Moab Workload Manager

Administrator Guide 7.2.9

September 2014

© 2014 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.

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.

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

Scan to open online help

Contents

Welcome ..... **xi**
Moab overview xi

1.0 Philosophy **1**
1.1 Value of a Batch System 1
1.2 Philosophy and Goals 2
1.3 Workload 3

2.0 Installation and Initial Configuration **7**
2.1 Hardware and Software Requirements 7
2.2 Installing Moab 8
2.3 Connecting Moab to MongoDB 15
2.4 Upgrading Moab 17
2.5 Initial Moab Configuration 20
2.6 Initial Moab Testing 22

3.0 Scheduler Basics **25**
3.1 Layout of Scheduler Components 25
3.2 Scheduling Environment 27
3.2.1 Scheduling Dictionary 33
3.3 Scheduling Iterations and Job Flow 40
3.4 Configuring the Scheduler 42
3.5 Credential Overview 45
3.5.1 Job Attributes/Flags Overview 72

4.0 Scheduler Commands **79**
4.1 Status Commands 82
4.2 Job Management Commands 83
4.3 Reservation Management Commands 83
4.4 Policy/Configuration Management Commands 84
4.5 End-user Commands 84
4.6 Commands 85
checkjob 85
checknode 94
mcredctl 98
mdiag 101
mdiag -a 104
mdiag -b 105
mdiag -c 105

mdiag -f 109
mdiag -g 111
mdiag -j 111
mdiag -n 112
mdiag -t 118
mdiag -p 118
mdiag -q 121
mdiag -r 121
mdiag -R 125
mdiag -S 129
mdiag -s 130
mdiag -T 130
mdiag -u 132
mjobctl 132
mnodectl 149
moab 155
mrmctl 156
mrsvctl 158
mschedctl 184
mshow 191
mshow -a 192
mshow -a 201
msub 203
Applying the msub Submit Filter 218
Sample Submit Filter Script 219
Submitting Jobs via msub in XML 219
mvcctl 223
mvmctl 228
showbf 232
showq 235
showhist.moab.pl 244
showres 249
showstart 253
showstate 256
showstats 257
showstats -f 268
TIMESPEC 270
4.6.1 Deprecated commands 271
canceljob 271
changeparam 272
diagnose 272
releasehold 273
releaseres 274
resetstats 275

runjob 275
sethold 276
setqos 277
setres 278
setspri 282
showconfig 283

5.0 Prioritizing Jobs and Allocating Resources 285
5.1 Job Prioritization 285
5.1.1 Priority Overview 285
5.1.2 Job Priority Factors 286
5.1.3 Fairshare Job Priority Example 296
5.1.4 Common Priority Usage 298
5.1.5 Prioritization Strategies 300
5.1.6 Manual Job Priority Adjustment 301
5.2 Node Allocation Policies 301
5.3 Node Access Policies 309
5.4 Node Availability Policies 311

6.0 Managing Fairness - Throttling Policies, Fairshare, and Allocation Management 319
6.1 Fairness Overview 319
6.2 Usage Limits/Throttling Policies 322
6.3 Fairshare 340
6.3.1 Sample FairShare Data File 353
6.4 Charging and Allocation Management 354
6.5 Charging a Workflow 369
6.6 NAMI Queuing 372

7.0 Controlling Resource Access - Reservations, Partitions, and QoS Facilities 375
7.1 Advance Reservations 375
7.1.1 Reservation Overview 375
7.1.2 Administrative Reservations 379
7.1.3 Standing Reservations 381
7.1.4 Reservation Policies 382
7.1.5 Configuring and Managing Reservations 386
7.1.6 Personal Reservations 417
7.2 Partitions 419
7.3 Quality of Service (QoS) Facilities 423

8.0 Optimizing Scheduling Behavior – Backfill and Node Sets 433
8.1 Optimization Overview 433
8.2 Backfill 434
8.3 Node Set Overview 439

9.0 Evaluating System Performance - Statistics, Profiling, Testing, and Simulation 445
9.1 Moab Performance Evaluation Overview 445
9.2 Accounting: Job and System Statistics 445
9.3 Testing New Versions and Configurations 447
9.4 Answering What If? Questions with the Simulator 448

10.0 General Job Administration 449
10.1 Job Holds 449
10.2 Job Priority Management 451
10.3 Suspend/Resume Handling 451
10.4 Checkpoint/Restart Facilities 452
10.5 Job Dependencies 452
10.6 Job Defaults and Per Job Limits 454
10.7 General Job Policies 455
10.8 Using a Local Queue 457
10.9 Job Deadlines 460
10.10 Job Arrays 462

11.0 General Node Administration 469
11.1 Node Location 470
11.2 Node Attributes 473
11.3 Node Specific Policies 483
11.4 Managing Shared Cluster Resources (Floating Resources) 484
11.5 Managing Node State 488
11.6 Managing Consumable Generic Resources 490
11.7 Enabling Generic Metrics 492
11.8 Enabling Generic Events 496

12.0 Resource Managers and Interfaces 503
12.1 Resource Manager Overview 504
12.2 Resource Manager Configuration 507
12.3 Resource Manager Extensions 537
12.3.1 PBS Resource Manager Extensions 563
12.4 Adding New Resource Manager Interfaces 565
12.5 Managing Resources Directly with the Native Interface 566
12.6 Utilizing Multiple Resource Managers 576
12.7 License Management 578
12.8 Resource Provisioning 580
12.9 Intelligent Platform Management Interface 581
12.10 Resource Manager Translation 584

13.0 Troubleshooting and System Maintenance 585
13.1 Internal Diagnostics/Diagnosing System Behavior and Problems 585
13.2 Logging Facilities 588
13.3 Object Messages 597

13.4 Notifying Administrators of Failures 599
13.5 Issues with Client Commands 600
13.6 Tracking System Failures 601
13.7 Problems with Individual Jobs 603
13.8 Diagnostic Scripts 604

14.0 Improving User Effectiveness 607
14.1 User Feedback Loops 607
14.2 User Level Statistics 608
14.3 Enhancing Wallclock Limit Estimates 608
14.4 Job Start Time Estimates 608
14.5 Providing Resource Availability Information 610
14.6 Collecting Performance Information on Individual Jobs 610

15.0 Cluster Analysis, Testing, and Simulation 611
15.1 Testing New Releases and Policies 611
15.2 Testing New Middleware 615
15.3 Simulations 618
15.3.1 Configuring Simulation 618
15.3.2 Configuring Resources for Simulation 620
15.3.3 Workload Event Format 621
15.3.4 Interactive Simulation Tutorial 631
15.3.4.1 Checking the Queue Status 632
15.3.4.2 Determining Why Jobs Are Not Running 633
15.3.4.3 Controlling Iterations 633
15.3.4.4 Managing Reservations Applying to the Queue 635
15.3.4.5 Verifying Fair Scheduling 640
15.3.4.6 Taking the System Down for Maintenance 640

