.42</td>
<td>2306</td>
</tr>
<tr>
<td>lui</td>
<td>2467</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>16</td>
<td>2.74</td>
<td>1970</td>
</tr>
<tr>
<td>evans</td>
<td>3092</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>62</td>
<td>10.60</td>
<td>4960</td>
</tr>
<tr>
<td>wengel</td>
<td>2430</td>
<td>2</td>
<td>64</td>
<td>824.90</td>
<td>1</td>
<td>0.17</td>
<td>767</td>
</tr>
<tr>
<td>mukho</td>
<td>2961</td>
<td>2</td>
<td>16</td>
<td>71.06</td>
<td>6</td>
<td>1.03</td>
<td>776</td>
</tr>
<tr>
<td>jimenez</td>
<td>1449</td>
<td>1</td>
<td>16</td>
<td>302.29</td>
<td>2</td>
<td>0.34</td>
<td>768</td>
</tr>
</tbody>
</table>
```
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:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserName</td>
<td>Name of user.</td>
</tr>
<tr>
<td>UID</td>
<td>User ID of user.</td>
</tr>
<tr>
<td>Jobs</td>
<td>Number of running jobs.</td>
</tr>
<tr>
<td>Procs</td>
<td>Number of procs allocated to running jobs.</td>
</tr>
<tr>
<td>ProcHours</td>
<td>Number of proc-hours required to complete running jobs.</td>
</tr>
<tr>
<td>Jobs*</td>
<td>Number of jobs completed.</td>
</tr>
<tr>
<td>%</td>
<td>Percentage of total jobs that were completed by user.</td>
</tr>
<tr>
<td>PHReq*</td>
<td>Total proc-hours requested by completed jobs.</td>
</tr>
<tr>
<td>%</td>
<td>Percentage of total proc-hours requested by completed jobs that were requested by user.</td>
</tr>
<tr>
<td>PHDed</td>
<td>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.</td>
</tr>
<tr>
<td>%</td>
<td>Percentage of total proc-hours dedicated that were dedicated by user.</td>
</tr>
<tr>
<td>FSTgt</td>
<td>Fairshare target. A user's fairshare target is specified in the \texttt{fs.cfg} file. This value should be compared to the user's node-hour dedicated percentage to determine if the target is being met.</td>
</tr>
<tr>
<td>AvgXF*</td>
<td>Average expansion factor for jobs completed. A job's XFactor (expansion factor) is calculated by the following formula: ((\text{QueuedTime} + \text{RunTime}) / \text{WallClockLimit})$.</td>
</tr>
<tr>
<td>MaxXF*</td>
<td>Highest expansion factor received by jobs completed.</td>
</tr>
<tr>
<td>AvgQH*</td>
<td>Average queue time (in hours) of jobs.</td>
</tr>
<tr>
<td>Effic</td>
<td>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.</td>
</tr>
<tr>
<td>WCAcc*</td>
<td>Average wallclock accuracy for jobs completed. Wallclock accuracy is calculated by dividing a job's actual run time by its specified wallclock limit.</td>
</tr>
</tbody>
</table>

*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 6

```plaintext
> showstats -T
statistics initialized Mon Jul 10 15:29:41

|-------- Active ---------|-------------------------|-------------------------|
|--- Completed ---|-------------------------|
user | Jobs | Procs | ProcHours | Mem | Jobs | % | PHReq | % |
-----|-----|-------|-----------|-----|------|---|-------|---|
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 |
| 11.1 | 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 |
| 11.2 | 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 ---------|-------------------------|
```

**See Also**

- [Moab Client Installation](#) - explains how to distribute this command to client nodes
- `resetstats` command - re-initialize statistics
- `showstats -f` command - display full matrix statistics
showstats -f

(showstats -f)

Synopsis

showstats -f statisticstype

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, wall clock accuracy, and backfill information.

Access

This command can be run by any Moab Scheduler Administrator.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVGBYPASS</td>
<td>Average bypass count. Includes summary of job-weighted expansion bypass and total samples.</td>
</tr>
<tr>
<td>AVGQTIME</td>
<td>Average queue time. Includes summary of job-weighted queue time and total samples.</td>
</tr>
<tr>
<td>AVGXFACOR</td>
<td>Average expansion factor. Includes summary of job-weighted expansion factor, processor-weighted expansion factor, processor-hour-weighted expansion factor, and total number of samples.</td>
</tr>
<tr>
<td>BFCOUNT</td>
<td>Number of jobs backfilled. Includes summary of job-weighted backfill job percent and total samples.</td>
</tr>
<tr>
<td>BFPHRUN</td>
<td>Number of proc-hours backfilled. Includes summary of job-weighted backfill proc-hour percentage and total samples.</td>
</tr>
<tr>
<td>ESTSTARTTIME</td>
<td>Job start time estimate for jobs meeting specified processor/duration criteria. This estimate is based on the reservation start time analysis algorithm.</td>
</tr>
<tr>
<td>JOBCOUNT</td>
<td>Number of jobs. Includes summary of total jobs and total samples.</td>
</tr>
<tr>
<td>JOBEFFICIENCY</td>
<td>Job efficiency. Includes summary of job-weighted job efficiency percent and total samples.</td>
</tr>
<tr>
<td>MAXBYPASS</td>
<td>Maximum bypass count. Includes summary of overall maximum bypass and total samples.</td>
</tr>
<tr>
<td>MAXXFACOR</td>
<td>Maximum expansion factor. Includes summary of overall maximum expansion factor and total samples.</td>
</tr>
<tr>
<td>PHREQUEST</td>
<td>proc-hours requested. Includes summary of total proc-hours requested and total samples.</td>
</tr>
<tr>
<td>PHRUN</td>
<td>proc-hours run. Includes summary of total proc-hours run and total samples.</td>
</tr>
<tr>
<td>QOSDELIVERED</td>
<td>Quality of service delivered. Includes summary of job-weighted quality of service success rate and total samples.</td>
</tr>
<tr>
<td>WCACCURACY</td>
<td>Wall clock accuracy. Includes summary of overall wall clock accuracy and total samples.</td>
</tr>
</tbody>
</table>

Example 1

> showstats -f AVGXFACOR
The `showstats -f` command returns a table with data for the specified `STATISTICTYPE` parameter. The leftmost column shows the maximum number of processors required by the jobs shown in the other columns. The column headers indicate the maximum wall clock 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.

### See Also
- [Moab Client Installation](#) - explains how to distribute this command to client nodes
- `resetstats` command
- `showstats` command
- `STATPROCMIN` parameter
- `STATPROCSTEPCOUNT` parameter
- `STATPROCSTEPSIZE` parameter
- `STATTIMemin` parameter
- `STATTIMESTEPcount` parameter
- `STATTIMESTEPSIZE` parameter
canceljob

(Cancel Job)

⚠️ 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 AdmCfg).

<table>
<thead>
<tr>
<th>Flag</th>
<th>Name</th>
<th>Format</th>
<th>Default</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>HELP</td>
<td>N/A</td>
<td></td>
<td>Display usage information</td>
<td>&gt; canceljob -h</td>
</tr>
<tr>
<td></td>
<td>JOB ID</td>
<td>&lt;STRING&gt;</td>
<td>---</td>
<td>a jobid, a job expression, or the keyword 'ALL'</td>
<td>&gt; canceljob 13001 13003</td>
</tr>
</tbody>
</table>

Example 1

Cancel job 6397

> canceljob 6397
**changeparam**

**(Change Parameter)**

⚠️ This command is deprecated. Use `mschedctl -m` instead.

**Synopsis**

`changeparam parameter value`

**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 affect at the beginning of the next scheduling iteration. They are not persistent, only lasting until Moab is shutdown.

`changeparam` is a compact command of `mschedctl -m`.

**Access**

This command can be run by a level 1 Moab administrator.

**Format**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Name</th>
<th>Format</th>
<th>Default</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PARAMETER</td>
<td>&lt;STRING&gt;</td>
<td>[NONE]</td>
<td>The name a Moab configuration parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VALUE</td>
<td>&lt;STRING&gt;</td>
<td>[NONE]</td>
<td>any valid value for <code>&lt;PARAMETER&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

**Example 1**

Set Moab's LOGLEVEL to 6 for the current run:

```plaintext
changeparam

> changeparam LOGLEVEL 6
parameters changed
```

**Example 2**

Set Moab's ADMIN1 userlist to sys, mike and peter

```plaintext
changeparam

> changeparam ADMIN1 sys mike peter
parameters changed
```
diagnose
(Diagnostics)

⚠️ This command is deprecated. Use mdiag instead.

Synopsis

```shell
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
(Release Hold)

⚠️ 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.

Example 1
releasehold -b

> releasehold -b 6443
batch hold released for job 6443

In this example, a batch hold was released from this one job.

Example 2
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.

See Also
- sethold
• mjobctl
releaseres

(Release Reservation)

⚠️ This command is deprecated. Use mrsctl -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

<table>
<thead>
<tr>
<th>RESERVATION ID</th>
<th>Name of reservation to release.</th>
</tr>
</thead>
</table>

Example 1
Release two existing reservations.

> releaseres system.1 bob.2 released User reservation 'system.1' released User reservation 'bob.2'
**runjob**

*(Run Job)*

⚠️ 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**

<table>
<thead>
<tr>
<th>Args</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>Clear job parameters from previous runs (used to clear PBS neednodes attribute after PBS job launch failure)</td>
</tr>
<tr>
<td>-f</td>
<td>Attempt to force the job to run, ignoring throttling policies</td>
</tr>
<tr>
<td>-n &lt;NODELIST&gt;</td>
<td>Attempt to start the job using the specified <code>nodelist</code> where nodenames are comma or colon delimited</td>
</tr>
<tr>
<td>-p &lt;PARTITION&gt;</td>
<td>Attempt to start the job in the specified <code>partition</code></td>
</tr>
<tr>
<td>-s</td>
<td>Attempt to suspend the job</td>
</tr>
<tr>
<td>-x</td>
<td>Attempt to force the job to run, ignoring throttling policies, QoS constraints, and reservations</td>
</tr>
</tbody>
</table>

**Example**

This example attempts to 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
(Set Hold)

⚠️ 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

-   Set a batch hold. Typically, only the scheduler places batch holds. This flag allows an administrator to manually set a batch hold.
-   Help for this command.

Example 1

In this example, a batch hold is placed on job fr17n02.1072.0 and job fr15n03.1017.0.

> sethold -b fr17n02.1072.0 fr15n03.1017.0 Batch Hold Placed on All Specified Jobs
**setqos**

*(Set QoS)*

⚠️ This command is deprecated. Use `mjobctl -m` instead.

**Synopsis**

`setqos qosid jobid`

**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.

**Example 1**

This example sets the Quality Of Service to a value of `high_priority` for job `moab.3`.

> `setqos high_priority moab.3` Job QOS Adjusted
setres
(Set Reservation)

⚠️ This command is deprecated. Use mrsvctl -c instead.

Synopsis

setres [arguments] resourceexpression

Overview

Reserve resources for use by jobs with particular credentials or attributes.

ARGUMENTS:

[ -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> ] // (ie CLASS_LIST)
[ -Q <QOSLIST> ]
[ -r <RESOURCE_DESCRIPTION> ]
[ -R <RESERVATION_PROFILE> ]
[ -s <STARTTIME> ]
[ -T <TRIGGER> ]
[ -u <USER_LIST> ]
[ -x <FLAGS> ]

Access

This command can be run by level 1 and level 2 Moab administrators.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNT_LIST</td>
<td>&lt;STRING&gt;[:&lt;STRING&gt;]...</td>
<td>---</td>
<td>list of accounts that will be allowed access to the reserved resources</td>
</tr>
<tr>
<td>SUBTYPE</td>
<td>&lt;STRING&gt;</td>
<td>---</td>
<td>specify the subtype for a reservation</td>
</tr>
<tr>
<td>CHARGE_SPEC</td>
<td>&lt;ACCOUNT&gt;,&lt;GROUP&gt;,&lt;USER&gt;</td>
<td>---</td>
<td>specifies which credentials will be accountable for unused resources</td>
</tr>
<tr>
<td>CLASS_LIST</td>
<td>&lt;STRING&gt;[:&lt;STRING&gt;]...</td>
<td>---</td>
<td>list of classes that will be allowed access to the reserved resource</td>
</tr>
<tr>
<td>DURATION</td>
<td>[[[DD:]HH:]MM:]SS</td>
<td>INFINITY</td>
<td>duration of the reservation (not needed if ENDTIME is specified)</td>
</tr>
<tr>
<td>ENDTIME</td>
<td>[[HH[:MM[:SS]]][_MO[/DD[/YY]]] or +[[[DD:]HH:]MM:]SS</td>
<td>INFINITY</td>
<td>absolute or relative time reservation will end (not required if Duration specified)</td>
</tr>
<tr>
<td>EXCLUSIVE</td>
<td>N/A</td>
<td>N/A</td>
<td>requests exclusive access to resources</td>
</tr>
<tr>
<td>FEATURE_LIST</td>
<td>&lt;STRING&gt;[:&lt;STRING&gt;]...</td>
<td>---</td>
<td>list of node features which must be</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLAGS</td>
<td>list of reservation flags (See Managing Reservations for details)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP_LIST</td>
<td>list of groups that will be allowed access to the reserved resources</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>name set to first name listed in ACL or SYSTEM if no ACL specified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OWNER</td>
<td>specifies which credential is granted reservation ownership privileges</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARTITION</td>
<td>partition in which resources must be located</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QOS_LIST</td>
<td>list of QOS's that will be allowed access to the reserved resource</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESERVATION_PROFILE</td>
<td>requests that default reservation attributes be loaded from the specified reservation profile (see RSVPROFILE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESOURCE_DESCRIPTION</td>
<td>specifies the resources to be reserved. ALL indicates all resources should be reserved. Note: 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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESOURCE_EXPRESSION</td>
<td>Required Field. No Default</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STARTTIME</td>
<td>absolute or relative time reservation will start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIGGER</td>
<td>comma delimited reservation trigger list following format described in the trigger format section of the reservation configuration overview.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_LIST</td>
<td>list of users that will be allowed access to the reserved resources</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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.

**Example 1**

Reserve two nodes for use by users john and mary for a period of 8 hours starting in 24 hours.

```bash
> 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
```

**Example 2**

Schedule a system wide reservation to allow a system maintenance on Jun 20, 8:00 AM until Jun 22, 5:00 PM.

```bash
> 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
```

**Example 3**

Reserve one processor and 512 MB of memory on nodes node003 through node 006 for members of the group staff and jobs in the interactive class.

```bash
> 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
```
setspri
(Set System Priorities)

⚠️ This command is deprecated. Use mjobctl -p instead.

Synopsis

setspri [-r] priority jobid

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

<table>
<thead>
<tr>
<th>JOB</th>
<th>Name of job.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIORITY</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>If a relative priority is specified, any value in the range +/- 1,000,000,000 is acceptable.</td>
</tr>
</tbody>
</table>

Flags

- -r  Set relative system priority on job.

Example 1

In this example, a system priority of 10 is set for job orion.4752

```
> setspri 10 orion.4752
job system priority adjusted
```

Example 2

In this example, system priority is cleared for job clusterB.1102

```
> setspri 0 clusterB.1102
job system priority adjusted
```

Example 3
In this example, the job's priority will be increased by 100000 over the value determined by configured priority policy.

```bash
> setspri -r 100000 job.00001
job system priority adjusted
```

**Note:** This command is deprecated. Use `mjobctl` instead.
showconfig
(Show Configuration)

⚠️ This command is deprecated. Use mschedctl -l

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
- Help for this command.
  h
- 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.

Example 1
showconfig

> showconfig
# moab scheduler version 4.2.4 (PID: 11080)
BACKFILLPOLICY FIRSTFIT
BACKFILLMETRIC NODES
ALLOCATIONPOLICY MINRESOURCE
RESERVATIONPOLICY CURRENTHIGHEST
...

IMPORTANT Note: The showconfig flag 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.

See Also:
- Use the mschedctl -m command to change the various Moab Scheduler parameters.
- See the Parameters document for details about configurable parameters.
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 Grid Services API**
  - Provide and use information, data, job, and resource management services in a distributed environment.

- **H.6 Discovery/Directory Services**

- **H.7 Job Submission and Management Interface**
  - Query resource availability, submit, modify, and manage jobs, and query the status of active and completed jobs.

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 provide a (Moab) workload manager server that supports a broad array of client services. These services can either be directly accessed via Moab client commands or by way of a number of APIs. Which approach is best will depend on the particular use case.

- Java API
- C API
- CLI/XML API

**H.1.1 Java API**

The Moab-API is a Java library that allows Java developers to easily communicate with and interact with the Moab Workload Manager. The API handles the details of creating a secure connection, reporting information about workload and resources, and creating basic commands.

New users are encouraged to view the Moab Java API Quick Start Guide for details on interfacing with this API. The complete Javadocs are also available for review.

**H.1.2 C API**

The Moab C API provides access to all Moab information and services. This API consists of both easy to use high-level calls and powerful and flexible low-level calls.

**High-Level C API Calls**

- `free`
- `initialize`
- `jobcancel`
- `jobcheckpoint`
- `jobgetres`
- `jobgettime`
### MCCInitialize(Host, Port, &C)

**Args:**
- `Host`: (char *) optional server hostname or NULL
- `Port`: (int) optional server port or 0
- `C`: (void **) opaque interface handle

**Return Code:**
Returns 0 on success or -1 on failure

**Description:**
This call initializes the interface between the client and the server. It must be called once before attempting to issue any Moab client calls.

### MCCJobCancel(C, JID, EMsg)

**Args:**
- `C`: (void **) address of interface handle
- `JID`: (char *) job id to cancel (optional)
- `EMsg`: (char *) optional error message

**Return Code:**
Returns 0 on success or -1 on failure

**Description:**
This call will cancel the specified job. If failures occur and `EMsg` is provided, this buffer will be populated with a human-readable error message.

### MCCJobCheckpoint(C, JID, EMsg)

**Args:**
- `C`: (void **) address of interface handle
- `JID`: (char *) job id to cancel (optional)
- `EMsg`: (char *) optional error message

**Return Code:**
Returns 0 on success or -1 on failure

**Description:**
This call will checkpoint the specified job. If failures occur and `EMsg` is provided, this buffer will be populated with a human-readable error message.

### MCCJobGetNumAllocRes(C, JID, Count, EMsg)

**Args:**
- `C`: (void **) address of interface handle
- `JID`: (char *) job id to query (optional)
- `Count`: (int *) output value (number of allocated hosts)
- `EMsg`: (char *) optional error message

**Return Code:**
Returns 0 on success or -1 on failure

**Description:**
This call will return the number of allocated hosts for the specified job. If failures occur and `EMsg` is provided, this buffer will be populated with a human-readable error message.
### MCCJobGetRemainingTime(C,JID,Time,EMsg)

**Args:**
- C: (void **) address of interface handle
- JID: (char *) job id to query (optional)
- Time: (long *) output value (remaining time in seconds)
- EMsg: (char *) optional error message

**Return Code:**
Returns 0 on success or -1 on failure

**Description:**
This call will return the remaining wallclock duration/time for the specified job. If failures occur and EMsg is provided, this buffer will be populated with a human-readable error message.

**Note:** This call can be made to cache data for performance reasons by setting the MCCCACHETIME #define to the desired cache duration (in seconds) at build time.

### MCCJobListAllocRes(C,JID,OBuf,EMsg)

**Args:**
- C: (void **) address of interface handle
- JID: (char *) job id to query (optional)
- OBuf: (char **) output response string (comma delimited host list)
- EMsg: (char *) optional error message

**Return Code:**
Returns 0 on success or -1 on failure

**Description:**
This call will return a list of allocated hosts for the specified job. If failures occur and EMsg is provided, this buffer will be populated with a human-readable error message.

### MCCJobMigrate(C,JID,Dest,EMsg)

**Args:**
- C: (void **) address of interface handle
- JID: (char *) job id to migrate (optional)
- Dest: (char *) destination cluster or list of hosts (optional)
- EMsg: (char *) optional error message

**Return Code:**
Returns 0 on success or -1 on failure

**Description:**
This call will migrate the specified job. If failures occur and EMsg is provided, this buffer will be populated with a human-readable error message.

### MCCJobModify(C,JID,Attr,Val,Op,EMsg)

**Args:**
- C: (void **) address of interface handle
- JID: (char *) id of job to modify (optional - if not specified, modify job in current context)
- Attr: (char *) job attribute to modify (see mjobctl)
- Val: (char *) value to use when modifying specified attribute
- Op: (char *) operation to perform on attribute (one of set, unset, incr, or decr)
- EMsg: (char *) optional error message

**Return Code:**
Returns 0 on success or -1 on failure

**Description:**
This call will modify the attributes of either the specified job (if JID is populated) or the job in the current context (if JID is empty). If failures occur and EMsg is provided, this buffer will be populated with a human-readable error message.