16.0 Green computing 643
16.1 About green computing 643
16.2 How-to's 644
16.2.1 Enabling green computing 644
16.2.2 Deploying Adaptive Computing IPMI scripts 647
16.2.3 Choosing which nodes Moab powers on or off 648
16.2.4 Adjusting green pool size 649
16.2.5 Handling power-related events 649
16.2.6 Maximizing scheduling efficiency 650
16.2.7 Troubleshooting green computing 651

17.0 Object triggers 655
17.1 About object triggers 655
17.2 How-to's 657
17.2.1 Creating a trigger 658
17.2.2 Using a trigger to send email 662

17.2.3 Using a trigger to execute a script 663
17.2.4 Using a trigger to perform internal Moab actions 663
17.2.5 Requiring an object threshold for trigger execution 664
17.2.6 Enabling job triggers 664
17.2.7 Modifying a trigger 665
17.2.8 Viewing a trigger 666
17.2.9 Checkpointing a trigger 667
17.3 References 667
17.3.1 Job triggers 667
17.3.2 Node triggers 668
17.3.3 Reservation triggers 670
17.3.4 Resource manager triggers 671
17.3.5 Scheduler triggers 672
17.3.6 Threshold triggers 673
17.3.7 Trigger components 674
17.3.8 Trigger exit codes 682
17.3.9 Node maintenance example 682
17.3.10 Environment creation example 683
17.4 Trigger variables 684
17.4.1 About trigger variables 684
17.4.2 How-to's 685
17.4.2.1 Setting and receiving trigger variables 685
17.4.2.2 Externally injecting variables into job triggers 686
17.4.2.3 Exporting variables to parent objects 686
17.4.2.4 Requiring variables from generations of parent objects 687
17.4.2.5 Requesting name space variables 687
17.4.3 References 688
17.4.3.1 Dependency trigger components 688
17.4.3.2 Trigger variable comparison types 689
17.4.3.3 Internal variables 689

18.0 Miscellaneous 691
18.1 User Feedback Overview 691
18.2 Enabling High Availability Features 692
18.3 Malleable Jobs 694
18.4 Identity Managers 695
18.5 Generic System Jobs 699
18.6 Implementing Guaranteed Start Time 702

19.0 Database Configuration 703
19.1 SQLite3 703
19.2 Connecting to a MySQL Database with an ODBC Driver 704
19.3 Connecting to a PostgreSQL Database with an ODBC Driver 707
19.4 Migrating Your Database to Newer Versions of Moab 709
19.5 Importing Statistics from stats/DAY.* to the Moab Database 714

20.0 Accelerators **715**
20.1 Scheduling GPUs 715
20.2 Using GPUs with NUMA 716
20.3 NVIDIA GPUs 717
20.4 GPU Metrics 719
20.5 Intel® Xeon Phi™ Coprocessor Configuration 721
20.6 Intel® Xeon Phi™ Co-processor Metrics 725

21.0 VMs **727**
21.1 Policy-based VM Migration 727
21.2 Overcommit Factor and Threshold 729
21.3 Overutilization Migration 731
21.4 Green Migration and Consolidation 731

22.0 Workload-Driven Cloud Services **733**
22.1 About workload-driven cloud services 733
22.2 Tasks 738
22.2.1 Enabling cloud services 738
22.2.2 Creating a generic system job 739
22.2.3 Creating a cloud workflow 740
22.2.4 Creating a service 743
22.2.5 Canceling a service 744
22.3 References 745
22.3.1 Cloud-specific job template attributes 745
22.3.2 Generic system job trigger requirements 747
22.3.3 VM service example 748

23.0 Preemption **751**
23.1 About preemption 751
23.2 Preemption tasks 752
23.2.1 Canceling jobs with preemption 752
23.2.2 Checkpointing jobs with preemption 755
23.2.3 Requeueing jobs with preemption 757
23.2.4 Suspending jobs with preemption 760
23.2.5 Using owner preemption 763
23.2.6 Using QoS preemption 767
23.3 Preemption references 768
23.3.1 Manual preemption commands 768
23.3.2 Preemption flags 769
23.3.3 PREEMPTPOLICY types 770
23.3.4 Simple example of preemption 771
23.3.5 Testing and troubleshooting preemption 774

24.0 Job templates **777**
24.1 About job templates 777

24.2 Job template how-to's 778
24.2.1 Creating job templates 778
24.2.2 Viewing job templates 779
24.2.3 Applying templates based on job attributes 779
24.2.4 Requesting job templates directly 780
24.2.5 Creating workflows with job templates 781
24.3 Job template references 782
24.3.1 Job template extension attributes 782
24.3.2 Job template matching attributes 793
24.3.3 Job template examples 794
24.3.4 Job template workflow examples 795

25.0 Appendices 797
Appendix A: Moab Parameters 797
Appendix B: Multi-OS Provisioning 949
Appendix D: Adjusting Default Limits 967
Appendix E: Security 971
Appendix G: Integrating Other Resources with Moab 979
Compute Resource Managers 980
Moab-TORQUE Integration Guide 980
TORQUE/PBS Integration Guide - RM Access Control 983
TORQUE/PBS Config - Default Queue Settings 983
Moab-SLURM Integration Guide 984
Installation Notes for Moab and TORQUE for Cray 988
Provisioning Resource Managers 1005
Validating an xCAT Installation for Use with Moab 1006
Hardware Integration 1008
Moab-NUMA Integration Guide 1008
Appendix H: Interfacing with Moab (APIs) 1012
Appendix I: Considerations for Large Clusters 1016
Appendix J: Configuring Moab as a Service 1021
Appendix K: Migrating from 3.2 1021
Appendix R: Node Allocation Plug-in Developer Kit 1024
Appendix S: Scalable Systems Software Specification 1031
Scalable Systems Software Job Object Specification 1031
Scalable Systems Software Resource Management and Accounting Protocol (SSSRMAP) Message Format 1064
Scalable Systems Software Node Object Specification 1088
Scalable Systems Software Resource Management and Accounting Protocol (SSSRMAP) Wire Protocol 1097
Appendix W: Moab Resource Manager Language Interface Overview 1113
W.1 Moab Resource Manager Language Data Format 1113
W.2 Managing Resources with SLURM 1122
W.3 Moab RM Language Socket Protocol Description 1132
SCHEDCFG flags 1136

Welcome

Welcome to the **Moab Workload Manager 7.2.9** *Administrator Guide*. This guide is intended for Moab system administrators and users.

The following sections will help you quickly get started:

- **Moab overview** on page xi: Gives an overview about Moab basics.
- **Philosophy** on page 1: Explains the value of using Moab and the philosophy behind what Moab is designed to do.
- **Installing Moab** on page 8: Provides instructions about how to install Moab.
- **Initial Moab Configuration** on page 20: Explains how to configure and set up Moab.

Moab overview

Moab Workload Manager is a highly advanced scheduling and management system designed for clusters, and on-demand/utility computing systems. At a high level, Moab applies site policies and extensive optimizations to orchestrate jobs, services, and other workload across the ideal combination of network, compute, and storage resources. Moab enables true adaptive computing allowing compute resources to be customized to changing needs and failed systems to be automatically fixed or replaced. Moab increases system resource availability, offers extensive cluster diagnostics, delivers powerful QoS/SLA features, and provides rich visualization of cluster performance through advanced statistics, reports, and charts.

Moab works with virtually all major resource management and resource monitoring tools. From hardware monitoring systems like IPMI to provisioning systems and storage managers, Moab takes advantage of domain expertise to allow these systems to do what they do best, importing their state information and providing them with the information necessary to better do their job. Moab uses its global information to coordinate the activities of both resources and services, which optimizes overall performance in-line with high-level mission objectives.

Related topics

- **Welcome** on page xi

1.0 Philosophy

The scheduler's purpose is to optimally use resources in a convenient and manageable way. System users want to specify resources, obtain quick turnaround on their jobs, and have reliable resource allocation. On the other hand, administrators want to understand both the workload and the resources available. This includes current state, problems, and statistics—information about what is happening that is transparent to the end-user. Administrators need an extensive set of options to enable management enforced policies and tune the system to obtain desired statistics.

There are other systems that provide batch management; however, Moab is unique in many respects. Moab matches jobs to nodes, dynamically reprovisions nodes to satisfy workload, and dynamically modifies workload to better take advantage of available nodes. Moab allows sites to fully visualize cluster and user behavior. It can integrate and orchestrate resource monitors, databases, identity managers, license managers, networks, and storage systems, thus providing a cohesive view of the cluster—a cluster that fully acts and responds according to site mission objectives.

Moab can dynamically adjust security to meet specific job needs. Moab can create real and virtual clusters on demand and from scratch that are custom-tailored to a specific request. Moab can integrate visualization services, web farms and application servers. Moab maintains complete accounting and auditing records, exporting this data to information services on command, and even providing professional billing statements to cover all used resources and services.

Moab provides user- and application-centric web portals and powerful graphical tools for monitoring and controlling every conceivable aspect of a cluster's objectives, performance, workload, and usage. Moab is unique in its ability to deliver a powerful user-centric cluster with little effort. Its design is focused on ROI, better use of resources, increased user effectiveness, and reduced staffing requirements.

This chapter contains these sections:

- **Value of a Batch System** on page 1
- **Philosophy and Goals** on page 2
- **Workload** on page 3

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:

- Traffic Control
- Mission Policies
- Optimizations

Traffic Control

A scheduler must prevent jobs from interfering. If jobs contend for resources, cluster performance decreases, job execution is delayed, and jobs may fail. Thus, the scheduler tracks resources and dedicates requested resources to a particular job, which prevents use of such resources by other jobs.

Mission Policies

Clusters and other HPC platforms typically have specific purposes; to fulfill these purposes, or mission goals, there are usually rules about system use pertaining to who or what is allowed to use the system. To be effective, a scheduler must provide a suite of policies allowing a site to *map* site mission policies into scheduling behavior.

Optimizations

The compute power of a cluster is a limited resource; over time, demand inevitably exceeds supply. Intelligent scheduling decisions facilitate higher job volume and faster job completion. Though subject to the constraints of the traffic control and mission policies, the scheduler must use whatever freedom is available to maximize cluster performance.

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.

Management Goals

A manager must ensure that a cluster fulfills the purpose for which it was purchased, so a manager must deliver cycles to those projects that are most critical to the success of the funding organizations. Management tasks to fulfill this role may include the following:

- Define cluster mission objectives and performance criteria
- Evaluate current and historical cluster performance
- Instantly graph delivered service

Administration Goals

An administrator must ensure that a cluster is effectively functioning within the bounds of the established mission goals. Administrators translate goals into cluster policies, identify and correct cluster failures, and train users in best practices. Given these objectives, an administrator may be tasked with each of the following:

- Maximize utilization and cluster responsiveness
- Tune fairness policies and workload distribution
- Automate time-consuming tasks
- Troubleshoot job and resource failures
- Instruct users of available policies and in their use regarding the cluster
- Integrate new hardware and cluster services into the batch system

End-user Goals

End-users are responsible for learning about the resources available, the requirements of their workload, and the policies to which they are subject. Using this understanding and the available tools, they find ways to obtain the best possible responsiveness for their own jobs. A typical end-user may have the following tasks:

- Manage current workload
- Identify available resources
- Minimize workload response time
- Track historical usage
- Identify effectiveness of prior submissions

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:

- Batch Workload
- Interactive Workload
- Calendar Workload
- Service Workload

Batch Workload

Batch workload is characterized by a *job* command file that typically describes all critical aspects of the needed compute resources and execution environment. With a batch job, the job is submitted to a job queue, and is run somewhere on the cluster as resources become available. In most cases, the submitter will submit multiple batch jobs with no execution time constraints and will process the job results as they become available.

Moab can enforce rich policies defining how, when, and where batch jobs run to deliver compute resources to the most important workload and provide general SLA guarantees while maximizing system utilization and minimizing average response time.

Interactive Workload

Interactive workload differs from batch in that requestors are interested in immediate response and are generally waiting for the interactive request to be executed before going on to other activities. In many cases, interactive submitters will continue to be *attached* to the interactive job, routing keystrokes and other input into the job and seeing both output and error information in real-time. While interactive workload may be submitted within a job file, commonly, it is routed into the cluster via a web or other graphical terminal and the end-user may never even be aware of the underlying use of the batch system.

For managing interactive jobs, the focus is usually on setting aside resources to guarantee immediate execution or at least a minimal wait time for interactive jobs. Targeted service levels require management when mixing batch and interactive jobs. Interactive and other jobs types can be dynamically steered in terms of what they are executing as well as in terms of the quantity of resources required by the application. Moab can apply dynamic or malleable job facilities to dynamically grow and shrink jobs as needed to meet these changing constraints.

Calendar Workload

Calendar workload must be executed at a particular time and possibly in a regular periodic manner. For such jobs, time constraints range from flexible to rigid. For example, some calendar jobs may need to complete by a certain time, while others must run exactly at a given time each day or each week.

Moab can schedule the future and can thus guarantee resource availability at needed times to allow calendar jobs to run as required. Furthermore, Moab provisioning features can locate or temporarily create the needed compute environment to properly execute the target applications.

Service Workload

Moab can schedule and manage both individual applications and long-running or persistent services. Service workload processes externally-generated transaction requests while Moab provides the distributed service with needed resources to meet target backlog or response targets to the service. Examples of service workload include parallel databases, web farms, and visualization services. Moab can apply cluster, or dynamically-generated on-demand resources to the service.

When handling service workload, Moab observes the application in a highly abstract manner. Using the JOBCFG parameter, aspects of the service jobs can be discovered or configured with attributes describing them as resource consumers possessing response time, backlog, state metrics, and associated QoS targets. In addition, each application can specify the type of compute resource required (OS, arch, memory, disk, network adapter, data store, and so forth) as well as the support environment (network, storage, external services, and so forth).