### MCCJobQuery(C,JID,OBuf,EMsg)

**Args:**
- C: (void **) address of interface handle
- JID: (char *) job id to query (optional)
- OBuf: (char **) output response string (comma delimited host list)
- EMsg: (char *) optional error message
<table>
<thead>
<tr>
<th>Function</th>
<th>Arguments</th>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCCJobRequeue(C,JID,EMsg)</td>
<td>C: (void **) address of interface handle, JID: (char *) job id to requeue (optional), EMsg: (char *) optional error message</td>
<td>Returns 0 on success or -1 on failure</td>
<td>This call will requeue the specified job. If failures occur and EMsg is provided, this buffer will be populated with a human-readable error message.</td>
</tr>
<tr>
<td>MCCJobResume(C,JID,EMsg)</td>
<td>C: (void **) address of interface handle, JID: (char *) job id to resume (optional), EMsg: (char *) optional error message</td>
<td>Returns 0 on success or -1 on failure</td>
<td>This call will resume the specified job. If failures occur and EMsg is provided, this buffer will be populated with a human-readable error message.</td>
</tr>
<tr>
<td>MCCJobSignal(C,JID,Signo,EMsg)</td>
<td>C: (void **) address of interface handle, JID: (char *) job id to resume (optional), Signo: (int) job signal to send to job, EMsg: (char *) optional error message</td>
<td>Returns 0 on success or -1 on failure</td>
<td>This call will signal the specified job. If failures occur and EMsg is provided, this buffer will be populated with a human-readable error message.</td>
</tr>
<tr>
<td>MCCJobStart(C,JID,EMsg)</td>
<td>C: (void **) address of interface handle, JID: (char *) job id to start/run (optional), EMsg: (char *) optional error message</td>
<td>Returns 0 on success or -1 on failure</td>
<td>This call will immediately run/execute the specified job ignoring policies and dependencies provided that adequate idle resources are available. If failures occur and EMsg is provided, this buffer will be populated with a human-readable error message.</td>
</tr>
</tbody>
</table>

**Description:** This call will suspend the specified job. If failures occur and EMsg is provided, this buffer will be populated with a human-readable error message.
### MCCJobSubmit(C,JDesc,JID,EMsg)

**Args:**
- **C**: (void **) address of interface handle
- **JDesc**: (char *) XML job description (Scalable Systems Software Job Object Specification)
- **JID**: (char **) job id of new job (JID is populated on success only)
- **EMsg**: (char *) optional error message

**Return Code:**
- Returns **0** on success or **-1** on failure

**Description:**
This call will submit the specified job. If failures occur and **EMsg** is provided, this buffer will be populated with a human-readable error message. On success, memory is allocated for **JID**. **JID** is then populated with a new job's ID (which can then be used in calls such as **MCCJobQuery** and **MCCJobCancel**). The client is responsible for freeing the space allocated for **JID**.

### MCCJobSuspend(C,JID,EMsg)

**Args:**
- **C**: (void **) address of interface handle
- **JID**: (char *) job id to suspend (optional)
- **EMsg**: (char *) optional error message

**Return Code:**
- Returns **0** on success or **-1** on failure

**Description:**
This call will suspend the specified job. If failures occur and **EMsg** is provided, this buffer will be populated with a human-readable error message.

### MCCNodeQuery(C,NID,OBuf,EMsg)

**Args:**
- **C**: (void **) address of interface handle
- **NID**: (char *) node id to query (optional - if not specified, report all nodes)
- **OBuf**: (char **) output response string (reported in XML)
- **EMsg**: (char *) optional error message

**Return Code:**
- Returns **0** on success or **-1** on failure

**Description:**
This call will suspend the specified job. If failures occur and **EMsg** is provided, this buffer will be populated with a human-readable error message.

### MCCNodeModify(C,NID,Attr,Val,Op,EMsg)

**Args:**
- **C**: (void **) address of interface handle
- **NID**: (char *) id of node to modify (optional - if not specified, modify job in current context)
- **Attr**: (char *) node attribute to modify (see **mnodectl**) (optional)
- **Val**: (char *) value to use when modifying specified attribute
- **Op**: (char *) operation to perform on attribute (one of **set**, **unset**, **incr**, or **decr**)
- **EMsg**: (char *) optional error message

**Return Code:**
- Returns **0** on success or **-1** on failure

**Description:**
This call will modify the attributes of either the specified node (if **NID** is populated) or the local node (if **NID** is empty). If failures occur and **EMsg** is provided, this buffer will be populated with a human-readable error message.

### MCCFree(C)

<table>
<thead>
<tr>
<th>Args:</th>
<th>C: (void **) address of interface handle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>NID</strong>: (char *) id of node to modify (optional - if not specified, modify job in current context)</td>
</tr>
<tr>
<td></td>
<td><strong>Attr</strong>: (char *) node attribute to modify (see <strong>mnodectl</strong>) (optional)</td>
</tr>
<tr>
<td></td>
<td><strong>Val</strong>: (char *) value to use when modifying specified attribute</td>
</tr>
<tr>
<td></td>
<td><strong>Op</strong>: (char *) operation to perform on attribute (one of <strong>set</strong>, <strong>unset</strong>, <strong>incr</strong>, or <strong>decr</strong>)</td>
</tr>
<tr>
<td></td>
<td><strong>EMsg</strong>: (char *) optional error message</td>
</tr>
<tr>
<td></td>
<td>Returns <strong>0</strong> on success or <strong>-1</strong> on failure</td>
</tr>
</tbody>
</table>

**Description:**
This call will modify the attributes of either the specified node (if **NID** is populated) or the local node (if **NID** is empty). If failures occur and **EMsg** is provided, this buffer will be populated with a human-readable error message.
Args: C: (void **) address of interface handle

Return Code:
Returns 0 on success or -1 on failure

Description: This call will free allocated interface handle memory

Example 1 - Interfacing to an Explicitly Specified Moab Server

```c
#include <stdio.h>
#include <stdlib.h>
#include "mapi.h"

int main()
{
    int   ArgC,
    char *ArgV[],
    char *EnvP[])

    { void *C;
    char  EMsg[1024];
    char *OBuf;

    char JobID[32];

    char *JobDescription = "<Job><Arguments>600</Arguments>
    <Executable>/bin/sleep</Executable><InitialWorkingDirectory>/home/wig
    <NodeCount>1</NodeCount><GroupId>wightman</GroupId><UserId>wightman</UserId><Requested></Requested><Processors>1</Processors></Job>";

    if (MCCInitialize("sn-master03",14522,&C) < 0)
        {
```

Example 2 - Interface to Monitor Remaining Time w/in a Batch Job

```c
#include <stdio.h>
#include <stdlib.h>
#include "mapi.h"

int main()
{
    int   ArgC,
    char *ArgV[],
    char *EnvP[])

    { mulong Time;

        /* from inside a batch job, API will auto allocate the interface 
           and discover the appropriate jobid */

        /* monitor time, clean-up and exit if less then 300 seconds remain */

        while (1)
        {
            MCCJobGetRemainingTime(NULL,NULL,&Time,NULL);
```
Low-Level C API

The Moab low-level C API allows direct access to all Moab client services. This API uses the **MCCExecute** routine and takes as an argument an XML request string. This interface provides a high-speed interface directly into the server and provides very broad and flexible cluster, workload, and resource control. This routine creates a direct socket connection to provides the following services:

- socket creation
- client authentication
- server discovery via built-in, configfile, and environment settings
- command request and response encryption
- marshalling and de-marshalling of requests and data

Requests are sent using native XML request strings and, depending on the command used, responses are returned in unformatted native XML or pretty-print human-readable format. The commands that do not return XML can be configured to return XML with the **--format=xml** option. The commands that do not natively return XML are:

- setspri
- setres
- releaseres
- runjob
- checkjob
- showconfig
-mdiag
- mjobctl

Use of Cluster Resources Consulting Services is recommended for organizations using this interface.

### MCCInitialize(Host,Port,&C)

**Args:**
- Host: (char *) optional server hostname or NULL
- Port: (int) optional server port or 0
- C: (void **) opaque interface handle

**Return Code:**
- Returns 0 on success or -1 on failure

**Description:**
This call initializes the interface between the client and the server. It should be called once before attempting to issue any Moab client calls if connecting to a server at a non-default location. If connecting to the default server, client commands will automatically initialize the interface handle.

### MCCExecute(C,RCmd,RBuf,&OBuf,EMsg)

**Args:**
- C: (void **) address of interface handle
- RCmd: (char *) request client command
- RBuf: (char *) request command XML
- OBuf: (char **) output response string
- EMsg: (char *) optional error message

**Return:**
- Returns 0 on success or -1 on failure
H.1.3 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:

```plaintext
> showq --format=xml
<Data>
<Object>queue</Object>
<cluster LocalActiveNodes="1" LocalAllocProcs="1" LocalIdleNodes="0"
  LocalIdleProcs="3" LocalUpNodes="1"
  RemoteActiveNodes="0" RemoteAllocProcs="0"
  RemoteIdleNodes="0" RemoteIdleProcs="0"
  RemoteUpNodes="0" RemoteUpProcs="0" time="1128451812"></cluster>
<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>
<job 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:

- **HTTP**
- **LDAP**
- **SQL**
- **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.5 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 for Grids for more information on provided services.

### H.6 Discovery/Directory Services

Moab can import and export key event information regarding workload, cluster resources, cluster and grid services, and other components of hardware and software infrastructure.

By default, these clients communicate with the scheduler using the U.S. Department of Energy (DOE) **Scalable Systems Software** socket and wire protocols. These protocols are largely HTML- and XML-based, using PKI, 3DES, MD5, challenge, and other security protocols and are documented within the SSS project pages.

As part of this initiative, the scheduler/client protocol has been extended to support multiple socket level protocol standards in communicating with its clients and peer services. These include SingleUseTCP, SSS-HALF, and HTTP. The client socket protocol can be specified by setting the **MCSOCKETPROTOCOL** parameter to **SUTCP**, **SSS-HALF**, or **HTTP**. Further protocols are being defined and standardized over time and backwards compatibility will be maintained. Documentation on the SSS-HALF implementation can be found within the DOE’s **SSS Project Notebooks**.

### H.7 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
Moab Java API Quick Start Guide

Introduction

The Moab Java API is a Java library that allows Java developers to easily communicate with and interact with the Moab Workload Manager. The API handles the details of creating a secure connection, reporting information about workload and resources, and creating basic commands.

Throughout this guide there are simple examples on how to use the various classes. The full code for these examples is included in the API in the com.ace.moab.example package. Please review these files for more detailed information on the various classes.

Also, note that the client is the program that will use the Moab Java API.

Requirements

This guide is intended to show how to begin using the Moab Java API for any Java based application. It assumes users have a basic knowledge of Moab and its commands. It is recommended that users consult other reference and tutorial resources for information about Moab and its use and purpose.

Basic Usage

The communication process from the client's point of view can be simplified in the following 3-step process:

1. Client creates a secure connection to the Moab Workload Manager.
2. Client queries Moab for system information such as jobs or reservations.
3. Client may create commands to manipulate these objects or create new commands in Moab.

The purpose of this API is to simplify each of the three steps in this process.

Note: In the documentation, request and command may be used interchangeably when speaking about a request to the Moab server.

Create a Connection to the Moab Workload Manager

In this API, there are two types of supported connections to the Moab Workload Manager:

- Local connections for instances of Moab running on the local machine. (See LocalConnection.java for more information.)
- SSH connections for instances of Moab running on any other host. (See SSHConnection.java for more information.)

Both of the connection types implement the same interface that provides the necessary functions to communicate with Moab. All Moab queries require an IMoabConnection object to execute commands and return the results. Typically, there only needs to be one IMoabConnection per Java application.

The IMoabConnection object allows users to not only create a connection, but to also run commands over this connection. However, in most cases the client will rarely have to call the executeCommand method directly; this will be called by the various query commands explained in the next section.

Example SSH Connection

```java
//Setup connection with the server called "myServer"
SSHConnection connection = new SSHConnection("myServer", 22);
//Attempt to connected with user bob. Returns true if successful.
boolean result = connection.connectWithPassword("bob", password);
```

Querying Moab
As previously stated, the Moab API uses commands to communicate with the Moab Workload Manager, and it parses the server’s response. The client runs any necessary command and gets the response through the MoabInputStream object. However, this API greatly simplifies the process of querying Moab using the various Query objects. These objects create the appropriate commands, sends them over the connection, parses the response from Moab, and returns to the caller the appropriate Java objects.

**Job Queries (JobQuery.java)**

There are five basic job queries in this version of the API:

- **getActiveJobs**: Returns a list of all the jobs currently active—not completed.
- **getCompletedJobs**: Returns all completed jobs still left in Moab's completed job buffer.
- **getCompleteJobSummary**: Gets the job summary information for all users from Moab.
- **getSpecificJob**: Gets a specific job from Moab.
- **getUserJobSummary**: Gets the job summary information for a certain user.

A job in Moab is represented in the API as a MoabJob object. For more information on these classes, please review the Java documentation under the JobQuery and MoabJob classes.

**Node (or Server) Queries (NodeQuery.java)**

There are four node queries in this version of the API:

- **getAllNodes**: Returns all nodes reported to Moab.
- **getUserSpecificNodes**: Returns all the nodes to which the given user has access.
- **getSpecificNode**: Gets a specific node from Moab.
- **getNodeSummary**: Returns a node summary either for all nodes or just the nodes accessible by a specified user.

A node or server in Moab is represented in the API as a MoabNode object. For more information on these classes, please review the Java documentation under the NodeQuery and MoabNode classes.

**Reservation Queries (ReservationQuery.java)**

There are three basic reservation queries in this version of the API:

- **GetReservationByName**: Returns the reservation corresponding to a specific reservation ID.
- **getAllReservations**: Queries Moab for all of the reservations in the system.
- **getStandingReservations**: Returns a list of all standing reservations in Moab.

A reservation in Moab is represented in the API as a Reservation object. Standing reservations (also known as reoccurring reservations) are represented as a StandingReservation object. For more information on these classes, please review the Java documentation under the ReservationQuery, Reservation, and StandingReservation classes.

**Credential Queries (CredentialQuery.java)**

There are two credential queries in this version of the API:

- **getAllCredentialsOfType**: Returns a list of all credentials of a given type (such as user or account).
- **getAllCredentials**: Returns a list of all standard credentials that this user can see.

All credential types in Moab are represented in the API as a Credential object. This includes users, groups, accounts, classes, and quality of service (QoS) objects. For more information on these classes, please review the Java documentation under the CredentialQuery and Credential classes.

**Sending Requests to Modify or Create Objects in Moab**

The Moab API not only enables Java clients to query Moab and parse the results, but it also allows clients the option to create simple Java objects and then use these objects to create identical objects in Moab. This greatly simplifies the process of submitting jobs and reservations as clients do not need to determine the Moab commands necessary to create objects inside Moab. Instead, clients create simple Java objects and then use the Request interface to get the commands necessary to create these same objects in Moab. For
example, a client may create a **MoabJob** object with a command file, a node list, and certain credentials specified. Instead of determining the Moab commands necessary to duplicate this job's attributes for submission, the client can use the **SubmitJobRequest** object to retrieve the commands necessary to submit this job to Moab.

As of this version of the API, the two supported objects that use the **Request** interface are jobs and reservations. For jobs, clients should use **SubmitJobRequest** for job submission and **ModifyJobRequest** to modify an existing job in Moab. For reservations, clients may use the **ModifyReservationRequest**, **CreateOneTimeRsvRequest**, and **CreateStandingRsvRequest** objects.

**Submit Job Request (SubmitJobRequest.java)**

The SubmitJobRequest object uses data from a MoabJob object to create a job submit command. Clients create a MoabJob object, populate this object with various attributes to be set at job submission, then use the Request interface to get the commands to submit the job. However, not all attributes inside the MoabJob object make sense for job submission. For example, a MoabJob contains an attribute called "suspendDuration" that represents the number of seconds a job is suspended. Obviously a user cannot submit a job with a suspend duration greater than 0 as this does not make sense. Likewise setting a job's state at submission doesn't make sense as Moab determines job state.

The following are supported attributes for the submit job request:

- Generic resources: type, quantity, and (optionally) the time frame
- Job dependencies
- Specified credentials (such as Account or QoS requested)
- Email notifications
- File operations (such as specifying the stdout file)
- Host list
- Job flags
- Job requirements including:
  - Processors per task
  - Node features (both required and preferred)
  - Operating system
  - Architecture
  - Node memory
  - Node swap space
  - Partition
  - Required reservation
- Job name
- Initial working directory
- User priority
- Template list
- Requested wall time
- Earliest start date requested

**Example Submit Job Request**

```java
//Create the job as a Java object
MoabJob job = new MoabJob();

//Set the various attributes for the Job upon submission
job.setJobName("myFirstJob");
job.setCommandFile("/path/to/command/file");
job.setAccount("Engineering");
job.setWallclockRequested(18000); //Job should run for 5 hours
job.setNodeCountRequested(15);

SubmitJobRequest request = new SubmitJobRequest(job);

List<String> moabCommands = request.getMoabCommands();
//Execute moabCommands
...```
Modify Job Request (ModifyJobRequest.java)

There are many ways to modify an existing Moab job. Other than the job attributes like account or wall time, jobs can be canceled, suspended, and re-queued. For a complete list of modification types, see the ModifyJobType enum.

The modify job request is like the submit job request: clients manipulate Java objects and then use these objects to create Moab commands necessary to cause these changes. Many of the job modifications only require one MoabJob object (such as cancel, resume, or checkpoint). However, to modify job attributes, two separate MoabJob objects must be created. The ModifyJobRequest class will then find the differences between these two objects and determine the Moab commands necessary to modify the first object to have the same attributes as the second.

Example Modify Job Request

```java
//Load the job from Moab
MoabJob newJob = JobQuery.getSpecificJob(...);
MoabJob oldJob = JobQuery.getSpecificJob(...);
newJob.setNodeCountRequested(10); //Set a different node count here

ModifyJobRequest request = new ModifyJobRequest(originalJob, newJob,
                                          ModifyJobType.JOB_ATTRIBUTES, moabConnection);

List<String> moabCommands = request.getMoabCommands();
//The moabCommands now contains the command to modify the node count to 10.
```

Like the submit job request, many job attributes cannot be modified using this class. The following is a list of supported attributes for the modify job request:

- Job holds
- Credential changes (such as Account or QoS)
- Job name
- Adding messages (but not removing messages)
- Requested node count
- System priority
- Requested reservation
- The job's partition access list
- Variables
- Requested wall time

Create Reservation Request (CreateOneTimeRsvRequest.java, CreateStandingRsvRequest.java)

In Moab, there are two types of reservations: (1) one-time reservations and (2) reoccurring (or standing) reservations. Because of the core differences between these reservation types, the Moab Java API has split these reservations into separate objects, Reservation and StandingReservation. The request classes used to create Moab commands for these objects are also different classes, but they are used the same way.

The attributes that can be set on a Reservation or StandingReservation object that will be used for reservation creation are listed below:

- All reservation requirements including
  - Task specification (and task count)
  - Architecture
  - Any node features
  - Network
  - Operating system
  - Node count
Host list
- Reservation owner
- The DEDICATEDRESOURCE flag
- Partition
- Start and end dates
- Trigger list
- Host list expression
- An access control list (ACL)

A valid reservation must have a host list expression specified or a positive number of tasks assigned to it. If task count is positive, the required task must be specified as well. Clients may verify a valid reservation before attempting to create the Moab commands by calling the `verifyReservation` method on the respective create reservation request object.

**Modify Reservation Request (ModifyReservationRequest.java)**

Modifying a reservation is very similar to the way clients may modify a job in Moab. The client passes two reservations into the ModifyReservationRequest object, the original and the reservation which contains the modifications. The request then compares the differences between the two reservations and creates Moab commands based on the differences between the reservations.

There are a limited number of attributes that can be modified after a reservation is created. The following is a list of supported attribute changes included in the API:

- Reservation duration
- Start time
- End time
- Any flags set (including the ability to unset flags)
- Host list expression

For more information, please review the Moab documentation about modifying reservation attributes.

**Example Modify Job Request**

```java
//Load the reservation from Moab
Reservation originalRsv = ReservationQuery.getReservationByName(...);
Reservation modifiedRsv = ReservationQuery.getReservationByName(...);

//Change the flags and the start date
ReservationFlags flags = modifiedRsv.getFlags();
flags.removeAll();
flags.addFlag(ReservationFlag.ADVRES);
flags.addFlag(ReservationFlag.ALLOWPRSV);
modifiedRsv.setStartDate(new Date(modifiedRsv.getStartDate().getTime() + 360000));

//Create the modification request and get the commands
ModifyReservationRequest request = new ModifyReservationRequest(originalRsv, modifiedRsv);
List<String> moabCommands = request.getMoabCommands();
```

**Additional Notes**

All classes released in the API are either in the com.ace.moab.* package or the com.moab.* package. All classes and packages in the com.ace.* package structure are a standard part of the Moab Java API and will be supported in the future. All other packages outside of com.ace can change at any time without notice and are not officially supported. These classes are included in the API only because they provide some "behind the scenes" functionality for various methods. Clients that use these other packages must understand they are using code that may not be tested and may not function correctly. For these reasons, there is no documentation available for the com.moab.* classes.
The examples included in this guide are simplified versions of examples included in the `com.ace.moab.example` package. They will not necessarily compile "as is" and may require additional setup such as establishing an `IMoabConnection` or surrounding method calls in a try catch block. Please view the code examples included in the API for more details.
Appendix I: Considerations for Large Clusters

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

Aggregating Scheduling Cycles - JOBAGGREGATIONTIME

With event driven resource manager interfaces (such as TORQUE, PBS, and SGE), each time a job is submitted, the resource manager notifies the scheduler of this fact. In an attempt to minimize response time, the scheduler will start a new scheduling cycle to determine if the newly submitted job can run. In systems with large numbers of jobs submitted at once, this may not result in the desired behavior for two reasons. First, by scheduling at every job submission, Moab will schedule 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 may place 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 would contact the scheduler, the scheduler would start a new iteration, and in this iteration, the scheduler would contact the resource manager requesting updated information on all jobs and resources available.

The JOBAGGREGATIONTIME parameter works by informing the scheduler to not process jobs as quickly as they are submitted, but rather to process these new jobs in groups.

Limited Job Checkpointing - LIMITEDJOBCP

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.

Reducing Job Start Time - RMCFG ASYNCSTART flag value.

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.

Reducing Job Reservation Creation Time - RMCFG JOBRSVRECREATE 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.

**Minimizing Compute Intensive Operations - ENABLESTARTESTIMATESTATS**

Where possible, these parameters should be disabled as they are expensive per job operations.

**Buffering Log Output - MOABENABLELOGBUFFERING environment variable**

When large or verbose logs are required, setting this environment variable to true will allow Moab to buffer its logs and speed up log writing. This capability is primarily useful when writing to remote file systems and is only of limited value with local file systems.

**Constraining Preemption - PREEMPTSEARCHDEPTH parameter**

When a large number of active serial jobs are present in a system, Moab may unnecessarily consider additional jobs even after an adequate number of feasible preemptible jobs have been located. Setting this parameter will cause Moab to cease its search after the specified number of target preemptees has been located.