If the QoS response time/backlog targets of the application are not being satisfied by the current resource allocation, Moab evaluates the needs of this application against all other site mission objectives and workload needs and determines what it must do to locate or create (that is, provision, customize, secure) the needed resources. With the application resource requirement specification, a site may also indicate proximity/locality constraints, partition policies, ramp-up/ramp-down rules, and so forth.

Once Moab identifies and creates appropriate resources, it hands these resources to the application via a site customized URL. This URL can be responsible for whatever application-specific hand-shaking must be done to launch and initialize the needed components of the distributed application upon the new resources. Moab engages in the hand-off by providing needed context and resource information and by launching the URL at the appropriate time.

Related topics

- Malleable Jobs
- QoS/SLA Enforcement

2.0 Installation and Initial Configuration

- **Hardware and Software Requirements** on page 7
- **Installing Moab** on page 8
- **Connecting Moab to MongoDB** on page 15
- **Upgrading Moab** on page 17
- **Initial Moab Configuration** on page 20
- **Initial Moab Testing** on page 22

2.1 Hardware and Software Requirements

- Hardware Requirements
- Supported Platforms

Hardware Requirements

Adaptive Computing recommends a quad-core system with 12 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.

Supported Platforms

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

Resource Managers that Integrate with Moab

The following resource managers integrate with Moab:

- SLURM
- TORQUE

Supported Operating Systems

Moab has been tested on the following variants of Linux:

- CentOS (5.7 and 6.3)
- RedHat (5.7 and 6.3)

- Scientific Linux (6.3)
- SuSE (11 SP2)

Moab has historically worked, but has not been tested, on the following operating systems:

- Debian
- AIX

Supported Architectures

Supported hardware architectures:

- Intel/AMD x86-64

2.2 Installing Moab

- Moab Server Installation
- 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").

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 and the `moab.lic` file. The default location for `moab.cfg` and `moab-private.cfg` is `/opt/moab/etc/` and is the recommended location for the license and configuration files.

`$MOABHOMEDIR` is required whenever the Moab binary is started or when client commands are used. It is recommended that you insert the `$MOABHOMEDIR` environment variable and its value into a global environment variable profile by editing the `/etc/profile`, `/etc/bashrc`, or `/etc/environment`

files (depending on your installation). Doing this will ensure that this environment variable is available to all users on the system without any action on their part.

All Moab executables are placed in `$MOABHOMEDIR/bin` or `$MOABHOMEDIR/sbin` (such as `/opt/moab/bin/`).

If you need to export your Moab home directory, run the following:

```
> export MOABHOMEDIR=/opt/moab
```

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`.

The following installation assumes that you have done a standard TORQUE installation according to the TORQUE 4.2 documentation, and that you have prepared TORQUE for a Moab installation, and that you use a RedHat or CentOS operating system.

To install Moab

The following instructions demonstrate installing Moab on a Red Hat 6 or CentOS 6 system. Run each step as the root user.

1. Install the required dependencies and packages.

   RHEL 5 and CentOS 5:

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

   RHEL 6, CentOS 6, and Scientific Linux 6:

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

   SLES:

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

2. Run each of the following commands in order.

   ```
   [root]# tar xzvf moab-7.2.9-xxxx.tar.gz (where xxxx can be one of: generic,
   generic-odbc, torque, torque-odbc)
   [root]# cd moab-7.2.9
   [root]# ./configure <option>
   ```

   In some cases, you might want to customize the location of the Moab home directory, the server daemon, and the client commands. You can make these configurations by using the `./configure` options (For a complete list of `./configure` options, use `./configure --help`.). We strongly recommend that you configure Moab with the `--with-init` and `--with-profile` options. If you

are using TORQUE as your resource manager, use the `--with-torque` option. If you are installing Moab Accounting Manager, configure Moab with the `--with-am` option.

Here are some examples of commonly used `./configure` options:

Option	Description	Example
--with-flexlm	Causes Moab to install the `license.mon.flexLM.pl` script in the `/opt/moab/tools` directory. For more information about this script, see **Interfacing to FLEXlm** on page 568.	`[root]# ./configure --with-flexlm`
--with-homedir	Specifies the location of the Moab configuration directory and the MOABHOMEDIR environment variable. The default location is `/opt/moab`. MOABHOMEDIR is automatically set on some distributions during installation, when the --with-profile option is enabled.	`[root]# ./configure --with-homedir=/var/moab` *The Moab home directory will be `/var/moab` instead of the default `/opt/moab`.*
--with-init	Enables the installation of a distribution-specific `/etc/init.d/moab` service startup script. This option is required if you want to install this script onto a new system. If you do not set this option, you must manually set up the Moab daemon service. The startup script is located at `OS/EL/etc/init.d/moab`. The TORQUE and Moab initialization scripts are provided in the `contrib/init.d` directory as a courtesy and may be modified at your discretion to work on your system.	`[root]# ./configure --with-init`

Option	Description	Example
--prefix	Specifies the location of the binaries and libraries of the Moab install. The default location is `/opt/moab`.	`[root]# ./configure --prefix=/usr/local`
--with-profile	Enables the installation of distribution-specific `/etc/profile.d/moab.[c]sh` setup script for bash and cshell. The MOABHOMEDIR, PERL5LIB, PATH and MANPATH environment variables are setup to specify where the new moab configuration, scripts, binaries and man pages reside. If you do not set this option, these scripts are not installed, and you must manually perform this set up. The environment setup scripts are located at `OS/EL/etc/profile.d/moab.[c]sh`.	`[root]# ./configure --with-profile`
--with-am	Specifies that you want to configure Moab with Moab Accounting Manager. *<type>* can be `mam` or `native`. There is a similar `--with-torque` option that configures Moab with TORQUE, but you do not need to specify this option if you install the "torque" tarball version.	`[root]# ./configure --with-am=<type>`

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

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

4. Install Moab.

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

5. *ONLY* if you are installing on non-RHEL distributions, copy the appropriate `init.d` file, set the permissions on it, and configure Moab to start automatically at system boot.

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

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

* If chkconfig doesn't work, try the following *
[root]# update-rc.d moab defaults
```

6. Modify the Moab configuration file.

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

Do the following:

a. Verify that **SUBMITCMD** is set up for your TORQUE resource manager (change `RMCFG<hostname>` to `RMCFG[torque]`), and that it points to a valid qsub executable. For example:

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

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

b. Make sure that you set **ENABLEPROXY** to *TRUE*:

```
ADMINCFG[1] ENABLEPROXY=TRUE
```

7. If you ran ./configure with the `--with-profile` option, source the following file to add the MWM home directory to your current shell *$PATH* environment.

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

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

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

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

Moab checks the status of the license every day just after midnight. At 60 and 45 days before, and daily from 30 days before license expiration to and including the license expiration date, Moab sends an e-mail to all level 1 administrators informing them of the pending Moab license expiration. A log record is also made of the upcoming expiration event. For the notifications to occur correctly, 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] EMAILADDRESS=u1@company.com
USERCFG[u2] EMAILADDRESS=u2@company.com
USERCFG[u3] EMAILADDRESS=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, 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.

9. Install and connect MongoDB. See **Connecting Moab to MongoDB** on page 15 for instructions.
10. Start Moab.

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

If Moab fails to start because `libodbc.so.1` cannot be found and `libodbc.so.2` is available on your system, you must create a symbolic link from `libodbc.so.1` to `libodbc.so.2`.