**Note**: Setting this parameter only impacts searches for serial preemtior jobs.

**Handling Transient Resource Manager Failures - MOABMAXRMFAILCOUNT=<INTEGER>**

**Disable Job Feasibility Analysis - MOABDISABLEFEASIBILITYCHECK**

Constrain the number of jobs started per iteration - JOBMAXSTARTPERITERATION parameter. (Some resource managers can take in excess of two seconds per job start.) Because Moab must serialize job launch, a system where many jobs are started each iteration may appear sluggish from the point of view of client commands. Setting this parameter will reduce the maximum duration of a scheduling cycle and thus the maximum duration a client command will wait for processing.

**Constrain the number of jobs preempted per iteration - JOBMAXPREEMPTPERITERATION parameter**

**Note**: 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.

### 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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Recommended Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMPOLLINTERVAL</td>
<td>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.</td>
</tr>
<tr>
<td>LIMITEDNODECP</td>
<td>Startup/shutdown time can be minimized by disabling full node state checkpointing that includes some statistics covering node availability.</td>
</tr>
<tr>
<td>--with-maxnodes</td>
<td>(special builds only - contact support) *</td>
</tr>
<tr>
<td>--with-maxtasks</td>
<td>(special builds only - contact support) *</td>
</tr>
</tbody>
</table>

* 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.

Other considerations include using grid based resource reporting when using peers and enabling virtual nodes to collapse scheduling decisions.
**Note:** See Appendix D for further information on default and supported object limits.

### 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Recommended Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXRSVPERNODE</td>
<td>By default, Moab does not expect more than 24 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.</td>
</tr>
<tr>
<td>--with-maxrange</td>
<td>(special builds only - contact support)</td>
</tr>
</tbody>
</table>

### I.6 Server Sizing

See Hardware and Software Requirements for recommendations.

**See Also**

- 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 Maui 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 (`/opt/moab`) 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 MOABHOMEDIR=[make directory]`.
2. To have Moab use all the same policies as Maui, copy `maui.cfg` to the MOABHOMEDIR and rename it `moab.cfg`.
   o You can also start your `moab.cfg` file from scratch. Just use the `moab.cfg` already in the MOABHOMEDIR.
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`.
   o 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:

<table>
<thead>
<tr>
<th>File</th>
<th>Maui</th>
<th>Moab</th>
</tr>
</thead>
<tbody>
<tr>
<td>executable</td>
<td>maui</td>
<td>moab</td>
</tr>
<tr>
<td>logs</td>
<td>maui.log</td>
<td>moab.log</td>
</tr>
<tr>
<td>configuration file</td>
<td>maui.cfg</td>
<td>moab.cfg</td>
</tr>
</tbody>
</table>

**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 Specific 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:

<table>
<thead>
<tr>
<th>Maui</th>
<th>Moab</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUIHOMEDIR</td>
<td>MOABHOMEDIR</td>
</tr>
<tr>
<td>MAUIDEBUG</td>
<td>MOABDEBUG</td>
</tr>
<tr>
<td>MAUCRASHVARIABLE</td>
<td>MOABCRASHVARIABLE</td>
</tr>
<tr>
<td>MAUICRASHVARIABLE</td>
<td>MOABCRASHVARIABLE</td>
</tr>
<tr>
<td>MAUICRASHVARIABLE</td>
<td>MOABCRASHVARIABLE</td>
</tr>
<tr>
<td>MAUICRASHVARIABLE</td>
<td>MOABCRASHVARIABLE</td>
</tr>
<tr>
<td>MAUIAMTEST</td>
<td>MOABAMTEST</td>
</tr>
<tr>
<td>MAUI-COMMANDS-PATH</td>
<td>MOAB-COMMANDS-PATH</td>
</tr>
<tr>
<td>MAUICRASHVARIABLE</td>
<td>MOABCRASHVARIABLE</td>
</tr>
<tr>
<td>MAUICRASHVARIABLE</td>
<td>MOABCRASHVARIABLE</td>
</tr>
<tr>
<td>MAUICRASHVARIABLE</td>
<td>MOABCRASHVARIABLE</td>
</tr>
<tr>
<td>MAUICRASHVARIABLE</td>
<td>MOABCRASHVARIABLE</td>
</tr>
<tr>
<td>MAUICRASHVARIABLE</td>
<td>MOABCRASHVARIABLE</td>
</tr>
</tbody>
</table>
Appendix O: 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.

O.1 Compute Resource Managers

- WIKI - WIKI Integration Guide
- Cray XT/Torque - Integration Guide (html, pdf), http://www.cray.com

O.2 Provisioning Resource Managers

- xCAT - Validating an xCAT Installation for Use with Moab
- xCAT - Integrating an xCAT Physical Provisioning Resource Manager with Moab
- SystemImager - Enabling Moab Provisioning with SystemImager

O.3 Hardware Integration

- NUMA - Integration Guide
Moab-Loadleveler Integration Guide

Overview

Moab can be used as an external scheduler for Loadleveler. In this configuration, Loadleveler manages the job queue and the compute resources while Moab queries the Loadleveler negotiator via the Loadleveler data API to obtain up to date job and node information. Using this information, Moab directs Loadleveler to manage jobs in accordance with specified Moab policies, priorities, and reservations.

LoadLeveler Configuration

Moab drives LL via the Loadleveler scheduling API. To enable this API and thus the external scheduler, the following steps must be taken:

- set 'SCHEDULER_API=yes' in the 'LoadL_config' file typically located in the user 'loadl' home directory.
- set the 'NEGOTIATOR_REMOVE_COMPLETED' parameter (also located in the 'LoadL_config' file) to a value of at least 5 minutes, ie 'NEGOTIATOR_REMOVE_COMPLETED=300'. (This allows Moab to obtain job info from LL required to maintain accurate job statistics)
- AGGREGATE_ADAPTERS should be set to NO in the LoadL_config file.
- recycle negotiator using the command 'llctl recycle' on the central manager node.

Moab Configuration

To enable Moab to communicate with the Loadleveler negotiator daemon, the RMCFG parameter must be set with a TYPE of LL. By default, this should already be done for you automatically by the configure script.

Because only a subset of LoadLeveler command file keywords can be interpreted by Moab, the parameter INSTANTSTAGE should be used when jobs are submitted through msub.

Issues

The Loadleveler scheduling API is not event driven so Moab has no way of knowing when a new job is submitted. Under these conditions, it will not evaluate a newly submitted job until its next scheduling iteration, typically within 15 to 30 seconds. This lag can be removed by utilizing Loadleveler's 'SUBMITFILTER'. The Moab command mschedctl -r 2 can be added as the last statement in this filter causing Moab to 'wake-up' and attempt to schedule new jobs immediately. The mschedctl command is an administrative command and so may need an suid wrapper in order to allow use by non-privileged users. (see example).

Note: Do NOT use the above submit filter when jobs will be submitted using msub or via Moab Access Portal.

Note: You can return to Loadleveler default scheduling at any time by setting 'SCHEDULER_API=no' in the LoadL_config file and re-issuing the 'llctl recycle' command.

Moab supports interactive job hostlists but these hostlists must currently be specified using the network interface Loadleveler utilizes. For example, an SP node may have two names, node001e and node001sw representing its ethernet and switch interfaces respectively. Loadleveler is configured to communicate with the nodes on one of these interfaces. (This can be determined by issuing 'llstatus' and observing the name used to specify each node.) Interactive job hostlists must be specified using the same interface that Loadleveler is configured to use. Efforts are underway to extend Moab interface tracking to remedy this.

Note: The LoadLeveler API is not thread safe, therefore, do not build Moab with __MTHREAD enabled.

Note: Some releases of Loadleveler will requeue all active jobs when reconfigured to use the external scheduler interface. In such cases, it may be best to drain the queue before enabling Moab.

Note: If using Loadleveler with Moab Access Portal or with a Moab Peer Based Grid, the parameter INSTANTSTAGE must be set.
Moab-TORQUE/PBS Integration Guide

- 1.0 Overview
- 2.0 Integration Steps
  - 2.1 Install TORQUE/PBS
  - 2.2 Install Moab
  - 2.3 Configure TORQUE/PBS
  - 2.4 Configure Moab
- 3.0 Current Limitations
- 4.0 Trouble-shooting

1.0 Overview

Moab can be used as an external scheduler for the PBS resource management system. In this configuration, PBS manages the job queue and the compute resources while Moab queries the PBS Server and the PBS MOM's to obtain up to date job and node information. Using this information, Moab directs PBS to manage jobs in accordance with specified Moab policies, priorities, and reservations.

2.0 Integration Steps

Moab manages PBS via the PBS scheduling API. The steps below describe the process for enabling Moab scheduling using this API.

2.1 Install TORQUE/PBS

- Install TORQUE/PBS

* Keep track of the PBS target directory, $PBSTARGDIR

2.2 Install Moab

- Untar the Moab distribution file.
- Change the directory to the moab-<X> 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).

2.3 General Configuration For All Versions of TORQUE/PBS

- 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:

```
moab.cfg

> 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>)

2.3.1 Version-Specific Configuration for TORQUE, OpenPBS or PBSPro 6.x or earlier

Do not start the pbs_sched daemon. This is the default scheduler for PBS/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.

2.3.2 Version-Specific Configuration for PBSPro 7.1 and higher

PBSPro 7.x, 8.x, and higher require that the pbs_sched daemon execute for proper operation, but PBS must be configured to take no independent action that conflicts with Moab. With these PBSPro releases, sites should allow pbs_sched to run after putting the following PBS configuration in place:

```
qmgr configuration

> qmgr -c 'set server scheduling = false'
> qmgr -c 'set server scheduler_iteration = 100000000'
> qmgr -c 'unset server node_fail_requeue'
```

sched_priv/sched_config

```
preemptive_sched: false ALL
```

2.4 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.

3.0 Current Limitations
**PBS Features Not Supported by Moab**

Moab supports basic scheduling of all PBS node specifications.

Moab is, by default, liberal in its interpretation of `<NODECOUNT>:PPN=<X>`. In its standard configuration, Moab interprets this as 'give the job `<NODECOUNT>`*<X> tasks with AT LEAST `<X>` tasks per node'. Set the `JOBNODEMATCHPOLICY` parameter to **EXACTNODE** to have Moab support PBS's default allocation behavior of `<NODECOUNT>` nodes with exactly `<X>` tasks per node.

**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.

**Some Versions of PBS Do Not Maintain Job Completion Information**

An external scheduler cannot determine if the job completed successfully or if internal PBS problems occurred preventing the job from being properly updated. This problem will not in any way affect proper scheduling of jobs but may potentially affect scheduler statistics. If your site is prone to frequent PBS **hangs**, you may want to set the Moab **JOBPURGETIME** parameter to allow Moab to hold job information in memory for a period of time until PBS recovers. (Note: It is not recommended that **PURGETIME** be set to over 2:00).

**4.0 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**.

```bash
> qmgr Qmgr: set server managers += <MOABADMIN>@*.<YOURDOMAIN> Qmgr: set server operators += <MOABADMIN>@*.<YOURDOMAIN> Qmgr: quit
```

For example:

```bash
> 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:

```bash
$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:

```bash
> 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:

```bash
> 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:

```bash
> qmgr
Qmgr: set server resources_default.nodect = 1
Qmgr: set server resources_default.walltime = 00:15:00
Qmgr: quit
```
Moab-SGE Integration Notes

Copyright © 2011 Adaptive Computing Enterprises, Inc.

This document provides information on the steps to integrate Moab with an existing functional installation of SGE.

Notice

Distribution of this document for commercial purposes in either hard or soft copy form is strictly prohibited without prior written consent from Adaptive Computing Enterprises, Inc.

Overview

Moab's native resource manager interface can be used to manage an SGE resource manager. The integration steps simply involve the creation of a complex variable and a default request definition. The Moab tools directory contains a collection of customizable scripts which are used to interact with sge. This directory also contains a configuration file for the sge tools.

Moab Integration Steps

You should follow the regular steps for installing Moab with the following exceptions:

Run Configure with the --with-sge option

When running the configure command, use the --with-sge option to specify the use of the native resource manager interface with the sge resource manager subtype. This will place a line similar to the following in the Moab configuration file (moab.cfg):

```
RMCFG[clusternamex] TYPE=NATIVE:sge
```

Example 1. Running configure

```
$ ./configure --prefix=/opt/moab --with-homedir=/var/moab --with-sge
```

Customize the moab configuration file

In order to allow the specification of a parallel environment (-l pe) via msub, you will need to tell Moab to pass through arbitrary resource types.

Example 2. Edit moab.cfg

```
# vi /var/moab/moab.cfg
# Transmit arbitrary resource types (ie. pe) from msub into the job-start script
CLIENTCFG[Moab] FLAGS=AllowUnknownResource
# Allow regular users to awaken the scheduler for responsive msubs
ADMINCFG[5] USERS=ALL SERVICES=mschedctl:resume
```

Customize the sge tools configuration file

You may need to customize the $MOABHOMEDIR/etc/config.sge.pl file to include the correct SGE_ROOT and PATH, and set other configuration parameters.

Example 3. Edit config.sge.pl

```
# vi /var/moab/etc/config.sge.pl
# Set the SGE_ROOT environment variable
$ENV{SGE_ROOT} = "/opt/sge-root";
# Set the PATH to include directories for sge commands -- qhost, etc.
$ENV{PATH} = "$ENV{SGE_ROOT}:/bin/1x24-x86:$ENV{PATH}";
```
SGE Integration Steps

After installing SGE on your cluster and verifying that it is running serial and parallel jobs satisfactorily, you should perform the following steps:

Define a new complex variable named nodelist

Use the qconf -mc command to edit the complex variable list and add a new requestable variable of the name nodelist and the type RESTRING.

```
# qconf -mc
nodelist          nodelist     RESTRING    ==    YES         NO         NONE  0
```

Add a default nodelist request definition

This step will set the nodelist complex variable for all jobs to the unassigned state until they are ready to run, at which time the job will be assigned a nodelist directing which nodes it can run on.

Example 4. Edit sge_request

```
# vi /opt/sge-root/default/common/sge_request
# Set the job's nodelist variable to the unassigned state until it is ready to
# start at which time it will be reset to the list of nodes it is designated to
# run on
-l nodelist=unassigned
```

Populate the node's nodelist variable

This step will set the nodelist complex variable for all exec hosts to their own short hostnames. This will allow jobs to start when their nodelist value matches up with a set of nodes.

Example 5. qconf -rattr exechost complex_values nodelist=$hostname $hostname

```
# for i in `qconf -sel | sed 's/..*//'; do echo $i; qconf -rattr exechost complex_values
nodelist=$i $i; done
```

Shorten the scheduler interval

Use the qconf -msconf command to edit the schedule_interval setting to be less than or equal to one half the time of the Moab RMPOLLINTERVAL (seen with showconfig | grep RMPOLLINTERVAL).

```
# qconf -msconf
schedule_interval                 0:0:15
```

Add the sge ports to the services file

In order for the sge client commands to know what port to use when communicating with the sge qmaster, the ports should be listed in the /etc/services file. (Alternatively, the SGE_QMASTER_PORT environment variable must be set in the config.sge.pl file).

Example 6. Edit /etc/services

```
# vi /etc/services
sge_qmaster     536/tcp                 # SGE QMaster
sge_execd       537/tcp                 # SGE Execd
```
Moab-SLURM Integration Guide

- S.1 Overview
- S.2 SLURM Configuration Steps
- S.3 Moab Configuration Steps
  - S.3.1 Configuration for Standby and Expedite
  - S.3.2 Configuration for the Quadrics Switch
  - S.3.3 Authentication
  - S.3.4 Queue/Class Support
  - S.3.5 Policies
  - S.3.6 Moab Queue and RM Emulation
  - S.3.7 SLURM High Availability

S.1 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.

S.2 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.

S.3 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

```plaintext
RMCFG[base] TYPE=WIKI:SLURM SERVER=Node01 VERSION=10124
...
```

### S.3.1 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:

```plaintext
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:

```plaintext
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
...
```

### S.3.2 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
S.3.3 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.

S.3.4 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.

S.3.5 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 parameter '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
```

S.3.6 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.

S.3.7 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 'mdiaq -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.

### See Also

- SLURM Admin Manual
- SLURM's Moab Integration Guide
- Additional SLURM Documentation
- Wiki Overview
Wiki Interface Overview

- Wiki Interface
- Socket Level Interface
- Configuring Wiki
Appendix W: Wiki Interface Specification, version 1.2

W.1.1 Commands

- W.1.1.1 Resource Query
  - W.1.1.1.1 Query Resources Request Format
  - W.1.1.1.2 Query Resources Response Format
  - W.1.1.1.3 Query Resources Example
  - W.1.1.1.4 Query Resources Data Format
- W.1.1.2 Workload Query
  - W.1.1.2.1 Query Workload Request Format
  - W.1.1.2.2 Query Workload Response Format
  - W.1.1.2.3 Query Workload Example
  - W.1.1.2.4 Query Workload Data Format
- W.1.1.3 Start Job
- W.1.1.4 Cancel Job
- W.1.1.5 Suspend Job
- W.1.1.6 Resume Job
- W.1.1.7 Requeue Job
- W.1.1.8 Signal Job
- W.1.1.9 Modify Job
- W.1.1.10 JobAddTask
- W.1.1.11 JobRemoveTask
- W.1.1.12 JobAddTask
- W.1.1.13 JobRemoveTask

W.1.2 Rejection Codes

All commands are requested via a socket interface, one command per socket connection. All fields and values are specified in ASCII text. Moab is configured to communicate via the wiki interface by specifying the following parameters in the moab.cfg file:

```
moab.cfg
```

Field values must backslash escape the following characters if specified:

' # ; : (i.e. \#)

Supported Commands are:

- Query Resources
- Query Workload
- Start Job
- Cancel Job
- Suspend Job
- Resume Job
- Requeue Job
- JOBADDTASK
- JOBRELEASETASK

W.1.1 COMMANDS

All commands are requested via a socket interface, one command per socket connection. All fields and values are specified in ASCII text. Moab is configured to communicate via the wiki interface by specifying the following parameters in the moab.cfg file:

```
moab.cfg
```

Field values must backslash escape the following characters if specified:

' # ; : (i.e. \#)

Supported Commands are:

- Query Resources
- Query Workload
- Start Job
- Cancel Job
- Suspend Job
- Resume Job
- Requeue Job
- JOBADDTASK
- JOBRELEASETASK

W.1.1.1 Wiki Query Resources

W.1.1.1.1 Wiki 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.1.1.1.2 Query Resources Response Format

The query resources response format is one or more line of the following format (separated with a newline, " "):

```
<NODEID> <ATTR>=<VALUE>[;<ATTR>=<VALUE>]...
```

<ATTR> is one of the names in the table below and the format of <VALUE> is dependent on <ATTR>.

W.1.1.1.3 Wiki Query Resources Example

request:

```
wiki resource query
```

```
### Wiki Query Resources Data Format

<table>
<thead>
<tr>
<th>NAME</th>
<th>FORMAT</th>
<th>DEFAULT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACLASS</td>
<td>one or more bracket enclosed &lt;NAME&gt;:&lt;COUNT&gt; pairs (ie, [batch:5][sge:3])</td>
<td>---</td>
<td>run classes currently available on node. If not specified, scheduler will attempt to determine actual ACLASS value.</td>
</tr>
<tr>
<td>ADISK</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>available local disk on node (in MB)</td>
</tr>
<tr>
<td>AFS</td>
<td>&lt;fs id=&quot;X&quot; size=&quot;Y&quot; io=&quot;Z&quot; rcount=&quot;x&quot; wcount=&quot;y&quot; ocount=&quot;z&quot;/&gt;&lt;/fs&gt;</td>
<td>0</td>
<td>available filesystem state</td>
</tr>
<tr>
<td>AMEMORY</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>available/free RAM on node (in MB)</td>
</tr>
<tr>
<td>ANET</td>
<td>one or more colon delimited &lt;STRING&gt;'s (ie, ETHER:ATM)</td>
<td>---</td>
<td>Available network interfaces on node. Available interfaces are those which are 'up' and not already dedicated to a job.</td>
</tr>
<tr>
<td>APROC</td>
<td>&lt;INTEGER&gt;</td>
<td>1</td>
<td>available processors on node</td>
</tr>
<tr>
<td>ARCH</td>
<td>&lt;STRING&gt;</td>
<td>---</td>
<td>compute architecture of node</td>
</tr>
<tr>
<td>ARES</td>
<td>one or more comma delimited &lt;NAME&gt;:&lt;VALUE&gt; pairs (ie, MATLAB:6,COMPILER:100)</td>
<td>---</td>
<td>Arbitrary consumable resources currently available on the node</td>
</tr>
<tr>
<td>ASWAP</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>available swap on node (in MB)</td>
</tr>
<tr>
<td>CCLASS</td>
<td>one or more bracket enclosed &lt;NAME&gt;:&lt;COUNT&gt; pairs (ie, [batch:5][sge:3])</td>
<td>---</td>
<td>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]</td>
</tr>
<tr>
<td>CDISK</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>configured local disk on node (in MB)</td>
</tr>
<tr>
<td>CFS</td>
<td>&lt;STRING&gt;</td>
<td>0</td>
<td>configured filesystem state</td>
</tr>
<tr>
<td>CHMEMORY</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>configured RAM on node (in MB)</td>
</tr>
<tr>
<td>CNET</td>
<td>one or more colon delimited &lt;STRING&gt;'s (ie, ETHER:FDDI:ATM)</td>
<td>---</td>
<td>configured network interfaces on node</td>
</tr>
<tr>
<td>CPROC</td>
<td>&lt;INTEGER&gt;</td>
<td>1</td>
<td>configured processors on node</td>
</tr>
<tr>
<td>CPULOAD</td>
<td>&lt;DOUBLE&gt;</td>
<td>0.0</td>
<td>one minute BSD load average</td>
</tr>
<tr>
<td>CRES</td>
<td>one or more comma delimited &lt;NAME&gt;:&lt;VALUE&gt; pairs (ie, MATLAB:6,COMPILER:100)</td>
<td>---</td>
<td>Arbitrary consumable resources supported and tracked on the node, ie software licenses or tape drives.</td>
</tr>
<tr>
<td>CSWAP</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>configured swap on node (in MB)</td>
</tr>
<tr>
<td>CURRENTTASK</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Number of tasks currently active on the node</td>
</tr>
<tr>
<td>EVENT</td>
<td>&lt;STRING&gt;</td>
<td>---</td>
<td>Event or exception which occurred on the node</td>
</tr>
<tr>
<td>FEATURE</td>
<td>one or more colon delimited &lt;STRING&gt;'s (ie, WIDE:HSM)</td>
<td>---</td>
<td>generic attributes, often describing hardware or software features, associated with the node.</td>
</tr>
<tr>
<td>GCOUNTER</td>
<td>&lt;INTEGER&gt;</td>
<td>---</td>
<td>current total number of gevent event occurrences since epoch. This value should be monotonically increasing.</td>
</tr>
<tr>
<td>GEVENT</td>
<td>GEVENT[&lt;EVENTNAME&gt;]=&lt;STRING&gt;</td>
<td>---</td>
<td>generic event occurrence and context data.</td>
</tr>
<tr>
<td>GMETRIC</td>
<td>GMETRIC[&lt;METRICNAME&gt;]=&lt;DOUBLE&gt;</td>
<td>---</td>
<td>current value of generic metric, i.e., 'GMETRIC[temp]=103.5'.</td>
</tr>
<tr>
<td>IDLETIME</td>
<td>&lt;INTEGER&gt;</td>
<td>---</td>
<td>number of seconds since last detected keyboard or mouse activity (often used with desktop harvesting)</td>
</tr>
<tr>
<td>MAXTASK</td>
<td>&lt;INTEGER&gt;</td>
<td>&lt;CPROC&gt;</td>
<td>Maximum number of tasks allowed on the node at any given time</td>
</tr>
<tr>
<td>OS</td>
<td>&lt;STRING&gt;</td>
<td>---</td>
<td>operating system running on node</td>
</tr>
<tr>
<td>OSLIST</td>
<td>&lt;STRING&gt;</td>
<td>---</td>
<td>operating systems accepted by node</td>
</tr>
<tr>
<td>OTHER</td>
<td>&lt;ATTR&gt;=&lt;VALUE&gt;[, &lt;ATTR&gt;=&lt;VALUE&gt;][,...]</td>
<td>---</td>
<td>opaque node attributes assigned to node</td>
</tr>
<tr>
<td>PARTITION</td>
<td>&lt;STRING&gt;[, &lt;ATTR&gt;=&lt;VALUE&gt;][,...]</td>
<td>DEFAULT</td>
<td>partition to which node belongs</td>
</tr>
<tr>
<td>RACK</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Rack location of the node</td>
</tr>
<tr>
<td>SLOT</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>Slot location of the node</td>
</tr>
<tr>
<td>SPEED</td>
<td>&lt;DOUBLE&gt;</td>
<td>1.0</td>
<td>Relative processor speed of the node</td>
</tr>
<tr>
<td>STATE*</td>
<td>one of the following: Idle, Running, Busy, Unknown, Drained, Draining, or Down Down</td>
<td>state of the node</td>
<td></td>
</tr>
<tr>
<td>UPDATETIME*</td>
<td>&lt;EPOCHTIME&gt;</td>
<td>0</td>
<td>time node information was last updated</td>
</tr>
<tr>
<td>VARIABLE*</td>
<td>&lt;ATTR&gt;=&lt;VAL&gt;</td>
<td>---</td>
<td>generic variables to be associated with node</td>
</tr>
</tbody>
</table>

* indicates required field

Note: node states have the following definitions:

- **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.1.1.2 Wiki Query Workload

W.1.1.2.1 Wiki Query Workload Request Format

CMD=GETJOBS ARG=\{<UPDATETIME>:<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.1.1.2.2 Wiki Query Workload Response Format

SC=<STATUSCODE>
ARG=<JOBCOUNT>#<JOBID>:<FIELD>=<VALUE>;[<FIELD>=<VALUE>;]...[#<JOBID>:<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.1.1.2.3 Wiki Query Workload Example

request syntax

CMD=GETJOBS ARG=0:ALL

response syntax

ARG=2#nebo3001.0:UPDATETIME=9780000320;STATE=Idle;WCLIMIT=3600;...

W.1.1.2.4 Wiki Query Workload Data Format

<table>
<thead>
<tr>
<th>NAME</th>
<th>FORMAT</th>
<th>DEFAULT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNT</td>
<td>&lt;STRING&gt;</td>
<td>---</td>
<td>AccountID associated with job</td>
</tr>
<tr>
<td>ALLOCSIZE</td>
<td>&lt;INTEGER&gt;</td>
<td>---</td>
<td>number of application tasks to allocate at each allocation adjustment.</td>
</tr>
<tr>
<td>APPBACKLOG</td>
<td>&lt;DOUBLE&gt;</td>
<td>---</td>
<td>backlogged quantity of workload for associated application (units are opaque), value may be compared against TARGETBACKLOG</td>
</tr>
<tr>
<td>APPLOAD</td>
<td>&lt;DOUBLE&gt;</td>
<td>---</td>
<td>load of workload for associated application (units are opaque), value may be compared against TARGETLOAD</td>
</tr>
<tr>
<td>APPRESPONSETIME</td>
<td>&lt;DOUBLE&gt;</td>
<td>---</td>
<td>response time of workload for associated application (units are opaque), value may be compared against TARGETRESPONSETIME</td>
</tr>
<tr>
<td>APPTHROUGHPUT</td>
<td>&lt;DOUBLE&gt;</td>
<td>---</td>
<td>throughput of workload for associated application (units are opaque), value may be compared against TARGETTHROUGHPUT</td>
</tr>
<tr>
<td>ARGS</td>
<td>&lt;STRING&gt;</td>
<td>---</td>
<td>job command-line arguments</td>
</tr>
<tr>
<td>COMMENT</td>
<td>&lt;STRING&gt;</td>
<td>0</td>
<td>job resource manager extension arguments including qos, dependencies, reservation constraints, etc</td>
</tr>
<tr>
<td>COMPLETETIME*</td>
<td>&lt;EPOCHTIME&gt;</td>
<td>0</td>
<td>time job completed execution</td>
</tr>
<tr>
<td>DDISK</td>
<td>&lt;INTEGER&gt;</td>
<td>0</td>
<td>quantity of local disk space (in MB) which must be dedicated to each task of the job</td>
</tr>
<tr>
<td>DGRES</td>
<td>name:value[,name:value]</td>
<td>---</td>
<td>Dedicated generic resources per task.</td>
</tr>
<tr>
<td>DPROCS</td>
<td>&lt;INTEGER&gt;</td>
<td>1</td>
<td>number of processors dedicated per task</td>
</tr>
<tr>
<td>DNETWORK</td>
<td>&lt;STRING&gt;</td>
<td>---</td>
<td>network adapter which must be dedicated to job</td>
</tr>
<tr>
<td><strong>DSWAP</strong></td>
<td><code>&lt;INTEGER&gt;</code></td>
<td>0</td>
<td>quantity of virtual memory (swap, in MB) which must be dedicated to each task of the job</td>
</tr>
<tr>
<td><strong>ENDDATE</strong></td>
<td><code>&lt;EPOCHTIME&gt;</code></td>
<td>[ANY]</td>
<td>time by which job must complete</td>
</tr>
<tr>
<td><strong>ENV</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>job environment variables</td>
</tr>
<tr>
<td><strong>EVENT</strong></td>
<td><code>&lt;EVENT&gt;</code></td>
<td>---</td>
<td>event or exception experienced by job</td>
</tr>
<tr>
<td><strong>ERROR</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>file to contain STDERR</td>
</tr>
<tr>
<td><strong>EXEC</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>job executable command</td>
</tr>
<tr>
<td><strong>EXITCODE</strong></td>
<td><code>&lt;INTEGER&gt;</code></td>
<td>---</td>
<td>job exit code</td>
</tr>
<tr>
<td><strong>FLAGS</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>job flags</td>
</tr>
<tr>
<td><strong>GEOMETRY</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>String describing task geometry required by job</td>
</tr>
<tr>
<td><strong>GNAME</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>GroupID under which job will run</td>
</tr>
<tr>
<td><strong>HOSTLIST</strong></td>
<td>comma or colon delimited list of hostnames - suffix the hostlist with a caret (^) to mean superset; suffix with an asterisk (*) to mean subset; otherwise, the hostlist is interpreted as an exact set</td>
<td>[ANY]</td>
<td>list of required hosts on which job must run. (see TASKLIST)</td>
</tr>
<tr>
<td><strong>INPUT</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>file containing STDIN</td>
</tr>
<tr>
<td><strong>IWD</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>job's initial working directory</td>
</tr>
<tr>
<td><strong>NAME</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>User specified name of job</td>
</tr>
<tr>
<td><strong>NODERANGE</strong></td>
<td><code>&lt;INTEGER&gt;[, &lt;INTEGER&gt;]</code></td>
<td>---</td>
<td>Minimum and maximum nodes allowed to be allocated to job.</td>
</tr>
<tr>
<td><strong>NODES</strong></td>
<td><code>&lt;INTEGER&gt;</code></td>
<td>1</td>
<td>Number of nodes required by job (See Node Definition for more info)</td>
</tr>
<tr>
<td><strong>OUTPUT</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>file to contain STDOUT</td>
</tr>
<tr>
<td><strong>PARTITIONMASK</strong></td>
<td>one or more colon delimited <code>&lt;STRING&gt;</code>s</td>
<td>[ANY]</td>
<td>list of partitions in which job can run</td>
</tr>
<tr>
<td><strong>PREF</strong></td>
<td>colon delimited list of <code>&lt;STRING&gt;</code>s</td>
<td>---</td>
<td>List of preferred node features or variables. (See PREF for more information.)</td>
</tr>
<tr>
<td><strong>PRIORITY</strong></td>
<td><code>&lt;INTEGER&gt;</code></td>
<td>---</td>
<td>system priority (absolute or relative - use '+' and '-' to specify relative)</td>
</tr>
<tr>
<td><strong>QOS</strong></td>
<td><code>&lt;INTEGER&gt;</code></td>
<td>0</td>
<td>quality of service requested</td>
</tr>
<tr>
<td><strong>QUEUETIME</strong></td>
<td><code>&lt;EPOCHTIME&gt;</code></td>
<td>0</td>
<td>time job was submitted to resource manager</td>
</tr>
<tr>
<td><strong>RARCH</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>architecture required by job</td>
</tr>
<tr>
<td><strong>RCLASS</strong></td>
<td>list of bracket enclosed <code>&lt;STRING&gt;</code>: <code>&lt;INTEGER&gt;</code> pairs</td>
<td>---</td>
<td>list of of <code>&lt;CLASSNAME&gt;</code>: <code>&lt;COUNT&gt;</code> pairs indicating type and number of class instances required per task. (ie, <code>[batch:1]</code> or <code>[batch:2][tape:1]</code>)</td>
</tr>
<tr>
<td><strong>RDISK</strong></td>
<td><code>&lt;INTEGER&gt;</code></td>
<td>0</td>
<td>local disk space (in MB) required to be configured on nodes allocated to the job</td>
</tr>
<tr>
<td><strong>RDISKCMP</strong></td>
<td>one of <code>&gt;=</code>, <code>&gt;</code>, <code>==</code>, <code>&lt;=</code></td>
<td><code>&gt;=</code></td>
<td>local disk comparison (ie, node must have &gt;= 2048 MB local disk)</td>
</tr>
<tr>
<td><strong>REJCODE</strong></td>
<td><code>&lt;INTEGER&gt;</code></td>
<td>0</td>
<td>reason job was rejected</td>
</tr>
<tr>
<td><strong>REJCOUNT</strong></td>
<td><code>&lt;INTEGER&gt;</code></td>
<td>0</td>
<td>number of times job was rejected</td>
</tr>
<tr>
<td><strong>REJMESSAGE</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>text description of reason job was rejected</td>
</tr>
<tr>
<td><strong>REQRSV</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>Name of reservation in which job must run</td>
</tr>
<tr>
<td><strong>RESACCESS</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>List of reservations in which job can run</td>
</tr>
<tr>
<td><strong>RFEATURES</strong></td>
<td>colon delimited list <code>&lt;STRING&gt;</code>s</td>
<td>---</td>
<td>List of features required on nodes</td>
</tr>
<tr>
<td><strong>RMEM</strong></td>
<td><code>&lt;INTEGER&gt;</code></td>
<td>0</td>
<td>real memory (RAM, in MB) required to be configured on nodes allocated to the job</td>
</tr>
<tr>
<td><strong>RMEMCMP</strong></td>
<td>one of <code>&gt;=</code>, <code>&gt;</code>, <code>==</code>, <code>&lt;=</code></td>
<td><code>&gt;=</code></td>
<td>real memory comparison (ie, node must have &gt;= 512MB RAM)</td>
</tr>
<tr>
<td><strong>RNETWORK</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>network adapter required by job</td>
</tr>
<tr>
<td><strong>ROPSYS</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>operating system required by job</td>
</tr>
<tr>
<td><strong>RSOFTWARE</strong></td>
<td>`&lt;RESTYPE&gt;[{+</td>
<td>:}&lt;COUNT&gt;[@&lt;TIMEFRAME&gt;]]`</td>
<td>---</td>
</tr>
<tr>
<td><strong>RSWAP</strong></td>
<td><code>&lt;INTEGER&gt;</code></td>
<td>0</td>
<td>virtual memory (swap, in MB) required to be configured on nodes allocated to the job</td>
</tr>
<tr>
<td><strong>RSWAPCMP</strong></td>
<td>one of <code>&gt;=</code>, <code>&gt;</code>, <code>==</code>, <code>&lt;=</code></td>
<td><code>&gt;=</code></td>
<td>virtual memory comparison (ie, node must have == 4096 MB virtual memory)</td>
</tr>
<tr>
<td><strong>SID</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>system id (global job system owner)</td>
</tr>
<tr>
<td><strong>SJJID</strong></td>
<td><code>&lt;STRING&gt;</code></td>
<td>---</td>
<td>system job id (global job id)</td>
</tr>
<tr>
<td><strong>STARTDATE</strong></td>
<td><code>&lt;EPOCHTIME&gt;</code></td>
<td>0</td>
<td>earliest time job should be allowed to start</td>
</tr>
<tr>
<td><strong>STARTTIME</strong></td>
<td><code>&lt;EPOCHTIME&gt;</code></td>
<td>0</td>
<td>time job was started by the resource manager</td>
</tr>
<tr>
<td><strong>STATE</strong></td>
<td></td>
<td>one of Idle, Running, Hold, Suspended, Completed, or Removed</td>
<td>Idle</td>
</tr>
<tr>
<td><strong>SUSPENDTIME</strong></td>
<td><code>&lt;INTEGER&gt;</code></td>
<td>0</td>
<td>Number of seconds job has been suspended</td>
</tr>
</tbody>
</table>
### TARGETBACKLOG

<DOUBLE>[,<DOUBLE>] --- Minimum and maximum backlog for application within job.

### TARGETLOAD

<DOUBLE>[,<DOUBLE>] --- Minimum and maximum load for application within job.

### TARGETRESPONSETIME

<DOUBLE>[,<DOUBLE>] --- Minimum and maximum response time for application within job.

### TARGETTHROUGHPUT

<DOUBLE>[,<DOUBLE>] --- Minimum and maximum throughput for application within job.

### TARGETVIOLATIONTIME

<ALLOCATIONTIME>[,<DEALLOCATIONTIME>] where values are specified using the format [[DD:]HH:]MM:]SS

By default, Moab allocates/deallocates resources as soon as a performance target violation is detected.

### TASKLIST

one or more comma-delimited <STRING>'s

By default, Moab allocates/deallocates resources as soon as a performance target violation is detected.

### 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 064000 walltime required by job

---

### Note:

Job states have the following definitions:

- **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

### Note:

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.

---

#### 1.1.3 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.

```plaintext
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

job start example

```

# 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'
```

#### 1.1.4 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'.

```plaintext
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

job cancel example

```bash
# Cancel job nebo.2
send 'CMD=CANCELJOB ARG=nebo.2 TYPE=ADMIN'
receive 'SC=0 RESPONSE=job nebo.2 canceled'
```

### 1.1.5 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.

```bash
send CMD=SUSPENDJOB 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
```

job suspend example

```bash
# Suspend job nebo.3
send 'CMD=SUSPENDJOB ARG=nebo.3'
receive 'SC=0 RESPONSE=job nebo.3 suspended'
```

### 1.1.6 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.

```bash
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
```

job resume example

```bash
# Resume job nebo.3
send 'CMD=RESUMEJOB ARG=nebo.3'
receive 'SC=0 RESPONSE=job nebo.3 resumed'
```

### 1.1.7 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.

```bash
send CMD=REQUEUETYPE JOB 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
```

job requeue example

```bash
# Requeue job nebo.3
send 'CMD=REQUEUETYPE JOB ARG=nebo.3'
receive 'SC=0 RESPONSE=job nebo.3 requeued'
```

### 1.1.8 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 signalled job will remain in the same state it was before the signal was issued.

```bash
send CMD=SIGNA ::= JOB ID 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

job signal example

```bash
# Signal job nebo.3
send 'CMD=SIGNALJOB ARG=nebo.3 ACTION=signal VALUE=13'
receive 'SC=0 RESPONSE=job nebo.3 signalled'
```

1.1.9 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
```

job modify example

```bash
# Signal job nebo.3
send 'CMD=MODIFYJOB ARG=nebo.3 TIMELIMIT=9600'
receive 'SC=0 RESPONSE=job nebo.3 modified'
```

1.1.10 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
```

job addtask example

```bash
# 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'
```

1.1.11 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
```

job removetask example

```bash
# 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'
```
1.2 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
  - 350 - 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
    - 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
  - 410 - general negotiation failure
    - 412 - negotiation request malformed
    - 414 - negotiation request not understood
    - 416 - negotiation request not supported
  - 420 - general authentication failure
    - 422 - client signature failure
    - 424 - server authentication failure
    - 426 - server signature failure
    - 428 - client authentication failure
  - 430 - general encryption failure
    - 432 - client encryption failure
    - 434 - server decryption failure
    - 436 - server encryption failure
    - 438 - client decryption failure
  - 440 - general authorization failure
    - 442 - client authorization failure
    - 444 - server authorization failure
  - 450 - component defined failure
  - 480 - application defined failure
  - 5xx - event management request failure
  - 500 - reserved
  - 500 - reserved
  - 6xx - reserved for future use
  - 600 - reserved
  - 7xx - server side error occurred
    - 700 - server side error
      - 700 - general server side error
    - 710 - server does not support requested function
    - 720 - general internal server error
    - 730 - general resource unavailable error
    - 732 - software resource unavailable error
    - 734 - hardware resource unavailable error
    - 740 - general policy violation
    - 750 - component-defined failure
    - 780 - application-defined failure
  - 8xx - client side error occurred
    - 800 - general client side error
      - 800 - general client side error
    - 810 - request not supported
    - 820 - application specific failure
  - 9xx - miscellaneous
    - 900 - general miscellaneous error
      - 900 - general miscellaneous error
    - 910 - general insufficient resources error
      - 910 - general insufficient resources error
    - 990 - general unknown error
      - 999 - unknown error
Wiki Socket Protocol Description

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>
```

- `<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.
- `<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.

Checksum Algorithm ('C' version)

```c
#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 short Buf[1]);
```

*Checksum Algorithm ('C' version)*
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;
    ittmpl = 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);
}

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);
$thisto=$thisport=pack($sockaddr, &AF_INET,0,$thisaddr);
$thatport=pack($sockaddr, &AF_INET,$BANKPORT,$thataddr);
socket(S, &PF_INET,&SOCK_STREAM,$proto) || die "cannot create socket\n";
bind(S,$thisto) || 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=SAUTH";
chomp($CHECKSUM = `$QSUM "$REQUEST"`);
$REQUEST .= " CHECKSUM=$CHECKSUM";
my $command=pack "a8 ai A",sprintf("%08d",length($REQUEST))," ",trecht;
print S "COMMAND";       # Send Command to server
@REPLY=();
while (<S>) { 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;
}

Header Processing (PERL code)

sysread(NS,$length,8);           # Read length string
sysread(NS,$delimiter,1);        # Read delimiter byte
$DEBUG && print STDERR "length=[$length]	delimiter=[$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"`);
$sme=$REQUEST{AUTH};
$command=$REQUEST{COMMAND};
if (!grep($command eg $, @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$delimiter$REPLY\n";
$buf="$len "$delim"$REPLY";
syswrite(NS,$buf,length($buf));
close NS;
Wiki Configuration

Configuring the WIKI interface requires setting the following attributes of the RMCFG parameter to be specified.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTHTYPE</td>
<td>CHECKSUM</td>
<td></td>
</tr>
<tr>
<td>SERVER</td>
<td>&lt;URL&gt;</td>
<td>Must be specified</td>
</tr>
<tr>
<td>TYPE</td>
<td>WIKI</td>
<td></td>
</tr>
</tbody>
</table>
Moab-LSF Integration Guide

Overview

Moab can be used as an external scheduler for LSF and significantly increase the cluster’s capabilities in terms of cluster management, grid integration, accounting, and optimization services. In this configuration, LSF manages the job queue and the compute resources while Moab queries the LSF daemons via the LSF scheduling API (only available in LSF 5.1 and higher) to obtain up-to-date job and node information. Using this information, Moab directs LSF to manage jobs in accordance with specified Moab policies, priorities, and reservations.

Installing Moab

To install Moab for LSF, from the Moab source code root directory, source the LSF environment file `$LSF_ENVDIR/lsf.conf`, run `configure`, and then execute `make install`.

Note: With LSF 5.x and earlier, use $LSF_CONFDIR in place of $LSF_ENVDIR.

Configuring Moab

The `configure` command should have already properly configured Moab to use LSF. However, to verify this, edit the moab.cfg file and confirm that the following line is specified:

```
moab.cfg
```

The resource manager flag `ignqueuestate` informs Moab to utilize queues which are disabled for usage by the LSF scheduler.

Note: Moab can pull information directly from the LSF API or can obtain it indirectly by parsing the output of LSF commands. See LSF Integration via the Native Interface for more information.

Configuring LSF

Moab must be run as an LSF admin user. To enable a user as an LSF admin, edit the ClusterAdmins stanza in the `$LSF_CONFDIR/lsf.cluster.<CLUSTER>` file as in the example below:

```
conf/lsf.cluster.<CLUSTER>
```

```
Begin   ClusterAdmins
Administrators = lsfadmin john steve
End    ClusterAdmins
```

Additionally, LSF must be configured to use Moab as its external scheduler. This can be done by adding the following line to the `lsf.conf` file and restarting the LSF head-node daemons:

```
lsf.conf
```

```
LSF_ENABLE_EXTSCHEDULER=Y
```

Disabling LSF Scheduler

To disable LSF scheduling for all queues, issue the following command:

```
disable LSF queues
```

```
> badmin qinact all
```
**Starting Moab**

To start Moab, source the LSF environment file, `lsf.conf` into the current environment and then start the Moab daemon:

```
starting moab
> source /usr/lsf/conf/lsf.conf
> moab
```

> `moab`d is a safe and recommended method of starting Moab if things are not installed in their default locations.

> It may also be necessary to add the path to your LSF libraries to the LD_LIBRARY_PATH variable as in `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lsf/6.0/linux2.4-glibc2.3-x86/lib`.

**Troubleshooting**

- For LSF to function correctly, the **mbatchd** daemon must be running. If Moab reports 'batch system daemon not responding ... still trying', check this process.
moab.cfg

```bash
-----
# moab.cfg

# Using the native interface to run query commands.
RMCFG[native] TYPE=NATIVE FLAGS=queueisprivate
RMCFG[native] CLUSTERQUERYURL=/usr/local/etc/node.query.lsf.pl
TIMEOUT=30
RMCFG[native] PARTITION=lsf

# Only use some features from the LSF (non-native) interface.
RMCFG[lsf] TYPE=LSF FLAGS=executionserver,ignorequeuestate
RMCFG[lsf] FNLIST=queuequery,workloadquery,jobstart
-----
```

node.query.lsf.pl - script to provide information about LSF nodes

```perl
#!/usr/bin/perl
use strict;

# Extract node data from LSF and pass it to Moab.

# Note: Should pay attention to command line,
#       but currently doesn't.

# Note: Other node information can be passed along,
#       for now this is the bare minimum required.

my $command = "bhosts -w |";
my $current_time = time();
my %job_state;
my $num_nodes;
my $node_id;
my $state;

open (DATA, $command);
for (<DATA>)
{
    if (/^HOST_NAME/) {
        # NO-OP
    }
    elsif (/(\S+)\s+(\S+)/) {
        $job_state{$1} = $2;
    }
```
Installation Notes for Moab and Torque on the Cray XT

Copyright © 2011 Adaptive Computing Enterprises, Inc.

This document provides information on the steps to install Moab and Torque on a Cray XT system.

Overview

Moab and Torque can be used to manage the batch system for a Cray XT4, XT5 or later supercomputers. This document describes how Moab can be configured to use Torque and Moab's native resource manager interface to bring Moab's unmatched scheduling capabilities to the Cray XT4/XT5.

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.

Torque Installation Notes

Perform the following steps from the boot node as root:

Download the latest Torque release

Download the latest Torque release.

Example 1. Download Torque

```
# cd /rr/current/software
# wget http://www.adaptivecomputing.com/downloads/torque/torque-2.2.0.tar.gz
```

Unpack the Torque tarball in an xtopview session

Using xtopview, unpack the Torque tarball into the software directory in the shared root.

Example 2. Unpack Torque

```
# xtopview
default:/# cd /software
default/:/software # tar -zxvf torque-2.2.0.tar.gz
```

Configure Torque

While still in xtopview, run configure with the options set appropriately for your installation. Run ./configure — help to see a list of configure options. Adaptive Computing recommends installing the Torque binaries into /opt/torque/$version and establishing a symbolic link to it from /opt/torque/default. At a minimum, you will need to specify the hostname where the Torque server will run (--with-default-server) if it is different from the host it is being compiled on. The Torque server host will normally be the SDB node for XT installations.

Example 3. Run configure

```
default/:/software # cd torque-2.2.0
default/:/software/torque-2.2.0 # ./configure --
```
**Note:** The `--enable-maxdefault` is a change from Torque 2.4.5 onwards. This will enforce max_default queue and server settings the same way previous versions of Torque did by default. Your site may choose not to follow this configuration setting and get the new behavior. See [Job Submission](#) for more information.

**Compile and Install Torque**

While still in xtopview, compile and install Torque into the shared root. You may also need to link /opt/torque/default to this installation. Exit xtopview.

**Example 4. Make and Make Install**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default/:/software/torque-2.2.0 # make</td>
<td>Compile Torque</td>
</tr>
<tr>
<td>default/:/software/torque-2.2.0 # make packages</td>
<td>Build packages</td>
</tr>
<tr>
<td>default/:/software/torque-2.2.0 # make install</td>
<td>Install Torque</td>
</tr>
<tr>
<td>default/:/software/torque-2.2.0 # ln -sf /opt/torque/2.2.0/</td>
<td>Link to shared root</td>
</tr>
<tr>
<td>default/:/software/torque-2.2.0 # exit</td>
<td>Exit xtopview</td>
</tr>
</tbody>
</table>

**Copy your Torque server directory to your Moab server host**

In this example we assume the Torque server will be running on the SDB node. Torque's home directory on the SDB will be /var/spool/torque which is mounted from the bootnode (persistent var). The SDB is usually nid00003 but you will need to confirm this by logging into the SDB and running `cat /proc/cray_xt/nid`. Use the numeric nodeid from this command in the following example.

**Example 5. On the boot node, copy the Torque home directory to the SDB node's persistent /var filesystem (as exported from the bootnode)**

```bash
# cd /rr/current/var/spool
# cp -pr torque /snv/3/var/spool
```

**Stage out MOM dirs to login nodes**

Stage out the MOM dirs and client server info on all login nodes. This example assumes you are using a persistent /var filesystems mounted from /snv on the boot node. Alternatively, a ram var filesystem must be populated by a skeleton tarball on the bootnode (/rr/current/.shared/var-skel.tgz) into which these files must be added. The example below assumes that you have 3 login nodes with nids of 4, 64 and 68. Place the hostname of the SDB node in the server_name file.

**Example 6. Copy out MOM dirs and client server info**

```bash
# cd /rr/current/software/torque-2.2.0/tpackages/mom/var/spool
# for i in 4 64 68
  > do cp -pr torque /snv/$i/var/spool
  > echo $i/var/spool/torque/server_name
  > # Uncomment the following if userids are not resolvable from the pbs_server host
  > # echo "QSUBSENDUID true" > /snv/$i/var/spool/torque/torque.cfg
  > done
```

**Perform the following steps from the Torque server node (sdb) as root:**

```bash
prefix=/opt/torque/2.2.0 --with-server-home=/var/spool/torque --with-default-server=sdb --enable-syslog --disable-gcc-warnings --enable-maxdefault --with-modulefiles=/opt/modulefiles
```
Setup the Torque server on the sdb node

Configure the Torque server by informing it of its hostname and running the Torque.setup script.

Example 7. Set the server name and run torque.setup

```bash
# hostname > /var/spool/torque/server_name
# export PATH=/opt/torque/default/sbin:/opt/torque/default/bin:$PATH
# cd /software/torque-2.2.0
# ./torque.setup root
```

Customize the server parameters

Add access and submit permission from your login nodes. You will need to enable host access by setting acl_host_enable to true and adding the nid hostnames of your login nodes to acl_hosts. In order to be able to submit from these same login nodes, you need to add them as submit_hosts and this time use their hostnames as returned from the hostname command.

Example 8. Customize server settings

Enable scheduling to allow Torque events to be sent to Moab. Note: If this is not set, Moab will automatically set it on startup.

```bash
# qmgr -c "set server scheduling = true"
```

Keep information about completed jobs around for a time so that Moab can detect and record their completion status. Note: If this is not set, Moab will automatically set it on startup.

```bash
# qmgr -c "set server keep_completed = 300"
```

Remove the default nodes setting

```bash
# qmgr -c "unset queue batch resources_default.nodes"
```

Set resources_available.nodes equal to the maximum number of procs that can be requested in a job.

```bash
# qmgr -c "set server resources_available.nodes = 1250"
```

Do this for each queue individually as well.

```bash
# qmgr -c "set queue batch resources_available.nodes = 1250"
```

Only allow jobs submitted from hosts specified by the acl_hosts parameter.

```bash
# qmgr -c "set server acl_host_enable = true"
# qmgr -c "set server acl_hosts += nid00004"
# qmgr -c "set server acl_hosts += nid00064"
# qmgr -c "set server acl_hosts += nid00068"
# qmgr -c "set server submit_hosts += login1"
# qmgr -c "set server submit_hosts += login2"
# qmgr -c "set server submit_hosts += login3"
```

Define your login nodes to Torque.

Define your login nodes to Torque. You should set np to the maximum number of concurrent jobs for your
system. A value of 128 is suggested as a typical setting.

Example 9. Populate the nodes file

In this example we have defined three execution hosts in Torque. Additionally, we assigned specific properties to a couple of the nodes so that particular workload can be directed to these hosts (moms).

```bash
# vi /var/spool/torque/server_priv/nodes

login1 np=128 login2 np=128 mom_himem login3 np=128 mom_netpipe
```

Install the pbs_server init.d script on the server (Optional)

Torque provides an init.d script for starting pbs_server as a service.

Example 10. Copy in init.d script

```bash
# cd /rr/current/software/torque-2.2.0
# cp contrib/init.d/pbs_server /etc/init.d
# chmod +x /etc/init.d/pbs_server

# vi /etc/init.d/pbs_server

PBS_DAEMON=/opt/torque/default/sbin/pbs_server
PBS_HOME=/var/spool/torque
```

Edit the init.d file as necessary -- i.e. change PBS_DAEMON and PBS_HOME as appropriate.

```bash
# vi /etc/init.d/pbs_server

PBS_DAEMON=/opt/torque/default/sbin/pbs_server
PBS_HOME=/var/spool/torque
```

Uncomment the following line to retain core dump files:

```bash
ulimit -c unlimited # Uncomment this to preserve core files
```

Install the pbs_mom init.d script on the login nodes (Optional)

Torque provides an init.d script for starting pbs_mom as a service.

Example 11. Copy in init.d script

```bash
# cd /rr/current/software/torque-2.2.0

# vi contrib/init.d/pbs_mom

PBS_DAEMON=/opt/torque/default/sbin/pbs_mom
PBS_HOME=/var/spool/torque
```

Edit the init.d file as necessary -- i.e. change PBS_DAEMON and PBS_HOME as appropriate, retain core files, etc.

```bash
# vi contrib/init.d/pbs_mom

PBS_DAEMON=/opt/torque/default/sbin/pbs_mom
PBS_HOME=/var/spool/torque
```

Uncomment the following line to retain core dump files:

```bash
ulimit -c unlimited # Uncomment this to preserve core files
```

Stop the Torque server
Example 12. Stop Torque

```bash
# /opt/torque/default/bin/qterm
```
Alternatively, if you installed the init.d script, you may run:

```bash
# service pbs_server stop
```

Update the Torque MOM config file on each MOM node

Edit the MOM config file so job output is copied to locally mounted directories.

Example 13. Edit the MOM config file

```bash
# vi var/spool/torque/mom_priv/config
$usecp *:/home/users /home/users $usecp *:/scratch /scratch
```

Note: It may be acceptable to use a $usecp *:/ / in place of the sample above. Consult with the site.

Startup the Torque Mom Daemons

On the boot node as root:

Example 14. Start up the pbs_moms on the login nodes.

```bash
# pdsh -w login1,login2,login3 /opt/torque/default/sbin/pbs_mom
```
Alternatively, if you installed the init.d script, you may run:

```bash
# pdsh -w login1,login2,login3 /sbin/service pbs_mom start
```

Startup the Torque Server

On the Torque server host as root:

Example 15. Start pbs_server

```bash
# /opt/torque/default/sbin/pbs_server
```
Alternatively, if you installed the init.d script, you may run:

```bash
# service pbs_server start
```

Moab Install Notes

Install Torque

If Torque is not already installed on your system, follow the Torque-XT Installation Notes to install Torque on the SDB node.

Perform the following steps from the boot node as root:

Download the latest Moab release
Download the latest Moab release from Cluster Resources, Inc.

**Note:** The correct tarball type can be recognized by the xt4 tag in its name. The xt4 tarball will be used when it is a Cray XT4, XT5 or later.

**Example 16. Download Moab**

```
# cd /rr/current/software
# wget --http-user=user --http-passwd=passwd
http://www.adaptivecomputing.com/download/mwm/moab-5.4.1-linux-
x86_64-torque2-xt4.tar.gz
```

**Unpack the Moab tarball**

Using xtopview, unpack the Moab tarball into the software directory in the shared root.

**Example 17. Unpack Moab**

```
# xtopview
default/:/ # cd /software
default/:/software # tar -zxvf moab-5.4.1-linux-x86_64-torque2-
xt4.tar.gz
```

**Configure Moab**

While still in xtopview, run configure with the options set appropriately for your installation. Run `./configure` — help to see a list of configure options. Adaptive Computing recommends installing the Moab binaries into `/opt/moab/5version` and establishing a symbolic link to it from `/opt/moab/default`. Since the Moab home directory must be read-write by root, Adaptive Computing recommends you specify the homedir in a location such as `/var/spool/moab`.

```
Moab no longer installs XT4 scripts by default. Use `--with-xt4` when running `./configure` to install them.
```

**Example 18. Run configure**

```
default/:/software # cd moab-5.4.1
default/:/software/moab-5.4.1 # ./configure --prefix=/opt/moab/5.4.1
--with-homedir=/var/spool/moab --with-torque=/opt/torque/default --
with-modulefiles=/opt/modulefiles --with-xt4
default/:/software/moab-5.4.1 # make install
default/:/software/moab-5.4.1 # ln -sf /opt/moab/5.4.1/
/opt/moab/default
```

**Compile and Install Moab**

While still in xtopview, install Moab into the shared root. You may also need to link `/opt/moab/default` to this installation.

**Example 19. Make Install**

```
default/:/software/moab-5.4.1 # make install
default/:/software/moab-5.4.1 # ln -sf /opt/moab/5.4.1/
```

**Install the module files (Optional)**

Moab provides a module file that can be used to establish the proper Moab environment. You may also want to install these module files onto the login nodes.
Example 20. make modulefiles

default::/software/moab-5.4.1 # make modulefiles

Install the Perl XML Modules and exit xtopview

Moab's native resource manager interface scripts require a Perl XML Module to communicate via the basil interface. The Perl XML::LibXML module should be installed. The default method is to use the perldeps make target to install a bundled version of the module into a local Moab lib directory. This module may also be downloaded and installed from Perl's CPAN directory. Exit xtopview.

Example 21. make perldeps

default::/software/moab-5.4.1 # make perldeps
default::/software/moab-5.4.1 # exit

Customize the Moab configuration file for your Moab server host

The moab.cfg file should be customized for your scheduling environment. We will use /rr/current/var/spool/moab as a temporary staging area before copying them out to their final destinations. See the Moab Admin Guide for more details about Moab configuration parameters.

Example 22. Edit the Moab configuration file

```
# cd /rr/current/var/spool/moab
# vi moab.cfg
SCHEDCFG[moab] SERVER=sdb:42559 TOOLSDIR /opt/moab/default/tools
RMCFG[clustername] TYPE=NATIVE:XT4 NODECFG[DEFAULT] OS=linux ARCH=XT
NODEACCESSPOLICY SINGLEJOB JOBMIGRATEPOLICY IMMEDIATE CLIENTCFG[msub]
FLAGS=AllowUnknownResource
```

Customize the XT4 native resource manager interface configuration file

Edit the configuration file ($MOABHOMEDIR/etc/config.xt4.pl) used by the xt tools.

Example 23. Edit the XT4 configuration file

```
# cd /rr/current/var/spool/moab/etc
# vi config.xt4.pl
$ENV(PATH) = "/opt/torque/default/bin:/usr/bin:$ENV(PATH)";
$batchPattern = "^login|xt1|xt2|nid00008\b|nid00011\b";  # Non-interactive jobs run here only
# The following two lines may also modified or uncommented to support
# interactive job launch. This allows the jobs to roam in the event
# the local MOM on the login node is down.
%loginReplaceTable = (nid00008 => login1, nid00011 => login2);
$allowInteractiveJobsToRoam = "True"
```

Copy your Moab home directory to your Moab server host

In this example we assume the Moab server will be running on the SDB node. If you are installing Moab with its server home in /var as in this example and assuming that your var filesystem is being served from your boot node under /snv, you will need to login to SDB and determine the nid with 'cat /proc/cray_xt/nid'.

Example 24. Copy out Moab home directory
Copy the Moab configuration files to all of the login nodes

Both the Moab configuration file (moab.cfg) and the configuration file for the xt4 scripts (config.xt4.pl) must be copied out to the /var filesystem on the login nodes. The only essential parameter that must be in the moab.cfg on the login nodes is the SCHEDCFG line so the clients can find the server.

Example 25. Copy out the configuration files

```bash
# cd /rr/current/var/spool
# cp -pr moab /snv/3/var/spool
```

Install the Moab init.d script (Optional)

Moab provides an init.d script for starting Moab as a service. Using xtopview into the SDB node, copy the init script into /etc/init.d.

Example 26. Copy in init.d script to the SDB node from the shared root.

```bash
# xtopview -n 3	node/3:/ # cp /software/moab/moab-5.1.0/contrib/init.d/moab
/etc/init.d/
```

Edit the init.d file as necessary -- i.e. retain core files, etc.

Uncomment the following line to retain core dump files

```bash
ulimit -c unlimited # Uncomment to preserve core files
```

Perform the following steps from the Moab server node (sdb) as root:

Set the proper environment

The MOABHOMEDIR environment variable must be set in your environment when starting Moab or using Moab commands. If you are on a system with a large number of nodes (thousands), you will need to increase your stack limit to unlimited. You will also want to adjust your path to include the Moab and Torque bin and sbin directories. The proper environment can be established by loading the appropriate Moab module, by sourcing properly edited login files, or by directly modifying your environment variables.

Example 27. Loading the Moab module

```bash
# module load moab
```

Example 28. Exporting the environment variables by hand (in bash)

```bash
# export MOABHOMEDIR=/var/spool/moab
# export PATH=$PATH:/opt/moab/default/bin:/opt/moab/default/sbin:/opt/torque/def
```
Example 29. Setting the stack limit to unlimited

If you are running on a system with large numbers of nodes (thousands), you may need to increase the stack size user limit to unlimited. This should be set in the shell from which Moab is launched. If you start Moab via an init script, this should be set in the script, otherwise it would be recommended to put this in the appropriate shell startup file for root.

```
# ulimit -s unlimited
```

Apply an orphan cleanup policy

Occasionally, Moab can encounter an orphaned ALPS partition -- that is a partition which is no longer associated with an active job. These orphans can occur under different circumstances, such as manually created alps partitions, partitions created by a different resource manager, or as a result of jobs that have been lost to Moab's memory by a catastrophic outage. By setting the MOABPARCLEANUP environment variable, you can set Moab's policy for handling orphaned ALPS partitions. If MOABPARCLEANUP is unset, Moab will not attempt to cleanup orphaned ALPS partitions. If MOABPARCLEANUP is set to Full, Moab will aggressively clean up any orphan it encounters, whether it was the creator of the partition or not. If MOABPARCLEANUP is set to anything else (such as 1, yes, TRUE, etc.), Moab will attempt to clean up only those orphans that it knows that it had a hand in creating. This environment variable must be set in the environment when starting Moab to take effect. This can be accomplished by including it in the appropriate module, init script, or via a manual setenv or export command.

Example 30. Activate aggressive ALPS partition cleanup in the optional Moab startup script

```
# vi /etc/init.d/moab
export MOABPARCLEANUP=Full
```

Customize Moab to use alps topology ordering (Optional)

Communication performance within parallel jobs may be improved by customizing Moab to allocate nodes according to a serialized alps XYZ topology ordering. There are two main methods for doing this -- by presenting the nodes to Moab in the serialized topology order, or by prioritizing the nodes in the serialized topology order. By default, Moab will allocate nodes according to an lexicographical (alphanumeric) ordering.

Option A -- Prioritizing the nodes in serialized topology order. This approach requires that you tell Moab to allocate its nodes according to a priority function based on an explicit priority for each node, which we set based on the alps XYZ ordering. An advantage of this method is that the mdiag -n output will remain in lexicographical ordering. If the apstat -no command is not supported in your version, you may build up the priority list by hand by using the XYZ topology information in the alps database. The example will show how to do this by running a script that uses apstat -no to populate a Moab configuration include file according to the XYZ topology ordering. If your current version of alps does not support the XYZ topology ordering, you may build up the nodeprio.cfg file yourself based on XYZ topology information obtained from alps.

Example 31. Populate a Moab configuration node priority file

```
# /opt/moab/default/tools/node.prioritize.xt.pl
/var/spool/moab/nodeprio.cfg
# echo "#INCLUDE /var/spool/moab/nodeprio.cfg" >
/var/spool/moab/moab.cfg
```

Option B -- Presenting the nodes in serialized topology order. This approach requires that the nodes are reported to Moab in the alps XYZ ordering. Moab will, by default, allocate nodes in the reverse order from which they are reported. This method requires alps support for the XYZ ordering. Its implementation is simple and dynamic but will cause mdiag -n to report the nodes in the serialized topology order. The example will show how to do this by setting a configuration option in the config.xt4.pl file (which was discussed in the previous section).

Example 32. Uncomment the topologyOrdering parameter
Enable steering of jobs to designated execution hosts (Optional)

It is possible to direct a job to launch from an execution host having a job-specified feature. Assigning features to the MOM nodes and declaring these features to be momFeatures allows you to indicate which job features will effect the steering of a job’s master task to certain moms as opposed to steering the job’s parallel tasks to certain compute nodes.

Example 33. Declaring MOM features

Indicates that when a feature of mom_himem or mom_netpipe are specified for a job, this will be used to steer the job to an execution_host (mom) having this feature as opposed to scheduling the parallel tasks on compute nodes having this feature.

```bash
# vi /var/spool/moab/etc/config.xt4.pl
$topologyOrdering = 1;
# Setting momFeatures allows you to indicate which job features will
# effect the steering of jobs to certain moms as opposed to steering to
# compute nodes
@momFeatures = ("mom_himem", "mom_netpipe");
```

Startup the Moab Workload Manager

Start up the Moab daemon.

Example 34. Start Moab

```bash
# /opt/moab/default/sbin/moab
```

Alternatively, if you installed the init.d script, you may run:

```bash
# service moab start
```

Torque Upgrade Notes

Quiesce the system.

It is preferable to have no running jobs during the upgrade. This can be done by closing all queues in Torque or setting a system reservation in Moab and waiting for all jobs to complete. Often, it is possible to upgrade Torque with running jobs in the system, but you may risk problems associated with Torque being down when the jobs complete and incompatibilities between the new and old file formats and job states.

Perform the following steps from the torque server node (sdb) as root:

Shutdown the Torque Mom Daemons

On the boot node as root:

Example 35. Shut down the pbs_moms on the login nodes.

```bash
# pdsh -w login1,login2,login3 /opt/torque/default/sbin/momctl -s
```

Alternatively, if you installed the init.d script, you may run:
Stop the Torque server

Example 36. Stop Torque

```
# /opt/torque/default/bin/qterm
```

Alternatively, if you installed the init.d script, you may run:

```
# service pbs_server stop
```

Perform the following steps from the boot node as root:

Download the latest Torque release.

Download the latest Torque release from Cluster Resources, Inc.

Example 37. Download Torque

```
# cd /rr/current/software
# wget http://www.adaptivecomputing.com/downloads/torque/torque-2.2.0.tar.gz
```

Unpack the Torque tarball

Using xtopview, unpack the Torque tarball into the software directory in the shared root.

Example 38. Unpack Torque

```
# xtopview
default/:/ # cd /software
default/:/software # tar -zxvf torque-2.2.0.tar.gz
```

Configure Torque

While still in xtopview, run configure with the options set appropriately for your installation. Run ./configure --help to see a list of configure options. Adaptive Computing recommends installing the torque binaries into /opt/torque/$version and establishing a symbolic link to it from /opt/torque/default. At a minimum, you will need to specify the hostname where the torque server will run (--with-default-server) if it is different from the host it is being compiled on. The torque server host will normally be the sdb node for XT installations.

Example 39. Run configure

```
default/:/software # cd torque-2.2.0
default/:/software/torque-2.2.0 # ./configure -- prefix=/opt/torque/2.2.0 --with-server-home=/var/spool/torque --with-default-server=nid00003 --enable-syslog
```

Compile and Install Torque

While still in xtopview, compile and install torque into the shared root. You may also need to link /opt/torque/default to this installation. Exit xtopview.
Example 40. Make and Make Install

```
default/:/software/torque-2.2.0 # make
default/:/software/torque-2.2.0 # make packages
default/:/software/torque-2.2.0 # make install
default/:/software/torque-2.2.0 # rm /opt/torque/default
default/:/software/torque-2.2.0 # ln -sf /opt/torque/2.2.0/
   /opt/torque/default
default/:/software/torque-2.2.0 # exit
```

**Startup the Torque Mom Daemons**

**Note:** If you have still have running jobs, you will want to start pbs_mom with the `-p` flag to preserve running jobs. By default, the init.d startup script will not preserve running jobs unless altered to start pbs_mom with the `-p` flag.

On the boot node as root:

**Example 41. Start up the pbs_moms on the login nodes.**

```
# pdsh -w login1,login2,login3 /opt/torque/default/sbin/pbs_mom -p
```

**Startup the Torque Server**

On the torque server host as root:

**Example 42. Start pbs_server**

```
# /opt/torque/default/sbin/pbs_server
```

Alternatively, if you installed the init.d script, you may run:

```
# service pbs_server start
```

**Moab Upgrade Notes**

**Quiesce the system.**

It is preferable to have no running jobs during the upgrade. This can be done by setting a system reservation in Moab and waiting for all jobs to complete. Often, it is possible to upgrade Moab with running jobs in the system, but you may risk problems associated with Moab being down when the jobs complete.

**Shutdown the Moab Workload Manager**

Shut down the Moab daemon.

**Example 43. Stop Moab**

```
# /opt/moab/default/sbin/mschedctl -k
```

Alternatively, if you installed the init.d script, you may run:

```
# service moab stop
```

**Perform the following steps from the boot node as root:**
Download the latest Moab release

Download the latest Moab release from Cluster Resources, Inc.

**Note:** The correct tarball type can be recognized by the xt4 tag in its name.

**Example 44. Download Moab**

```bash
# cd /rr/current/software
# wget --http-user=user --http-passwd=passwd
http://www.adaptivecomputing.com/downloads/mwm/temp/moab-5.2.2.s10021-
linux-x86_64-torque2-xt4.tar.gz
```

Unpack the Moab tarball

Using xtopview, unpack the Moab tarball into the software directory in the shared root.

**Example 45. Unpack Moab**

```bash
# xtopview
default/:/# cd /software
default/:/software # tar -zxvf moab-5.2.2.s10021-linux-x86_64-
torque2-xt4.tar.gz
```

Configure Moab

While still in xtopview, run configure with the options set appropriately for your installation. Run `./configure --help` to see a list of configure options. Adaptive Computing recommends installing the Moab binaries into `/opt/moab/5version` and establishing a symbolic link to it from `/opt/moab/default`. Since the Moab home directory must be read-write by root, Adaptive Computing recommends you specify the homedir in a location such as `/var/spool/moab`.

**Example 46. Run configure**

```bash
default/:/software # cd moab-5.2.2.s10021
default/:/software/moab-5.2.2.s10021 # autoconf
default/:/software/moab-5.2.2.s10021 # ./configure --
prefix=/opt/moab/5.2.2.s10021 --with-homedir=/var/spool/moab --with-
torque
```

Compile and Install Moab

While still in xtopview, install Moab into the shared root. You may also need to link `/opt/moab/default` to this installation.

**Example 47. Make Install**

```bash
default/:/software/moab-5.2.2.s10021 # make install
default/:/software/moab-5.2.2.s10021 # ln -sf /opt/moab/5.2.2.s10021/
/opt/moab/default
```

Install the Perl XML Modules and exit xtopview

If you have previously installed the perl modules in the perl site directories (configure --with-perl-libs=site), you should not need to remake the perl modules. However, the default is to install the perl modules local to the Moab install directory and since it is normal practice to configure the Moab upgrade to use a new install directory (configure --prefix), it will generally be necessary to reinstall the perl modules. Exit xtopview when
done with this step.

**Example 48. make perldeps**

```bash
default/:/software/moab-5.2.2.s10021 # make perldeps
default/:/software/moab-5.2.2.s10021 # exit
```

**Manually merge any changes from the new XT4 native resource manager interface configuration file**

If the upgrade brings in new changes to the config.xt4.pl file, you will need to edit the file and manually merge in the changes from the config.xt4.pl.dist file. One way to discover if new changes have been introduced is to diff the config.xt4.pl.dist from the old and new etc directories. This is rare, but does happen on occasion. One will generally discover quite quickly if necessary changes were not made because the xt4 scripts will usually fail if the config file has not been updated.

**Example 49. Merge any updates into the XT4 configuration file**

```bash
# diff /snv/3/var/spool/moab/etc/config.xt4.pl.dist
/rr/current/software/moab-5.2.2.s10021/etc/config.xt4.pl.dist
# vi /snv/3/var/spool/moab/etc/config.xt4.pl
```

**Reload the new environment**

**Example 50. Swapping in the new Moab module**

```bash
# module swap moab/5.2.2.s10021
```

**Startup the Moab Workload Manager**

Start up the Moab daemon.

**Example 51. Start Moab**

```bash
# /opt/moab/default/sbin/moab
```

Alternatively, if you installed the init.d script, you may run:

```bash
# service moab start
```

**Special Moab Configurations**

**Maintenance Reservations**

For systems using a standing reservation method to block off time for system maintenance, the following examples show two standing reservations which are required.

The first standing reservations is for the compute nodes in the cluster. Set TASKCOUNT to the total number of procs in your cluster:

```bash
SRCFG[PM] TASKCOUNT=7832 NODEFEATURES=compute
SRCFG[PM] PERIOD=DAY DAYS=TUE
SRCFG[PM] FLAGS=OWNERPREFER
SRCFG[PM] STARTTIME=8:00:00 ENDTIME=14:00:00
SRCFG[PM] JOBADTRLIST=PREEMPTEE
```
The second standing reservation is for the login/mom nodes that do not have procs, but execute size 0 jobs using the GRES method. Set TASKCOUNT to the total number of GRES resources on those nodes:

| SRCFG[PM] | TRIGGER=EType=start, Offset=300, AType=internal, Action="rsv::modify:acl:jattr-=PREEMPTEE" |
| SRCFG[PM] | TRIGGER=EType=start, Offset=-60, AType=jobpreempt, Action="cancel" |

| SRCFG[PMsvc] | TASKCOUNT=16 |
| SRCFG[PMsvc] | RESOURCES=GRES=master:100 |
| SRCFG[PMsvc] | PERIOD=DAY DAYS=TUE |
| SRCFG[PMsvc] | FLAGS=OWNERPREEMPT |
| SRCFG[PMsvc] | STARTTIME=8:00:00 ENDTIME=14:00:00 |
| SRCFG[PMsvc] | JOBATTRLIST=PREEMPTEE |
| SRCFG[PMsvc] | TRIGGER=EType=start, Offset=300, AType=internal, Action="rsv::modify:acl:jattr-=PREEMPTEE" |
| SRCFG[PMsvc] | TRIGGER=EType=start, Offset=-60, AType=jobpreempt, Action="cancel" |
Validating an xCAT Installation for Use with Moab

- Introduction to Validating xCAT Configuration
- Verifying Node List
- Reporting Node Status
- Verifying Hardware Management Configuration
- Verifying Provisioning Images
- Verifying VM Migration

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`.

```bash
[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.

```bash
[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.

```bash
[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.

```bash
[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
```

### See Also

- [Native Resource Manager Overview](#)
- [Resource Provisioning](#)
Integrating an xCAT Physical Provisioning Resource Manager with Moab

- xCAT Configuration Requirements
- MSM Installation
- Integrating MSM and xCAT
- MSM Configuration
- Configuration Validation
- Troubleshooting
- Deploying Images with TORQUE
- Installing Moab on the Management Node
- Moab Configuration File Example
- Verifying the Installation
- xCAT Plug-in Configuration Parameters

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

```bash
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:

```bash
> cp $MSMHOMEDIR/contrib/xcat/MSM.pm $XCATROOT/lib/perl/xCAT_schema
```

Restart xcatd and check the x_msm table is correctly created:

```bash
> 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:

```bash
> tabedit x_msm
```

- **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
- **vmgrouname** - 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

```bash
> tabedit nodegroup
```

Edit as necessary to simulate the following example:
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:
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 Node Manager (MOM) Configuration 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/moab.cfg. A sample configuration file, set up and optimized for adaptive computing follows:

```
# Example moab.cfg
SCHEDCFG[Moab]     SERVER=gpc-sched:42559
ADMCFG[1]           USERS=root,egan
```
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 NODEIDLEPOWERMEMORYTHRESHOLD), Moab creates a power-off system job. This job communicates with xCAT to power off the nodes and remain 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   Resource Type: 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/wightm
```
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
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)
```
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------------------------
<table>
<thead>
<tr>
<th>JOBID</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROCs</th>
<th>REMAINING</th>
</tr>
</thead>
<tbody>
<tr>
<td>provision-4</td>
<td>root</td>
<td>Running</td>
<td>8</td>
<td>00:01:00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Fri Jun 19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>09:12:56</td>
</tr>
<tr>
<td>1 active job</td>
<td>8 of 40 processors in use by local jobs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 of 10 nodes active (20.00%)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

eligible jobs----------------------
<table>
<thead>
<tr>
<th>JOBID</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROCs</th>
<th>WCLIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>yuby.3</td>
<td>wightman</td>
<td>Idle</td>
<td>8</td>
<td>00:10:00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Fri Jun 19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>09:12:55</td>
</tr>
<tr>
<td>1 eligible job</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

blocked jobs-----------------------
<table>
<thead>
<tr>
<th>JOBID</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROCs</th>
<th>WCLIMIT</th>
</tr>
</thead>
</table>

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------------------------
<table>
<thead>
<tr>
<th>JOBID</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROCs</th>
<th>REMAINING</th>
</tr>
</thead>
<tbody>
<tr>
<td>yuby.3</td>
<td>wightman</td>
<td>Running</td>
<td>8</td>
<td>00:08:49</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Fri Jun 19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>09:13:29</td>
</tr>
<tr>
<td>1 active job</td>
<td>8 of 40 processors in use by local jobs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 of 10 nodes active (20.00%)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

eligible jobs----------------------
<table>
<thead>
<tr>
<th>JOBID</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROCs</th>
<th>WCLIMIT</th>
</tr>
</thead>
</table>

0 eligible jobs

blocked jobs-----------------------
<table>
<thead>
<tr>
<th>JOBID</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROCs</th>
<th>WCLIMIT</th>
</tr>
</thead>
</table>
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**

| Description | Module                  | LogLevel        | PollInterval      | TimeOut          | NodeRange          | CQxCATSessions | DORVitals      | PowerString | DoNodeStat | DoxCATStats | LockDir | _HVxCATPasswdKey | _FeatureGroups | _UseStates | _ImagesTabName | _VerifyRPower | _RPowerTimeOut | _QueueRPower | _RPowerQueueAge | _RPowerQueueSize | _MaskOSWhenOff | _ModifyTORQUE | _ReportNETADDR | _UseOpIDs | _VMInterfaces | _ESXCFGPath | _ESXStore | _KVMStoragePath |
|-------------|-------------------------|-----------------|-------------------|------------------|-------------------|------------------|---------------|--------------|------------|------------|------------|--------|----------------|---------------|------------|----------------|---------------|----------------|--------------|----------------|----------------|------------|----------------|---------------|-----------|---------------|----------|---------------|-------------|-----------|----------------|----------|
| **Description** | None | 5 | 1-9 | Double quoted string containing brief description of plugin. | Name of the plugin module to load. | 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

<table>
<thead>
<tr>
<th>Default Value:</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid Value:</td>
<td>Integer &gt; 0</td>
</tr>
</tbody>
</table>
| Comments:      | 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

<table>
<thead>
<tr>
<th>Default Value:</th>
<th>300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid Value:</td>
<td>Integer value &gt; POLLINTERVAL</td>
</tr>
<tr>
<td>Comments:</td>
<td>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.</td>
</tr>
</tbody>
</table>

### _NodeRange

<table>
<thead>
<tr>
<th>Default Value:</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid Value:</td>
<td>Any valid noderange (see the xCAT noderange manpage).</td>
</tr>
<tr>
<td>Comments:</td>
<td>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.</td>
</tr>
</tbody>
</table>

### _CQxCATSessions

<table>
<thead>
<tr>
<th>Default Value:</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid Value:</td>
<td>Positive integer &gt; 1</td>
</tr>
<tr>
<td>Comments:</td>
<td>MSM will divide the node list generated by 'nodelist' 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.</td>
</tr>
</tbody>
</table>

### _DORVitals

<table>
<thead>
<tr>
<th>Default Value:</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Valid Value:</strong></td>
<td>0</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
<td>When set to 1, MSM will poll rvitals power and led status (see the xCAT rvitals manpage). This only works with IBM BMCs currently. In order to use this, xCAT should respond without error to the 'rvitals &lt;noderange&gt; watts' and 'rvitals &lt;noderange&gt; leds' commands. Status is reported as GMETRIC[watts] and GMETRIC[leds]. See also the _POWERSTRING configuration parameter.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>_PowerString</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Value:</strong></td>
<td>'AC Avg Power'</td>
</tr>
<tr>
<td><strong>Valid Value:</strong></td>
<td>single quote delimited string</td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
<td>Only meaningful when used with _DORVITALS=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 &lt;node_name&gt; power' and examine the output to determine what the appropriate value of this string is.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>_DoNodeStat</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Value:</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Valid Value:</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>_DoxCATStats</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Value:</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Valid Value:</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
<td>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 configuration parameters.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>_LockDir</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Value:</strong></td>
<td>$MSMHOMEDIR/lck</td>
</tr>
<tr>
<td><strong>Valid Value:</strong></td>
<td>Existing path on MSM host</td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
<td>This is a path to where MSM maintains lock files to control concurrency with some Xen and KVM operations.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>_HVxCATPasswdKey</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
</tr>
<tr>
<td>vmware</td>
<td></td>
</tr>
</tbody>
</table>

**FeatureGroups**

<table>
<thead>
<tr>
<th>Default Value</th>
<th>Valid Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>Comma delimited string of xCAT group names.</td>
<td>MSM builds the OLIST 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 parameter.</td>
</tr>
</tbody>
</table>

**_DefaultVMCProc**

<table>
<thead>
<tr>
<th>Default Value</th>
<th>Valid Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-?</td>
<td>If not explicitly specified in the create request, MSM will create VMs with this many processors.</td>
</tr>
</tbody>
</table>

**_DefaultVMDisk**

<table>
<thead>
<tr>
<th>Default Value</th>
<th>Valid Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096</td>
<td>Positive integer values, minimum is determined by your vm image needs</td>
<td>If not explicitly specified in the create request, MSM will create VMs with this much disk allocated.</td>
</tr>
</tbody>
</table>

**_DefaultVMCMemory**

<table>
<thead>
<tr>
<th>Default Value</th>
<th>Valid Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>Positive integer values, minimum is determined by your vm image needs</td>
<td>If not specified, MSM will create VMs with this much memory allocated.</td>
</tr>
</tbody>
</table>

**_KVMStoragePath**

<table>
<thead>
<tr>
<th>Default Value</th>
<th>Valid Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>/vms</td>
<td>Existing path on MSM host</td>
<td>File backed disk location for stateful KVM VMS will be placed here.</td>
</tr>
</tbody>
</table>
### _ESXStore

<table>
<thead>
<tr>
<th>Default Value:</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid Value:</td>
<td>Mountable NFS Path</td>
</tr>
</tbody>
</table>

**Comments:** Location of ESX stores.

### _ESXCFGPath

<table>
<thead>
<tr>
<th>Default Value:</th>
<th>ESXStore</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid Value:</td>
<td>Mountable NFS Path</td>
</tr>
</tbody>
</table>

**Comments:** Location of ESX VM configuration files.

### _VMInterfaces

<table>
<thead>
<tr>
<th>Default Value:</th>
<th>br0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid Value:</td>
<td>Name of bridge device in your VM image</td>
</tr>
</tbody>
</table>

**Comments:** Bridge device name passed to libvirt for network configuration of VMs (overrides _XENHOSTINTERFACES and _KVMHOSTINTERFACES if specified).

### _XenHostInterfaces

<table>
<thead>
<tr>
<th>Default Value:</th>
<th>xenbr0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid Value:</td>
<td>Name of bridge device in your VM image</td>
</tr>
</tbody>
</table>

**Comments:** Bridge device name passed to libvirt for network configuration of Xen VMs.

### _KVMHostInterfaces

<table>
<thead>
<tr>
<th>Default Value:</th>
<th>br0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid Value:</td>
<td>Name of bridge device in your VM image</td>
</tr>
</tbody>
</table>

**Comments:** Bridge device name passed to libvirt for network configuration of KVM VMs.

### _VMSovereign

<table>
<thead>
<tr>
<th>Default Value:</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid Value:</td>
<td>0</td>
</tr>
</tbody>
</table>

**Comments:** Setting this attribute will cause VMs to be reported to Moab with SOVEREIGN=1 in the
**VARATTR Flag.** Setting this causes Moab to reserve VMs memory and procs on the hypervisor, and treat the VM as the workload - additional workload cannot be scheduled on the VMs.

<table>
<thead>
<tr>
<th>_UseStates</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Value:</strong></td>
</tr>
<tr>
<td><strong>Valid Value:</strong></td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>_ImagesTabName</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Value:</strong></td>
</tr>
<tr>
<td><strong>Valid Value:</strong></td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>_VerifyRPower</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Value:</strong></td>
</tr>
<tr>
<td><strong>Valid Value:</strong></td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
</tr>
<tr>
<td><strong>NOTE:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>_RPowerTimeOut</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Value:</strong></td>
</tr>
<tr>
<td><strong>Valid Value:</strong></td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>_QueueRPower</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Value:</strong></td>
</tr>
<tr>
<td><strong>Valid Value:</strong></td>
</tr>
<tr>
<td>Parameters</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
<tr>
<td>_RPowerQueueAge</td>
</tr>
<tr>
<td>_RPowerQueueSize</td>
</tr>
<tr>
<td>_MaskOSWhenOff</td>
</tr>
<tr>
<td>_ModifyTORQUE</td>
</tr>
<tr>
<td>_ReportNETADDR</td>
</tr>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>Value:</td>
</tr>
<tr>
<td>_UseOpIDs</td>
</tr>
<tr>
<td>_VMIPRange</td>
</tr>
<tr>
<td>_xCATHost</td>
</tr>
</tbody>
</table>
Enabling Moab Provisioning with SystemImager

The **SystemImager** tool is a widely used open source tool that allows flexible automated installation or provisioning of compute hosts within a cluster. Interfacing Moab with **SystemImager** can be done in one of two ways: (1) using triggers and (2) using a native resource manager interface.