```
[root]# ln -s /usr/lib64/libodbc.so.2 /usr/lib64/libodbc.so.1
```

If Moab fails to start because `libodbc.so.2` cannot be found and `libodbc.so.1` is available on your system, you must create a symbolic link from `libodbc.so.2` to `libodbc.so.1`.

```
[root]# ln -s /usr/lib64/libodbc.so.1 /usr/lib64/libodbc.so.2
```

For more information, see the Unix ODBC documentation.

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

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

Moab Client Installation

After installing the Moab server on the head node, Moab can create a "client commands-only" tarball you can use to install just the Moab client commands on a login/client node. The tarball allows you to install the binary Moab client command files, with their man pages, using a single `tar` command. In addition, the tarball contains a `moab.cfg` file configured with the Moab host name and port number so you do not have to manually configure this information on the login/client node.

Command Installation when Server and Client Have Similar Architecture

After installing Moab on the head node, enter the following command:

```
> make client-pkg
```

A tarball is created with the name "client.tgz". Copy the tarball to the root directory of the client node, log in to the client node as root, and install the client commands using the following command:

```
> tar xvf client.tgz
```

The Moab client commands are now available on the login/client node.

Command Installation when Server and Client Have Diverse Architecture

By default, Moab client commands (from any build) are able to communicate and authenticate with any server. This can be a security risk depending on the type of environment in which Moab is running. If your site needs secure communication and authentication between Moab client commands and Moab server, it is recommended that you create a site-specific key and place it in the same directory as your `moab.cfg` file. By default, this is `$MOABHOMEDIR/etc/.moab.key`. When the Moab server and client commands detect the presence of those two files they will use the key in those files to authenticate and communicate, instead of the default key.

For more details, please see **Mauth Authentication** on page 975.

Preparing TORQUE for a Moab Installation

Several steps must be taken before installing Moab with TORQUE to ensure that they will communicate properly.

To prepare TORQUE for a Moab Installation

1. Edit the nodes file to list all of your nodes somewhere inside of it.

   ```
   > vim /var/spool/torque/server_priv/nodes
   ```

   ```
   ...
   node04
   node05
   node06
   ...
   ```

2. Copy `pbs_server` to the `etc/init.d/` directory.

   ```
   > cp contrib/init.d/pbs_server /etc/init.d
   ```

3. Configure TORQUE to start automatically when the system boots.

   ```
   > chkconfig --add pbs_server
   ```

   The chkconfig command is RedHat-specific. If you are using a different operating system, consult its documentation for a similar command.

4. Specify a TORQUE setup user.

   ```
   > ./torque.setup root
   ```

5. Stop TORQUE and restart it to verify that the startup script runs correctly.

```
> qterm
> /etc/init.d/pbs_server start
```

You can verify that the installation was successful by running the following command:

```
> pbsnodes
```

TORQUE returns information about each node. If TORQUE is properly configured, each node should report `state = free` to indicate that the server and moms are communicating.

Related topics

- End User Commands

2.3 Connecting Moab to MongoDB

Moab connects to a MongoDB database to store information for use by MWS. This feature allows MWS and Viewpoint to do very fast queries on various Moab objects without querying Moab directly.

The following instructions assume that the MongoDB server is on the same machine as Moab.

To connect Moab to MongoDB

1. Install MongoDB.

   RHEL, CentOS, and Scientific Linux:

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

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

   Install `mongo20-10gen` and `mongo20-10gen-server`.

   ```
   [root]# yum install mongo20-10gen mongo20-10gen-server
   ```

   SLES:

   ```
   [root]# zypper ar http://download.opensuse.org/repositories/server:/database/SLE_
   11_SP2 OpenSuseDatabase
   [root]# zypper install mongodb
   ```

2. Start MongoDB.

   RHEL, CentOS, and Scientific Linux:

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

   SLES:

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

3. Prepare the MongoDB database by doing the following:

   a. Add the required MongoDB users.

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

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

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

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

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

   b. Enable authentication in MongoDB.

   RHEL, CentOS, and Scientific Linux:

```
[root]# vi /etc/mongod.conf
...
auth = true
...
[root]# service mongod restart
```

   SLES:

   MongoDB authentication is enabled by default in SLES. To verify, check the value of `auth` as shown below.

```
[root]# nano /etc/mongodb.conf
...
auth = true
...
[root]# service mongodb restart
```

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

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

   You only need to specify a port if you have changed it from the default. When a port is not specified, Moab assumes the default Mongo port.

5. In `/opt/moab/etc/moab-private.cfg`, set the **MONGOUSER** and **MONGOPASSWORD** parameters to the MongoDB `moab_user` credentials you set in step 3.

```
MONGOUSER     moab_user
MONGOPASSWORD secret2
```

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

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

2.4 Upgrading Moab

The following instructions will guide you through a 6.1.x, 7.0.x, or 7.1.x to 7.2.0 upgrade. Depending on which version of Moab you are presently running, upgrade instructions may vary, so unless otherwise noted, all instructions assume use of a RHEL operating system; notes for SLES users are added in appropriate places.

Upgrading Moab may require changing the database. Please see the `README.database` file included in the Moab distribution for specific version information. Also, please see Migrating Your Database to Newer Versions of Moab for specific details on migrating your database.

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

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

To upgrade Moab

1. Untar the distribution file. For example:

```
> tar -xzvf moab-7.2.9.linux-x86_64-generic.tar.gz
```

2. Navigate to the unpacked directory.

```
> cd moab-7.2.9
```

3. Install the GNU C++ compiler.

```
> sudo yum install gcc-c++
```

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

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

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

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

5. Install the Boost C++ headers and shared development libraries.

```
> sudo yum install mongodb-devel boost-devel
```

6. 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`.

7. If you use the ODBC, you must upgrade to the 7.2 schema. See **Migrating Your Database to Newer Versions of Moab** on page 709 for more information.

8. 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, and new jobs cannot be submitted to Moab.

9. Run the make perldeps command to install the necessary perl modules using CPAN. When first running CPAN, you will be asked for configuration information. It is recommended that you choose an automatic configuration. You will be prompted to provide input during module installation; running the make perldeps command with a script is not recommended.

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

10. Install Moab.