**Trigger Based Provisioning Interface**

When a job or reservation becomes active, Moab can custom tailor its environment including changing the operating system of the allocated nodes through the use of triggers. In the case of a job trigger, You can use something like the following:

```bash
moab.cfg
```

```bash
RSVPPROFILE[rhel3]
TRIGGER=etype=start,atype=exec,action="/usr/local/tools/nodeinstall.si.$REQOS $HOSTLIST"

RSVPPROFILE[rhel3]
TRIGGER=etype=end,atype=exec,action="/usr/local/tools/nodeinstall.si.pl
$DEFOS $HOSTLIST"

RSVPPROFILE[rhel3]
TRIGGER=etype=cancel,atype=exec,action="/usr/local/tools/nodeinstall.si
$DEFOS $HOSTLIST"
```

In the preceding example, any reservation that uses the `rhel3` profile will reinstall all nodes to use the `rhel3` operating system for the duration of the reservation. The second and third trigger makes certain that when the reservation ends or if it is canceled, the nodes are restored to their default operating system.

**Resource Manager Based Provisioning Interface**

With a resource manager based provisioning interface, Moab uses the provisioning manager to create the nodes needed by various jobs. In this model, the provisioning manager can be set up, users can submit jobs requiring any available operating system, and Moab can dynamically reprovision compute nodes to meet current workload needs.

```bash
moab.cfg
```

```bash
RMCFG[torque] TYPE=PBS
RMCFG[si] TYPE=native RTYPE=provision
RMCFG[si] CLUSTERQUERYURL=exec://$TOOLSDIR/clusterquery.si.pl
RMCFG[si] SYSTEMQUERYURL=exec://$TOOLSDIR/systemquery.si.pl
RMCFG[si] SYSTEMMODIFYURL=exec://$TOOLSDIR/systemmodify.si.pl
```

With this configuration, Moab can automatically load-balance resources to meet the needs of submitted jobs. Correct operation of the interface’s querying capabilities can be verified by issuing `mdiag -R` and `mdiag -n` to look at resource manager and node configuration respectively.

To verify that Moab can correctly drive **SystemImager** to install a node, use the `mnodectl -m` command as demonstrated in the following example:

```bash
mnodectl -m > mnodectl -m os=rhel3 node002
```

**See Also**

- Native Resource Manager Overview
- Resource Provisioning
Moab-NUMA Integration Guide

Scheduling a NUMA type system requires some special configuration. Moab uses NODESETs and NODEEAVAILABILITYPOLICY to determine when and where jobs can run in a NUMA environment.

This guide assumes Moab is scheduling a single NUMA system. Each node in the system must be configured with the same feature.

To integrate Moab and NUMA, follow these steps:

1. Configure the NODESETs to use the node features.

   NODESETPOLICY ONEOF
   NODESETATTRIBUTE FEATURE
   NODESETISOPTIONAL FALSE
   NODESETPRIORITYTYPE MINLOSS
   NODESETLIST uv

2. Configure Moab to use the "PRIORITY" NODEALLOCATIONPOLICY.

   NODEALLOCATIONPOLICY PRIORITY

3. Configure Moab so that any job that runs on the NUMA partition requests SharedMem.

   RMCFG[base] TYPE=PBS
   PARCFG[base] FLAGS=SharedMem

If Moab is scheduling a single partition the following flag can be used:

   PARCFG[ALL] FLAGS=SharedMem

Jobs requesting shared memory should be submitted using the "-l flags=sharedmem" option.
Appendix Q: Moab in the Data Center

Moab provides support for today’s data centers, a hybrid of traditional data center workflows and dynamic, service-oriented processes. The sections below describe how a data center can best take advantage of Moab’s capabilities.

- Q.1 Introduction
  - Q.1.1 The Traditional Data Center
  - Q.1.2 Moab Utility/Hosting Suite
- Q.2 Installation
  - Q.2.1 Moab
  - Q.2.2 Resource Managers
  - Q.2.3 Checking the Install
- Q.3 Transitioning Workflow
  - Q.3.1 Defining the Workflow
  - Q.3.2 Inventory of Resources
    - Q.3.2.1 Moab Node Structure
    - Q.3.2.2 Defining Nodes in Moab
    - Q.3.2.3 Mapping Workflow Requirements to Resources
  - Q.3.3 Defining Job Groups
  - Q.3.4 Setting the Schedule
    - Q.3.4.1 Determining Scheduling Requirements
    - Q.3.4.2 Creating Standing Reservations
    - Q.3.4.3 Submitting a Job to a Standing Reservation
  - Q.3.5 Workflow Dependencies
    - Q.3.5.1 Converting Compute Jobs
    - Q.3.5.2 Introduction to Triggers
    - Q.3.5.3 Internal Dependencies
    - Q.3.5.4 External Dependencies
    - Q.3.5.5 Cascading Triggers
- Q.4 Dynamic Workload
- Q.5 Supporting SLAs and Other Commitments
- Q.CS Case Studies
  - Q.CS.1 Cascading Triggers

Q.1 Introduction

Note: The Intersection of HPC & the Data Center is a video tutorial of a session offered at Moab Con that offers further details for understanding High Performance Computing (HPC) and data centers.

Note: Adaptive Data Center is a video tutorial that offers further details for understanding adaptive data centers.

Welcome to Moab in the Data Center. Widely used in the HPC sector, Moab provides many unique solutions to the problems faced by today’s data center administrators on a daily basis. In addition to supporting the traditional data center workload model, Moab leverages more than ten years of experience in HPC to provide the dynamic scheduling needed in today’s data center where Web services and other ad hoc, service-oriented processes are becoming more prevalent. This document outlines the easy process of evolving to this new paradigm—a world where the data center and service-oriented computing intertwine.

Q.1.1 The Traditional Data Center

Data centers often view compute resources differently than traditional HPC centers. The flow of data and computation within many data centers is fairly static in nature, with each new day being very similar to the last. Workload may vary throughout the day and week to match standard business hours. Month-end, quarter-end and year-end create predictable spikes in work. So, while workload may vacillate, there is a predictable ebb and flow.
Table 1: Data Center vs. HPC Comparison

<table>
<thead>
<tr>
<th>Data Center</th>
<th>HPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-Centric Jobs</td>
<td>Compute-Centric Jobs</td>
</tr>
<tr>
<td>Standardized/Static Workload</td>
<td>Dynamic Workload</td>
</tr>
<tr>
<td>Short Jobs</td>
<td>Long Jobs</td>
</tr>
<tr>
<td>Many Dependencies</td>
<td>Few Dependencies</td>
</tr>
<tr>
<td>Specific Resources for Specific Jobs</td>
<td>Non-Specific Resources for All Jobs</td>
</tr>
</tbody>
</table>

Table 1 is an overly-simplified, overly-generalized view of the differences between the traditional Data Centers and HPC models. However, in most cases, this description is fairly accurate. However, this is changing within the industry. Recently, many data centers have begun to offer additional, on-demand, service-oriented products to their clients, such as Web services support and database searches, thus creating a hybrid model somewhere between traditional data center and traditional HPC.

The infusion of these dynamic jobs into the data center have put new demands on the system and its administrators. The new dynamic jobs must be handled in such a way as to protect the core business processes, while still providing the contracted dynamic services to clients and meeting SLAs. The sophistication required in the management software has stretched many current data center solutions to or past the breaking point, prompting administrators to search for new solutions.

**Q.1.2 Moab Utility/Hosting Suite**

Moab Utility/Hosting Suite is a professional cluster workload management solution that integrates the scheduling, managing, monitoring and reporting of cluster workloads. Moab Utility/Hosting Suite simplifies and unifies management across one or multiple hardware, operating system, storage, network, license and resource manager environments. Its task-oriented management and the industry's most flexible policy engine ensure service levels are delivered and workload is processed faster. This enables organizations to accomplish more work resulting in improved cluster ROI.

The power is in the software. Moab will run on today's common hardware solutions. There isn’t a need to replace or make changes to the data center's underlying architecture, in most cases. Moab can interface with many different types of resource management software, including in-house solutions, to learn about its environment. With the gathered information, Moab dynamically schedules resources and plans for the future. So, the system is optimized not only for the moment, but also for the future.

With its state-of-the-art scheduling engine, Moab can empower today's data centers to handle tomorrow's workload. Relying on technology and techniques developed over the last decade, Moab's three-dimensional scheduling algorithms provide an environment where static workloads and dynamic jobs can peacefully coexist, each receiving the proper resources at the proper time. The fully-customizable policy control mechanism allows data centers to enforce any needed policy, whether technical, financial, or political. Both GUI and CLI tools exist for administration.

**Q.2 Installation**

The installation process for a Moab system is straightforward. However, it is accomplished in two separate steps: Moab and the resource managers.

**Q.2.1 Moab**

In most cases, Moab is distributed as a binary tarball. Builds are readily available for all major architectures and operating systems. These packages are available to those with a valid full or evaluation license.

Example 1: Moab install process

```bash
> tar xvzf moab-5.1.0-i386-libtorque-p2.tar.gz
> ./configure
> make
> make install
```

Example 1 shows the commands to do a basic installation from the command line for Moab. In this case, the install package for Moab 5.1.0 (patch 2) needs to be in the current directory.
Example 2: Default contents of /opt/moab

By default, Moab installs to the /opt/moab directory. Example 2 shows a sample ls -l output in /opt/moab.

The binary installation creates a default moab.cfg file in the etc/ folder. This file contains the global configuration for Moab that is loaded each time Moab is started. The definitions for users, groups, nodes, resource manager, quality of services and standing reservations are placed in this file. While there are many settings for Moab, only a few will be discussed here. The default moab.cfg that is provided with a binary installation is very simple. The installation process defines several important default values, but the majority of configuration needs to be done by the administrator, either through directly editing the file or using one of the provided administrative tools such as Moab Cluster Manager (MCM).

Example 3: Default moab.cfg file

Example 3 shows the default moab.cfg file sans-comments. The first line defines a new scheduler named Moab. In this case, it is located on a host named allbe and listening on port 42559 for client commands. These values are added by the installation process, and should be kept in most cases.

The second line, however, requires some editing by the administrator. This line defines what users on the system have Level 1 Administrative rights. These are users who have global access to information and unlimited control over scheduling operations in Moab. There are five default administrative levels defined by Moab, each of which is fully customizable. In Example 3, this line needs to be updated. The second root entry needs to be changed to the username of the administrator(s) of the system. The first root entry needs to remain, as Moab needs to run as root in order to submit jobs to the resource managers as the original owner.

The final line in this example is the configuration for the default resource manager. This particular binary distribution is for the TORQUE resource manager. Because TORQUE follows the PBS style of job handling, the resource manager is given a type of PBS. To differentiate it from other resource managers that may be added in the future, it is also given the name base. Resource managers will be discussed in the next section.

This constitutes the basic installation of Moab. Many additional parameters can be added to the moab.cfg file in order to fully adapt Moab to the needs of your particular data center. A more detailed installation guide is available.

Q.2.2 Resource Managers

The job of Moab is to schedule resources. In fact, Moab views the world as a vast collection of resources that can be scheduled for different purposes. It is not, however, responsible for the direct manipulation of these resources. Instead, Moab relies on resource managers to handle the fine details in this area. Moab makes decisions and sends the necessary commands to the resource managers, which execute the commands and return state information back to Moab. This decoupling of Moab from the actual resources allows Moab to support all possible resource types, as it doesn't need specific knowledge about the resources, only a knowledge of how to communicate with the resource manager.

Moab natively supports a wide range of resource managers. For several of these, Moab interacts directly with the resource manager's API. These include TORQUE, LSF and PBS. For these resource managers, a specific binary build is required to take advantage of the API calls. For other resource managers, Moab supports a
generic interface known as the **Native Interface**. This allows interfaces to be built for any given type of resource manager, including those developed in-house. Cluster Resources supplies a large number of pre-built interfaces for the most common resource managers. They also provide information on building custom interfaces, as well as contract services for specialized development.

The setup of each individual resource manager is beyond the scope of this document. However, most resource managers come with ample instructions and/or wizards to aid in their installation. **TORQUE**, an open-source resource manager under the auspice of Cluster Resources, has a documentation WIKI. This includes instructions on installing and testing TORQUE. Please note that special SSH or NFS configuration may also be required in order to get data staging to work correctly.

Once the resource manager(s) are installed and configured, the `moab.cfg` file will need to be updated if new or different resource managers have been added. Valid resource manager types include: LL, LSF, PBS, SGE, SSS and WIKI. General information on resource managers is found in Chapter 13. Integration guides for specific resource managers are also available.

**Q.2.3 Checking the Installation**

Once Moab and the resource managers have been installed, there are several steps that should be followed to check the installation.

1. **Start the Resource Manager** — See resource manager's documentation
2. **Start Moab** — Run Moab from the command line as root
   ```bash
   > moabd
   ```
   *moabd* is a safe and recommended method of starting Moab if things are not installed in their default locations.
3. **Run Resource Manager Tests** — See resource manager's documentation
4. **Check Moab Install** — See below

Checking the Moab installation is a fairly straightforward process. The first test is to run the program `showq`. This displays the Moab queue information. **Example 4** shows a sample output from `showq`. In this case, the system shown is a single node, which is a 64-processor SMP machine. Also, there are currently no jobs running or queued. So, there are no active processors or nodes.

**Example 4: Sample showq**

<p>| active jobs-------------------------- |</p>
<table>
<thead>
<tr>
<th>JOBID</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROC</th>
<th>REMAINING</th>
<th>STARTTIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 active jobs</td>
<td>0 of 64 processors in use by local jobs</td>
<td>(0.00%)</td>
<td>0 of 1 nodes active</td>
<td>(0.00%)</td>
<td></td>
</tr>
</tbody>
</table>

<p>| eligible jobs---------------------- |</p>
<table>
<thead>
<tr>
<th>JOBID</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROC</th>
<th>WCLIMIT</th>
<th>QUEUETIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 eligible jobs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| blocked jobs---------------------- |</p>
<table>
<thead>
<tr>
<th>JOBID</th>
<th>USERNAME</th>
<th>STATE</th>
<th>PROC</th>
<th>WCLIMIT</th>
<th>QUEUETIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 blocked jobs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The important thing to look for at this point is the total number of processors and nodes. If either the total number of processors or nodes is 0, there is a problem. Generally, this is would be caused by a communication problem between Moab and the resource manager, assuming the resource manager is
configured correctly and actively communicating with each of the nodes for which it has responsibility.

The current state of the communication links between Moab and the resource managers can be viewed using the `mdiag -R -v` command. This gives a verbose listing of the resource managers configured in Moab, including current state, statistics, and any error messages.

**Example 5: Sample `mdiag -R -v` output**

```
<table>
<thead>
<tr>
<th>Resource Manager</th>
<th>State</th>
<th>Type</th>
<th>Resource Type</th>
<th>Version</th>
<th>Objects Reported</th>
<th>Flags</th>
<th>Partition</th>
<th>Event Management</th>
<th>Total Jobs Started</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM[base]</td>
<td>Active</td>
<td>PBS</td>
<td>COMPUTE</td>
<td>'2.2.0'</td>
<td>Nodes=1 (64 pros)</td>
<td>executionServer, noTaskOrdering</td>
<td>base</td>
<td>EPORT=15004</td>
<td>3</td>
</tr>
<tr>
<td>RM[internal]</td>
<td>---</td>
<td>SSS</td>
<td></td>
<td>'SSS4.0'</td>
<td></td>
<td>localQueue</td>
<td></td>
<td>(event interface disabled)</td>
<td></td>
</tr>
</tbody>
</table>
```

In **Example 5**, two different resource managers are listed: `base` and `internal`. The `base` resource manager is the TORQUE resource manager that was defined in **Example 3**. It is currently showing that it is healthy and there are no communication problems. The `internal` resource manager is used internally by Moab for a number of procedures. If there were any problems with either resource manager, messages would be displayed here. Where possible, error messages include suggested fixes for the noted problem.

Another command that can be very helpful when testing Moab is `mdiag -C`, which does a format check on the `moab.cfg` file to ensure that each line has a recognizable format.

**Example 6: Sample `mdiag -C` output**

```
INFO: line #15 is valid: 'SCHEDCFG[Moab]  SERVER=allbe:42559'
INFO: line #16 is valid: 'ADMINCFG[1]     USERS=root,guest2'
INFO: line #23 is valid: 'RMCFG[base]     TYPE=PBS'
```

The state of individual nodes can be checked using the `mdiag -n` command. Verbose reporting of the same information is available through `mdiag -n -v`.

**Example 7: Sample `mdiag -n` output**

```
compute node summary
Name     | State | Procs | Memory | Opsys
-------------------|-------|-------|--------|-------
```
In this case (Example 7), there is only a single compute node, `allbe`. This node has 64 processors and is currently idle, meaning it is ready to run jobs, but is not currently doing anything. If a job or jobs had been running on the node, the node would be noted as active, and the `Procs` and `Memory` column would indicate not only the total number configured, but also the number currently available.

The next test is to run a simple job using Moab and the configured resource manager. This can be done either through the command line or an administrative tool like MCM. This document will show how this is done utilizing the command line.

Example 8: Simple `sleep` job

```bash
> echo "sleep 60" | msub
```

The command in Example 8 submits a job that simply sleeps for 60 seconds and returns. While this may appear to have little or no point, it allows for the testing of the job submission procedures. As the root user is not allowed to submit jobs, this command needs to be run as a different user. When this command is run successfully, it will return the Job ID of the new job. The job should also appear in `showq` as running (assuming the queue was empty), seen in Example 9.

Example 9: Sample `showq` output with running job

```
active jobs------------------------
JOBID  USERNAME  STATE  PROC  REMAINING  STARTTIME
40277  user1    Running  1   00:59:59  Tue Apr  3 11:23:33
1 active job
(1.56%)  1 of 64 processors in use by local jobs
1 of 1 nodes active  (100.00%)
eligible jobs----------------------
JOBID  USERNAME  STATE  PROC  WCLIMIT  QUEUETIME
0 eligible jobs
blocked jobs----------------------
JOBID  USERNAME  STATE  PROC  WCLIMIT  QUEUETIME
0 blocked jobs
Total jobs:  1
```

If `showq` indicated a problem with the job, such as it being blocked, additional information regarding the job can be gained using `checkjob job_id`. Example 10 shows some sample output of this command. More verbose information can be gathered using `checkjob -v job_id`.

Example 10: Sample `checkjob` output

```
job 40277
```
If jobs can be submitted and run properly, the system is configured for basic use. As the transition to a Moab-centric system continues, additional items will be placed in the *moab.cfg* file. After each change to the *moab.cfg* file, it is necessary to restart Moab for the changes to take effect. This simple process is shown in Example 11.

**Example 11: Restarting Moab**

```
> mschedctl -R
```

Other commands administrators will find useful are shown below.

**Example 12: Shutting down Moab**

```
> mschedctl -k
```

**Example 13: Show server statistics**

```
> mdiag -S
```

**Example 14: Show completed jobs (last 5 minutes)**

```
> showq -c
```

## Q.3 Transitioning Workflow

With its advanced scheduling capabilities, Moab can easily handle all the scheduling needs of data centers. Core resources can be protected while still optimizing the workload to get the highest efficiency, productivity, and ROI possible. Almost all of the Data Center's existing structure and architecture is maintained. Additional steps must be made to describe the workflow and related policies to Moab, which will then intelligently schedule the resources to meet the organization's goals. Transitioning to Moab follows some important steps:
1. Determine existing business processes and related resources and policies.
2. For each business process, determine and chart the process, making sure to include all required resources, start times, deadlines, internal dependencies, external dependencies, decision points and critical paths.
3. Where possible, divide processes into functional groups that represent parts of the overall process that should be considered atomic or closely related.
4. Translate functional groups and larger processes into programmatic units that will be scheduled by Moab.
5. Build control infrastructure to manage programmatic units to form processes.
6. Identify on-demand services and related resources and policies.
7. Implement system-wide policies to support static and dynamic workloads.

Most data centers have already done many, if not all, of these steps in the normal management of their system. The other major part of the transition is the taking of these steps and applying them to Moab to gain the desired results. The remainder of this section covers approaches and techniques you can use to accomplish these tasks.

Q.3.1 Defining the Workflow

Workflow is the flow of data and processing through a dynamic series of tasks controlled by internal and external dependencies to accomplish a larger business process. As one begins the transition to Moab, it is important to have a clear understanding of the needed workflow and the underlying business processes it supports. Figure 1 shows a simple workflow diagram with multiple processing paths and external data dependencies.

Figure 1: Sample workflow diagram

It is important to create one or more diagrams such as this to document the workflow through one's system. This diagram provides a visual representation of the system, which clearly shows how data flows through the processing jobs, as well as all dependencies.

In this diagram, External Data is a type of external dependency. An external dependency is a dependency that is fulfilled external to the workflow (and maybe even the system). It is not uncommon for external
dependencies to be fulfilled by processes completely external to the cluster on which Moab is running. Consequently, it is very important to clearly identify such dependencies, as special steps must be taken for Moab to be alerted when these dependencies are fulfilled.

Once all of the business processes have been succinctly described and diagrammed, it is possible to continue with the conversion process. The next step is to inventory the available resources. It is also at this point that the pairing of resources and workflow tasks is done.

Q.3.2 Inventory of Resources

At the most basic level, Moab views the world as a group of resources onto which reservations are placed, managed, and optimized. It is aware of a large number of resource types, including a customizable generic resource that can be used to meet the needs of even the most complex data center or HPC center. Resources can be defined as software licenses, database connections, network bandwidth, specialized test equipment or even office space. However, the most common resource is a compute node.

Q.3.2.1 Moab Node Structure

During the transition to Moab, the administrator must make an inventory of all relevant resources and then describe them to Moab. Because the approach with Moab is different than the traditional data center paradigm, this section will focus on the configuration of compute nodes. Figure 2 shows a basic view of Moab node structure.

Figure 2: Moab node structure

All clusters have a head node, which is where Moab resides. It is from this node that all compute jobs are farmed out to the compute nodes. These compute nodes need not be homogeneous. Figure 2 depicts a heterogeneous cluster where the different colors represent different compute node types (architecture, processors, memory, software, and so forth).

Q.3.2.2 Defining Nodes in Moab

Moab interacts with the nodes through one or more resource managers. Common resource managers include TORQUE, SLURM, LSF, LoadLeveler and PBS. In general, the resource manager will provide Moab with most of the basic information about each node for which it has responsibility. This basic information includes number of processors, memory, disk space, swap space and current usage statistics. In addition to the information provided by the resource manager, the administrator can specify additional node information in the moab.cfg file. Following is an extract from the moab.cfg file section on nodes. Because this is a simple example, only a few node attributes are shown. However, a large number of possible attributes, including customizable features are available. Additional node configuration documentation is available.

Example 15: Sample node configuration

```
NODECFG[Node01] ARCH=ppc OS=osx
NODECFG[Node02] ARCH=ppc OS=osx
NODECFG[Node07] ARCH=i386 OS=suse10
NODECFG[Node08] ARCH=i386 OS=suse10
NODECFG[Node18] ARCH=opteron OS=centos CHARGERATE=2.0
```
In Example 15, we see six nodes. The architecture and operating system is specified on all of these. However, the administrator has enabled a double charge rate for jobs that are launched on the last two. This is used for accounting purposes.

Node attributes are considered when a job is scheduled by Moab. For example, if a specific architecture is requested, the job will only be scheduled on those nodes that have the required architecture.

Q.3.2.3 Mapping Workflow Requirements to Resources

Once relevant resources have been configured in Moab, it is necessary to map the different stages of the workflow to the appropriate resources. Each compute stage in the workflow diagrams needs to be mapped to one or more required resource. In some cases, specifying "No Preference" is also valid, meaning it does not matter what resources are used for the computation. This step also provides the opportunity to review the workflow diagrams to ensure that all required resources have an appropriate mapping in Moab.

Q.3.3 Defining Job Groups

With the completed workflow diagrams, it is possible to identify functional groups. Functional groups can be demarcated along many different lines. The most important consideration is whether the jobs will need to share information amongst themselves or represent internal dependencies. These functional groups are known as job groups. Job groups share a common name space wherein variables can be created, allowing for communication among the different processes.

Often job groups will be defined along business process lines. In addition, subgroups can also be defined to allow the workflow to be viewed in more manageable sections.

Q.3.4 Setting the Schedule

As was previously mentioned, Moab views the world as a set of resources that can be scheduled. Scheduling of resources is accomplished by creating a reservation. Some reservations are created automatically, such as when a job is scheduled for launch. Other reservations can be created manually by the administrator. Manual reservations can either be static, meaning they are part of Moab’s configuration (moab.cfg) or ad hoc, created as needed via the command line or the MCM tool.

Q.3.4.1 Determining Scheduling Requirements

Within any data center, some jobs are high priority, while others are low priority. It is necessary to make sure resources are available to the high priority jobs, especially when these jobs are part of an SLA or part of a time-critical business process. Moab supports a number of configuration options and parameters to support these scheduling goals.

In this section we will look at the situation where part of the cluster needs to be reserved for certain processes during the day. Figure 3 shows a ten node cluster and its daily scheduling requirements. There are four service areas that must have sole access to their associated nodes during specific times of the day. All other jobs are allowed to float freely among the free processors. This example is typical of many data centers where workload is dependent on the time of day or day of week.

Figure 3: Daily standing reservations
It is important to have a clear understanding of what jobs require hard reservations such as the ones shown here and those that are more forgiving. A particular strength of Moab is its ability to schedule in a dynamic environment while still being able to support many scheduling goals. The standing reservation approach limits the ability for Moab to dynamically schedule, as it limits what can be scheduled on certain nodes during certain times. Cycles lost in a standing reservation because the jobs for which the reservation was made do not use the entire time block cannot be reclaimed by Moab. However, standing reservations are a powerful way to guarantee resources for mission-critical processes and jobs.

Q.3.4.2 Creating Standing Reservations

If it is determined that standing reservations are appropriate, they must be created in Moab. Standing reservations are created in the \texttt{moab.cfg} file. However, before the standing reservation can be defined, a quality of service (QoS) should be created that will specify which users are allowed to submit jobs that will run in the new standing reservations.

QoS is a powerful concept in Moab, often used to control access to resources, instigate different billing rates, and control certain administrative privileges. Additional information on QoS and standing reservations is available.

Like the standing reservations, QoS is defined in the \texttt{moab.cfg} file. \textit{Example 16} shows how to create the four QoS options that are required to implement the standing reservations shown in Figure 3. This example assumes the noted users (\texttt{admin, datacenter, web, apache} and \texttt{customer1}) have been previously defined in the \texttt{moab.cfg} file.

\textbf{Example 16: Sample QoS configuration}

\begin{verbatim}
QOSCFG[nightly] QFLAGS=DEADLINE,TRIGGER
QOSCFG[nightly] MEMBERULIST=admin, datacenter
QOSCFG[daily] QFLAGS=DEADLINE,TRIGGER
QOSCFG[daily] MEMBERULIST=admin, datacenter
QOSCFG[web] MEMBERULIST=web, apache
QOSCFG[contract1] QFLAGS=DEADLINE MEMBERULIST=customer1
\end{verbatim}
Example 16 shows that four QoS options have been defined. The first and second QoS (nightly and daily) have special flags that allow the jobs to contain triggers and to have a hard deadline. All the QoS options contain a user list of those users who have access to submit to the QoS. Notice that multiple configuration lines are allowed for each QoS.

Once the QoS options have been created, the associated standing reservations must also be created. Example 17 shows how this is done in the moab.cfg file.

Example 17: Sample standing reservation configuration

| SRCFG[dc_night]   | STARTTIME=00:00:00 ENDTIME=04:00:00 |
| SRCFG[dc_day]     | HOSTLIST=Node0[1-8]$ QOSLIST=nightly |
| SRCFG[dc_night]   | STARTTIME=06:00:00 ENDTIME=19:00:00 |
| SRCFG[dc_day]     | HOSTLIST=Node0[1-5]$ QOSLIST=daily |
| SRCFG[webserv1]   | STARTTIME=08:00:00 ENDTIME=16:59:59 |
| SRCFG[webserv1]   | HOSTLIST=Node0[7-8]$ QOSLIST=web |
| SRCFG[webserv2]   | STARTTIME=06:00:00 ENDTIME=19:00:00 |
| SRCFG[webserv2]   | HOSTLIST=Node09 QOSLIST=web |
| SRCFG[webserv3]   | STARTTIME=00:00:00 ENDTIME=23:59:59 |
| SRCFG[webserv3]   | HOSTLIST=Node10 QOSLIST=web |
| SRCFG[cust1]      | STARTTIME=17:00:00 ENDTIME=23:59:59 |
| SRCFG[cust1]      | HOSTLIST=NODE0[6-8]$ QOSLIST=contract1 |

Example 17 shows the creation of the different standing reservations. Each of the standing reservations has a start and end time, as well as the host list and the associated QoS. Three separate standing reservations were used for the web services because of the different nodes and time period sets. This setup works fine here because Web services is going to be small, serial jobs. In other circumstances, a different reservation structure would be needed for maximum performance.

Q.3.4.3 Submitting a Job to a Standing Reservation

It is not uncommon for a particular user to have access to multiple QoS options. For instance, Example 16 shows the user datacenter has access to both the nightly and daily QoS options. Consequently, it is necessary to denote which QoS option is to be used when a job is submitted.

Example 18: Specifying a QoS option at job submission

```
> msub -l qos=nightly nightly.job.cmd
```

In Example 18, the script nightly.job.cmd is being submitted using the QoS option nightly. Consequently, it will be able to run using the nodes reserved for that QoS option in Example 17.

Q.3.5 Workflow Dependencies

With the different standing reservations and associated QoSs configured in Moab, the process of converting the workflow can continue. The next steps are to convert the compute jobs and build the dependency tree to support the workflow.

Q.3.5.1 Converting Compute Jobs

Most compute jobs will require few, if any, changes to run under the new paradigm. All jobs will be submitted to Moab running on the head node. (See Figure 2.) Moab will then schedule the job based on a number of criteria, including user, QoS, standing reservations, dependencies and other job requirements. While the scheduling is dynamic, proper use of QoS and other policies will ensure the desired execution of jobs on the cluster.

Jobs may read and write files from a network drive. In addition, Moab will return all information written to
**Example 19: Staging of** `STDOUT` **and** `STDERR` **in files denoted by the Job ID to the user who submitted the job.**

```bash
> echo env | msub
40278
> ls -l
-rw------- 1 user1 user1  0 2007-04-03 13:28 STDIN.e40278
-rw------- 1 user1 user1 683 2007-04-03 13:28 STDIN.o40278
```

**Example 19** shows the created output files. The user `user1` submitted the `env` program to Moab, which will return the environment of the compute node on which it runs. As can be seen, two output files are created: `STDIN.e40278` and `STDIN.o40278` (representing the output on `STDERR` and `STDOUT`, respectively). The Job ID (40278) is used to denote which output files belong to which job. The first part of the file names, `STDIN`, is the name of the job script. In this case, because the job was submitted to `msub`'s `STDIN` via a pipe, `STDIN` was used.

**Q.3.5.2 Introduction to Triggers**

Triggers are one way to handle dependencies in Moab. This section introduces triggers and several of their key concepts. Additional trigger information is available.

In its simplest form, a trigger has three basic parts:

1. An object
2. An event
3. An action

The concept of resources in Moab has already been introduced. In addition to resources, Moab is aware of a number of other entities, including the scheduler, resource managers, nodes and jobs. Each of these are represented in Moab as an object. Triggers may be attached to any of these object types. Also an object instance can have multiple triggers attached to it.

In addition to an object, each trigger must have an event specified. This event determines when the trigger's action will take place. In Moab terminology, this is known as an event type. Some popular event types include `cancel`, `create`, `end`, `start` and `threshold`.

An action consists of two parts: (1) the action type and (2) the action command. The action type specifies what type of an action is to occur. Some examples include `exec`, `internal` and `mail`. The format of the action command is determined by the action type. For example, `exec` requires a command line, while `mail` needs an email address.

While triggers can be placed on any object type within Moab, the job object is the most useful for creating workflows. A trigger can be attached to an object at submission time through the use of `msub` and the `-l` flag.

**Example 20: Submitting a job with a basic trigger from the command line**

```bash
> msub -l trig=AType=exec&EType=start&Action="job_check.pl" Job.cmd
```

In **Example 20**, the job `Job.cmd` is submitted with one trigger. This trigger will execute the `job_check.pl` script when the compute job starts on the compute node. It is important to note that all triggers are run on the head node, even those attached to compute jobs.

Often, triggers are attached to special jobs known as system jobs. System jobs are simply an instantiation of a job object that does not require any resources by default. In addition, it also runs on the head node.

**Example 21: Creating a system job with a trigger from the command line**
Just like normal jobs, system jobs can serve as a job group. In other words, when forming job groups, one is associating one job with another. The job to which all others attach is known as the job group, and its variable name space can be used as a common repository for each of the child jobs.

Example 22: Creating a job group and associating two jobs with the group

| msub -l flags=NOREOURCES, trig=AType=exec & EType=start & Action="job_check.pl" Job |
| echo true | msub -N MyJobGroup -l flags=NOREOURCES |
| msub -W x=JGroup:MyJobGroup Job1 |
| msub -W x=JGroup:MyJobGroup Job2 |

In Example 22, a system job is created and given the name MyJobGroup to simplify later use of the job group. Then two compute jobs, Job1 and Job2, are submitted. They are made part of the MyJobGroup job group, meaning they will have access to the variable name space of the first job.

As a security measure, only jobs submitted to QoS with the trigger flag can have triggers attached. An example of the configuration for this can be seen in Example 16.

Another important point with triggers is that they are event-driven. This means that they operate outside of the normal Moab batch scheduling process. During each scheduling iteration, Moab evaluates all triggers to see if their event has occurred and if all dependencies are fulfilled. If this is the case, the trigger is executed, regardless of the priority of the job to which it is attached. This provides the administrator with another degree of control over the system.

By combining compute jobs, system jobs, and triggers, one can build fairly complex workflows. The next few sections cover additional information on how to map the previously created workflow diagram to these concepts.

Q.3.5.3 Internal Dependencies

Internal dependencies are those that can be fulfilled by other jobs within the workflow. For example, if Job1 must complete before Job2, then Job1 is an internal dependency of Job2. Another possibility is that Job1 may also stage some data that Job3 requires, which is another type of internal dependency.

Internal dependencies are often handled through variables. Each trigger can see all the variables in the object to which it is attached. In the case of the object being a job, the trigger can also see the variables in any of the job’s job group hierarchy. Triggers may require that certain variables simply exist or have a specific value in order to launch. Upon completion, triggers can set variables depending on the success or failure of the operation. In some instances, triggers are able to consume variables that already exist when they launch, removing them from the name space.

Figure 4: Jobs, job group and variables

Figure 4 shows a simple setup where there is a job group (MyJobGroup) with two compute jobs associated with it (Job1 and Job2). In addition to their own variables, both Job1 and Job2 can see var1 and var2 because they are in MyJobGroup. However, Job1 cannot see var5 and var6, nor can Job2 see var3 and var4.
Also, MyJobGroup cannot see var3, var4, var5 and var6. One can only see up the hierarchy tree.

Notice that var3 has a caret symbol (^) attached to the front. This means that when Job1 completes, var3 is going to be exported to the job group, MyJobGroup in this case. This can be seen in Figure 5, which shows the state of things after Job1 has completed.

Figure 5: Jobs, job group and variables after complete

With the completion of Job1, var3 has been exported to MyJobGroup. However, var4, which was not set to export, has been destroyed. At this point, Job2 can now see var3. If a trigger attached to Job2 required var3, it would now be able to run because its dependency is now fulfilled.

This is a very simple example of how variables can be used in conjunction with triggers. Detailed information on the interaction between triggers and variables is available.

Q.3.5.4 External Dependencies

There are instances where dependencies must be fulfilled by entities external to the cluster. This may occur when external entities are staging in data from a remote source or when a job is waiting for the output of specialized test equipment. In instances such as these, Moab provides a method for injecting a variable into a job's name space from the command line. The mjobctl command is used.

Example 23: Injecting variables into objects from the command line

> mjobctl -m var=newvar1=1 MyJobGroup

In this example, the variable newvar1 with a value of 1 is injected into MyJobGroup, which was created in Example 22. This provides a simple method for allowing external entities to notify Moab when an external dependency has been fulfilled.

Q.3.5.5 Cascading Triggers

Another approach that can be used when converting a workflow is the idea of dynamic workflow. A dynamic workflow is one in which sections of the workflow are created on-the-fly through the use of triggers or other means. The technique of using triggers to accomplish this is known as cascading triggers. This is where triggers dynamically create other jobs and triggers, which handle parts of the overall workflow.

Cascading triggers reduces the number of triggers and jobs in the system, thereby reducing administrative overhead. While not the perfect solution in every case, they provide a great tool for those using Moab in the data center. See the Case Studies for an example of their benefits.

Q.4 Dynamic Workload

A dynamic workload is a non-static workload that changes over the course of a day, week, month or year. Moab supports this through the use of traditional HPC techniques. As jobs are submitted to Moab, they are evaluated for execution. This evaluation is based on a number of factors including policies, QoS, and standing reservations and assigns a priority to the job. These priorities are used when scheduling the job.

Where possible, Moab will backfill empty sections of the schedule with lower priority jobs if it will not affect
the higher priority jobs. This scheduling occurs every scheduling iteration, allowing Moab to take advantage of changing situations. Because of the dynamic nature of Moab scheduling, it is able to handle the changing workload in a data center—providing services and resources for ad hoc jobs, as well as the normal, static workflow.

Q.5 Supporting SLAs and Other Commitments

When SLAs are in place, it is requisite that the workflow support these agreements. Contracted work must be accomplished accurately and on time. A number of Moab policies exist to ensure these goals can be met.

Moab has a number of different credential types which allow for the grouping of users in a large number of ways. Access rights can be given to each credential, and limits on system usage for each credential can be defined. Users are allowed to have multiple credentials, thereby providing a rich set of access control mechanisms to the administrator.

Moab's scheduling algorithms are customizable on a number of different levels, allowing administrators to determine when and where certain jobs are allowed to run. The dynamic nature of the scheduling engine allows Moab to react to changing circumstances.

Moab's scheduling engine takes multiple circumstances and priorities into consideration when ordering jobs. Because Moab considers the future when scheduling, jobs may be scheduled to start at or complete by specific times. In addition, resources can be statically or dynamically allocated for jobs submitted via certain credentials. Policies can be put in place to describe Moab behavior with jobs that are not abiding by the imposed restrictions.

In combination, all these features allow the administrator to customize Moab behavior to perfectly match their data center's needs.

Additional resources:
- Prioritizing Jobs and Allocating Resources
- Controlling Resource Access
- Managing Shared Resources
- Job Administration
- Node Administration

Q.CS Case Studies

This section illustrates Moab functionality in the data center. All company names presented in this section are fictitious.

Q.CS.1 Cascading Triggers

A data collection company handles a large amount of incoming information each day. New information is available every five minutes for processing. The information is processed by their cluster in a series of steps. Each step consolidates the information from the previous steps and then generates a report for the given time period, as shown in what follows:

- **Every 5 Minutes** – Information gathered and "5 Minute" report generated
- **Every 15 Minutes** – All new "5 Minute" data handled
- **Every Hour** – All new "15 Minute" data handled
- **Every 3 Hours** – All new "1 Hour" data handled
- **Every Day** – All new "3 Hour" data handled

The original approach was to create the entire workflow structure for the entire day with a standing trigger that fired every night at midnight. However, this produced a large number of triggers that made management of the system more difficult. For every "5 Minute" task requiring a trigger, 288 triggers are required. This quickly made the output of `mdiag -T` very difficult to parse for a human, as multiple tasks, and therefore triggers, were required for each step.

To address this issue, the cascading triggers approach was adopted. With this approach, triggers were only created as needed. For example, each "3 Hour" trigger created its underlying "1 Hour" triggers when it was its time to run.
The resulting reduction of triggers in the system was very impressive. Figure 6 shows the difference between the two approaches in the number of triggers when only one trigger is needed per step. The difference is even greater when more than one trigger is needed per step.

Figure 6: Cascading triggers comparison graphs

The charts show the number of triggers over the full day that would be required if all jobs were generated at the beginning of the day versus a dynamic approach where jobs and their associated triggers were created only as needed. The difference in the two approaches is clear.
# SCHEDCFG Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOWMULTICOMPUTE</td>
<td>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.</td>
</tr>
<tr>
<td>DISABLEPERJOBNODESETS</td>
<td>Disables a job's ability to override the system specified node set. See 13.3 Resource Manager Extensions for more information.</td>
</tr>
<tr>
<td>FASTGROUPLOOKUP</td>
<td>Moab will use the system call <code>getgrouplist</code> to gather group information. This can significantly improve performance on some LDAP systems.</td>
</tr>
<tr>
<td>FASTRSVSTARTUP</td>
<td>Speeds up start time if there are existing reservations.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Without flag:</td>
</tr>
<tr>
<td></td>
<td>Reservations:</td>
</tr>
<tr>
<td></td>
<td>brian.1x1 User -00:12:52 -&gt; INFINITY ( INFINITY)</td>
</tr>
<tr>
<td></td>
<td>Blocked Resources@-00:00:02 Procs: 5/6 (83.33%) Mem: 0/5000 (0.00%)</td>
</tr>
<tr>
<td></td>
<td>Blocked Resources@00:04:58 Procs: 6/6 (100.00%) Mem: 0/5000 (0.00%)</td>
</tr>
<tr>
<td></td>
<td>m.2x1 Job:Running -00:00:02 -&gt; 00:04:58 (00:05:00)</td>
</tr>
<tr>
<td></td>
<td>Jobs: m.2</td>
</tr>
<tr>
<td></td>
<td>With flag:</td>
</tr>
<tr>
<td></td>
<td>Reservations:</td>
</tr>
<tr>
<td></td>
<td>brian.1x1 User -00:00:15 -&gt; INFINITY ( INFINITY)</td>
</tr>
<tr>
<td></td>
<td>Blocked Resources@-00:00:02 Procs: 6/6 (100.00%) Mem: 0/5000 (0.00%)</td>
</tr>
<tr>
<td></td>
<td>Blocked Resources@00:04:58 Procs: 6/6 (100.00%) Mem: 0/5000 (0.00%)</td>
</tr>
<tr>
<td></td>
<td>m.1x1 Job:Running -00:00:02 -&gt; 00:04:58 (00:05:00)</td>
</tr>
<tr>
<td></td>
<td>Jobs: m.1</td>
</tr>
<tr>
<td>FILELOCKHA</td>
<td>This is a High Availability feature. FILELOCKHA prevents scheduling conflicts between multiple Moab servers.</td>
</tr>
<tr>
<td>JOBSUSERSVWALLTIME</td>
<td>Allows jobs submitted without a walltime request or default walltime</td>
</tr>
</tbody>
</table>
received from a class or queue but with an ADVRES:reservation 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.

| NORMALIZETASKDEFINITIONS | 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:
  
  \[
  \text{mshow -a -w mintasks=1@procs:4+mem:4096}
  \]
  
  It is changed to this:
  
  \[
  \text{mshow -a -w mintasks=4@procs:1+,mem:1024,tnp=4}
  \] |

| 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. |

| STRICTSPOOLDIRPERMISSIONS | Enforces at least a 511 permission on the Moab spool directory. |

| USELOCALUSERGROUP | This enables the mapping of a job’s GROUP to become the user’s GROUP upon receiving a job from a remote Moab instance. The job is then in the GROUP of the user who owns the job during execution. If not enabled, the job is rejected if the GROUP is not valid on the executing Moab instance.

- When the source Moab instance queries the destination job status, the job reflects the GROUP value of the job on the destination.