```
> sudo make install
```

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

11. For SLES users only. If you are using a RHEL operating system, proceed to the next step.

    - If you are using SLES, convert `libmongoclient.a` to `libmongoclient.so`.

      ```
      cd /usr/lib64
      mkdir mongo
      cp libmongoclient.a mongo/
      cd mongo/
      ar -x libmongoclient.a
      gcc -shared *.o -o libmongoclient.so
      cp libmongoclient.so ../
      ```

    - If you are using SLES, create symbolic links that Moab will recognize to the boost libraries. The following script which is run in the `/usr/lib64` directory may be useful.

      ```
      ln -s libboost_date_time.so.1.36.0 libboost_date_time.so.5
      ln -s libboost_filesystem.so.1.36.0 libboost_filesystem.so.5
      ln -s libboost_graph.so.1.36.0 libboost_graph.so.5
      ln -s libboost_iostreams.so.1.36.0 libboost_iostreams.so.5
      ln -s libboost_math_c99.so.1.36.0 libboost_math_c99.so.5
      ln -s libboost_math_c99f.so.1.36.0 libboost_math_c99f.so.5
      ln -s libboost_math_c99l.so.1.36.0 libboost_math_c99l.so.5
      ln -s libboost_math_tr1.so.1.36.0 libboost_math_tr1.so.5
      ln -s libboost_math_tr1f.so.1.36.0 libboost_math_tr1f.so.5
      ln -s libboost_math_tr1l.so.1.36.0 libboost_math_tr1l.so.5
      ln -s libboost_mpi.so.1.36.0 libboost_mpi.so.5
      ln -s libboost_mpi_python.so.1.36.0 libboost_mpi_python.so.5
      ln -s libboost_prg_exec_monitor.so.1.36.0 libboost_prg_exec_monitor.so.5
      ln -s libboost_program_options.so.1.36.0 libboost_program_options.so.5
      ln -s libboost_python.so.1.36.0 libboost_python.so.5
      ln -s libboost_regex.so.1.36.0 libboost_regex.so.5
      ln -s libboost_serialization.so.1.36.0 libboost_serialization.so.5
      ln -s libboost_signals.so.1.36.0 libboost_signals.so.5
      ln -s libboost_system.so.1.36.0 libboost_system.so.5
      ln -s libboost_thread.so.1.36.0 libboost_thread.so.5
      ln -s libboost_unit_test_framework.so.1.36.0 libboost_unit_test_framework.so.5
      ln -s libboost_wave.so.1.36.0 libboost_wave.so.5
      ln -s libboost_wserialization.so.1.36.0 libboost_wserialization.so.5
      ln -s libboost_thread.so.1.36.0 libboost_thread-mt.so.5
      ```

12. 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: Tue May 17 16:38:27 MST 2011
    Build args: NA
    Compiled as little endian.
    Version: moab server 7.2.0 (revision 992)
    Copyright 2012 Adaptive Computing Enterprises, Inc., All Rights Reserved
    ```

13. If you are upgrading to Moab 7.2 and use Moab Accounting Manager:

    - If you use the native interface (**AMCFG** on page 802[...] **TYPE**=*native*), locate the following entries in the `moab.cfg` file:

      ```
      AMCFG[mam] QUOTEURL=exec:///$HOME/tools/mam/bank.quote.mam.pl
      AMCFG[mam] RESERVEURL=exec:///$HOME/tools/mam/bank.reserve.mam.pl
      ```

```
AMCFG[mam] CHARGEURL=exec://$HOME/tools/mam/bank.charge.mam.pl
AMCFG[mam] DELETEURL=exec:///$HOME/tools/mam/bank.delete.mam.pl
...
AMCFG[mam] RESERVEFAILUREACTION=hold,hold
AMCFG[mam] CREATEFAILUREACTION=ignore
```

Update the paths to the NAMI scripts to reflect the new `mam/usage.*.mam.pl` URL, add the **CREATEURL** on page 357, **UPDATEURL** on page 363, and **PAUSEURL** on page 361 attributes; replace the **RESERVEURL** attribute and script name with **STARTURL** on page 362 and its corresponding path; replace **CHARGEURL** with **ENDURL** on page 358 and its corresponding path; and replace the **RESERVEFAILUREACTION** attribute with **STARTFAILUREACTION** on page 362. You may also update the **CREATEFAILUREACTION** on page 357 attribute to specify how Moab handles different types of create job failures.

```
AMCFG[mam] QUOTEURL=exec://$TOOLSDIR/mam/usage.quote.mam.pl
AMCFG[mam] CREATEURL=exec://$TOOLSDIR/mam/usage.create.mam.pl
AMCFG[mam] STARTURL=exec://$TOOLSDIR/mam/usage.start.mam.pl
AMCFG[mam] UPDATEURL=exec://$TOOLSDIR/mam/usage.update.mam.pl
AMCFG[mam] PAUSEURL=exec://$TOOLSDIR/mam/usage.pause.mam.pl
AMCFG[mam] ENDURL=exec://$TOOLSDIR/mam/usage.end.mam.pl
AMCFG[mam] DELETEURL=exec://$TOOLSDIR/mam/usage.delete.mam.pl
...
AMCFG[mam] STARTFAILUREACTION=hold,hold
AMCFG[mam] CREATEFAILUREACTION=ignore,ignore
```

- If you use the gold interface (**AMCFG**[...] **TYPE**=*GOLD* or **AMCFG**[...] **SERVER**=*gold://...*), the interface name has changed to MAM. Modify the **AMCFG** **TYPE** or **SERVER** attribute to reference MAM (**AMCFG**[...] **TYPE**=*MAM* or **AMCFG** [...] **SERVER**=*mam://...*). You must also replace the **JOBFAILUREACTION** attribute with **STARTFAILUREACTION** on page 362.

```
AMCFG[mam] SERVER=mam://my_accounting_server
AMCFG[mam] STARTFAILUREACTION=hold,hold
```

14. Start Moab.

```
> moabd
```

2.5 Initial Moab Configuration

Configuring an RPM-based install of Moab

When Moab is installed via an RPM source, such as with the Moab HPC Suite or Moab Cloud Suite, the `moab.cfg` file contains only one directive - an `#IMPORT` line that imports all the configuration files in `/opt/moab/etc`. The usual configuration settings that are normally contained in `moab.cfg` have been moved to `moab-server.cfg`. Moab still reads the moab.cfg file and, due to the `#INCLUDE` directive, reads in all the other configuration files as well.

To configure Moab in the case of an RPM install, you can modify the `moab.cfg` file, the `moab-server.cfg` file, or any of the configuration files that are read in by `moab.cfg` such as the accounting manager configuration file (`am.cfg`) or the resource manager configuration file (`rm.cfg`).

The RPMs allow for a client install of Moab, instead of a server install. In this instance, the `moab-server.cfg` file is replaced with a `moab-client.cfg` file. The server and client RPMs cannot be installed on the same machine.

Basic configuration of Moab

After Moab is installed, there may be minor configuration remaining within the primary configuration file, `moab.cfg`. While the `configure` script automatically sets these parameters, sites may choose to specify additional parameters. If the values selected in `configure` are satisfactory, then this section may be safely ignored.

The parameters needed for proper initial startup include the following:

Parameter	Instructions

SCHEDCFG	The SCHEDCFG parameter specifies how the Moab server will execute and communicate with client requests. The **SERVER** attribute allows Moab client commands to locate the Moab server and is specified as a URL or in *<HOST>[:<PORT>]* format. For example:
	`SCHEDCFG[orion] SERVER=cw.psu.edu`
	The **SERVER** attribute can also be set using the environment variable **$MOABSERVER**. Using this variable allows you to quickly change to Moab server that client commands will connect to.
	`> export MOABSERVER=cluster2:12221`

ADMINCFG	Moab provides role-based security enabled via multiple levels of admin access. Users who are to be granted full control of all Moab functions should be indicated by setting the ADMINCFG 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.

Parameter	Instructions
RMCFG	For Moab to properly interact with a resource manager, the interface to this resource manager must be defined as described in the Resource Manager Configuration Overview. Further, it is important that the primary Moab administrator also be a resource manager administrator within each of those systems. For example, to interface to a TORQUE resource manager, the following may be used: `RMCFG[torque1] TYPE=pbs`

Related topics

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

2.6 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:

- Minimal Configuration Required To Start
  - Normal Mode
  - Monitor Mode
  - Interactive Mode
  - Simulation Mode

Scheduler Modes

Central to Moab testing is the **MODE** attribute of the SCHEDCFG parameter. This parameter attribute allows administrators to determine how Moab will run. The possible values for **MODE** are *NORMAL*, *MONITOR*, *INTERACTIVE*, and *SIMULATION*. For example, to request monitor mode operation, include the following in the `moab.cfg` file:

```
SCHEDCFG MODE=MONITOR
```

Normal Mode

If initial evaluation is complete or not required, you can place the scheduler directly into *production* by setting the **MODE** attribute of the **SCHEDCFG** parameter to *NORMAL* and (re)starting the scheduler.

Monitor Mode (or Test Mode)

Monitor mode allows evaluation of new Moab releases, configurations, and policies in a risk-free manner. In monitor mode, the scheduler connects to the resource manager(s) and obtains live resource and workload information. Using the policies specified in the `moab.cfg` file, the monitor-mode Moab behaves identical to a live or normal-mode Moab except the ability to start, cancel, or modify jobs is disabled. In addition, allocation management does not occur in monitor mode. This allows safe diagnosis of the scheduling state and behavior using the various diagnostic client commands. Further, the log output can also be evaluated to see if any unexpected situations have arisen. At any point, the scheduler can be dynamically changed from monitor to normal mode to begin *live* scheduling.

To set up Moab in monitor mode, do the following:

```
> vi moab.cfg
  (change the MODE attribute of the SCHEDCFG parameter from NORMAL to MONITOR)
> moab
```

Remember that Moab running in monitor mode will not interfere with your production scheduler.

Running Multiple Moab Instances Simultaneously

If running multiple instances of Moab, whether in simulation, normal, or monitor mode, make certain that each instance resides in a different home directory to prevent conflicts with configuration, log, and statistics files. Before starting each additional Moab, set the **MOABHOMEDIR** environment variable in the execution environment to point to the local home directory. Also, each instance of Moab should run using a different port to avoid conflicts.

If running multiple versions of Moab, not just different Moab modes or configurations, set the *$PATH* variable to point to the appropriate Moab binaries.

To *point* Moab client commands (such as showq) to the proper Moab server, use the appropriate command line arguments or set the environment variable **MOABHOMEDIR** in the client execution environment as in the following example:

```
# point moab clients/server to new configuration
> export MOABHOMEDIR=/opt/moab-monitor
# set path to new binaries (optional)
> export PATH=/opt/moab-monitor/bin:/opt/moab-monitor/sbin:$PATH
# start Moab server
> moab
# query Moab server
> showq
```

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

Interactive Mode

Interactive mode allows for evaluation of new versions and configurations in a manner different from monitor mode. Instead of disabling all resource and job control functions, Moab sends the desired change request to the screen and asks for permission to complete it. For example, before starting a job, Moab may post something like the following to the screen:

```
Command:  start job 1139.ncsa.edu on node list test013,test017,test018,test021
Accept:  (y/n) [default: n]?
```

The administrator must specifically accept each command request after verifying that it correctly meets desired site policies. Moab will then execute the specified command. This mode is useful in validating scheduler behavior and can be used until configuration is appropriately tuned and all parties are comfortable with the scheduler's performance. In most cases, sites will want to set the scheduling mode to normal after verifying correct behavior.

Simulation Mode

Simulation mode is of value in performing a *test drive* of the scheduler or when a stable production system exists and an evaluation is desired of how various policies can improve the current performance. See the Simulations documentation for more information.

3.0 Scheduler Basics

- **Layout of Scheduler Components** on page 25
- **Scheduling Environment** on page 27
  - **Scheduling Dictionary** on page 33
- **Scheduling Iterations and Job Flow** on page 40
- **Configuring the Scheduler** on page 42
- **Credential Overview** on page 45
  - **Job Attributes/Flags Overview** on page 72

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.

- `* $(MOABHOMEDIR)` (default is `/opt/moab` and can be modified via the `--with-homedir` parameter during `./configure`) contains the following files:

Filename	Description
.moab.ck	Checkpoint file
.moab.pid	Lock file
moab.lic	License file
contrib/	Directory containing contributed code and plug-ins
docs/	Directory for documentation
etc/	Directory for configuration files
moab.cfg	General configuration file

Filename	Description
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
stats/	Directory for statistics files: ◦ `events.<date>` – event files ◦ `{DAY|WEEK|MONTH|YEAR}.<date>` – usage profiling data ◦ `FS.<PARTITION>.<epochtime>` – fairshare usage data
samples/	Directory for sample configuration files, simulation trace files, etc.

- `$(MOABINSTDIR)` (default is `/opt/moab` and can be modified via the `--prefix` parameter during `./configure`) contains the following files:

Filename	Description
bin/	Directory for client commands (for example, showq, setres, etc.)
sbin/	Directory for server daemons
moab	Moab binary
tools/	Directory for resource manager interfaces and local scripts

- `/etc/moab.cfg` – If the Moab home directory cannot be found at startup, this file is checked to see if it declares the Moab home directory. If a declaration exists, the system checks the declared directory to find Moab. The syntax is: `MOABHOMEDIR=<DIRECTORY>`.

If you want to run Moab from a different directory other than `/opt/moab` but did not use `the --with-homedir` parameter during `./configure`, you can set the *$MOABHOMEDIR* environment variable,

declare the home directory in the `/etc/moab.cfg` file, or use the `-C` command line option when using the Moab server or client commands to specify the configuration file location.

When Moab runs, it creates a log file, `moab.log`, in the `log/` directory and creates a statistics file in the `stats/` directory with the naming convention `events.WWW_MMM_DD_YYYY` (for example, `events.Sat_Oct_10_2009`). Additionally, a checkpoint file, `.moab.ck`, and lock file, `.moab.pid`, are maintained in the Moab home directory.

Layout of Scheduler Components with Integrated Database Enabled

If USEDATABASE INTERNAL is configured, the layout of scheduler components varies slightly. The `.moab.ck` file and usage profiling data (`stat/{DAY|WEEK|MONTH|YEAR}.<date>`) are stored in the `moab.db` database. In addition, the event information is stored in both event files: (`stat/events.<date>`) and `moab.db`.

Related topics

- Commands Overview
- Installation

3.2 Scheduling Environment

Moab functions by manipulating a number of elementary objects, including jobs, nodes, reservations, QoS structures, resource managers, and policies. Multiple minor elementary objects and composite objects are also used; these objects are defined in the scheduling dictionary.

- Jobs
    - Job States
    - Requirement (or Req)
- Nodes
- Advance Reservations
- Policies
- Resources
- Task
- PE
- Class (or Queue)
- Resource Manager (RM)

Moab functions by manipulating a number of elementary objects, including jobs, nodes, reservations, QoS structures, resource managers, and policies. Multiple minor elementary objects and composite objects are also used; these objects are defined in the scheduling dictionary.

Jobs

Job information is provided to the Moab scheduler from a resource manager such as Loadleveler, PBS, Wiki, or LSF. Job attributes include ownership of the job, job state, amount and type of resources required by the job, and a wallclock limit indicating how long the resources are required. A job consists of one or more task groups, each of which requests a number of resources of a given type; for example, a job may consist of two task groups, the first asking for a single master task consisting of *1 IBM SP node with at least 512 MB of RAM* and the second asking for a set of slave tasks such as *24 IBM SP nodes with at least 128 MB of RAM*. Each task group consists of one or more tasks where a task is defined as the minimal independent unit of resources. By default, each task is equivalent to one processor. In SMP environments, however, users may wish to tie one or more processors together with a certain amount of memory and other resources.

Job States

The job's *state* indicates its current status and eligibility for execution and can be any of the values listed in the following tables:

Table 3-1: Pre-execution states

State	Definition
Deferred	Job that has been held by Moab due to an inability to schedule the job under current conditions. Deferred jobs are held for DEFERTIME before being placed in the idle queue. This process is repeated DEFERCOUNT times before the job is placed in batch hold.
Hold	Job is idle and is not eligible to run due to a user, (system) administrator, or batch system *hold* (also, batchhold, systemhold, userhold).
Idle	Job is currently queued and eligible to run but is not executing (also, **notqueued**).
NotQueued	The job has not been queued.
Unknown	Moab cannot determine the state of the job.

Table 3-2: Execution states

State	Definition
Starting	Batch system has attempted to start the job and the job is currently performing *pre-start* tasks that may include provisioning resources, staging data, or executing system pre-launch scripts.
Running	Job is currently executing the user application.
Suspended	Job was running but has been suspended by the scheduler or an administrator; user application is still in place on the allocated compute resources, but it is not executing.

Table 3-3: Post-execution states

State	Definition
Completed	Job has completed running without failure.
Removed	Job has run to its requested walltime successfully but has been canceled by the scheduler or resource manager due to exceeding its walltime or violating another policy; includes jobs canceled by users or administrators either before or after a job has started.
Vacated	Job canceled after partial execution due to a system failure.

Task Group (or Req)

A job *task group* (or req) consists of a request for a single type of resources. Each task group consists of the following components:

Component	Description
Task Definition	A specification of the elementary resources that compose an individual task.
Resource Constraints	A specification of conditions that must be met for resource matching to occur. Only resources from nodes that meet *all* resource constraints may be allocated to the job task group.
Task Count	The number of task instances required by the task group.
Task List	The list of nodes on which the task instances are located.
Task Group Statistics	Statistics tracking resource utilization.

Nodes

Moab recognizes a node as a collection of resources with a particular set of associated attributes. This definition is similar to the traditional notion of a node found in a Linux cluster or supercomputer wherein a node is defined as one or more CPUs, associated memory, and possibly other compute resources such as local disk, swap, network adapters, and software licenses. Additionally, this node is described by various attributes such as an architecture type or operating system. Nodes range in size from small uniprocessor PCs to large symmetric multiprocessing (SMP) systems where a single node may consist of hundreds of CPUs and massive amounts of memory.

In many cluster environments, the primary source of information about the configuration and status of a compute node is the resource manager. This information can be augmented by additional information

sources including node monitors and information services. Further, extensive node policy and node configuration information can be specified within Moab via the graphical tools or the configuration file. Moab aggregates this information and presents a comprehensive view of the node configuration, usages, and state.

While a node in Moab in most cases represents a standard compute host, nodes may also be used to represent more generalized resources. The GLOBAL node possesses floating resources that are available cluster wide, and created virtual nodes (such as network, software, and data nodes) track and allocate resource usage for other resource types.

For additional node information, see General Node Administration.

Advance Reservations

An advance reservation dedicates a block of specific resources for a particular use. Each reservation consists of a list of resources, an access control list, and a time range for enforcing the access control list. The reservation ensures the matching nodes are used according to the access controls and policy constraints within the time frame specified. For example, a reservation could reserve 20 processors and 10 GB of memory for users Bob and John from Friday 6:00 a.m. to Saturday 10:00 p.m. Moab uses advance reservations extensively to manage backfill, guarantee resource availability for active jobs, allow service guarantees, support deadlines, and enable metascheduling. Moab also supports both regularly recurring reservations and the creation of dynamic one-time reservations for special needs. Advance reservations are described in detail in the Advance Reservations overview.

Policies

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

Resources

Jobs, nodes, and reservations all deal with the abstract concept of a resource. A resource in the Moab world is one of the following:

Resource	Description
processors	Specify with a simple count value
memory	Specify real memory or RAM in megabytes (MB)
swap	Specify virtual memory or *swap* in megabytes (MB)
disk	Specify local disk in megabytes (MB)

In addition to these elementary resource types, there are two higher level resource concepts used within Moab: **Task** and the processor equivalent, or (**PE**).

Task

A task is a collection of elementary resources that must be allocated together within a single node. For example, a task may consist of one processor, 512 MB of RAM, and 2 GB of local disk. A key aspect of a task is that the resources associated with the task must be allocated as an atomic unit, without spanning node boundaries. A task requesting 2 processors cannot be satisfied by allocating 2 uniprocessor nodes, nor can a task requesting 1 processor and 1 GB of memory be satisfied by allocating 1 processor on 1 node and memory on another.

In Moab, when jobs or reservations request resources, they do so in terms of tasks typically using a task count and a task definition. By default, a task maps directly to a single processor within a job and maps to a full node within reservations. In all cases, this default definition can be overridden by specifying a new task definition.

Within both jobs and reservations, depending on task definition, it is possible to have multiple tasks from the same job mapped to the same node. For example, a job requesting 4 tasks using the default task definition of 1 processor per task can be satisfied by 2 dual processor nodes.

PE

The concept of the processor equivalent, or PE, arose out of the need to translate multi-resource consumption requests into a scalar value. It is not an elementary resource but rather a derived resource metric. It is a measure of the actual impact of a set of requested resources by a job on the total resources available system wide. It is calculated as follows:

```
 PE = MAX(ProcsRequestedByJob / TotalConfiguredProcs,
MemoryRequestedByJob / TotalConfiguredMemory,
DiskRequestedByJob / TotalConfiguredDisk,
SwapRequestedByJob / TotalConfiguredSwap) * TotalConfiguredProcs
```

For example, if a job requested 20% of the total processors and 50% of the total memory of a 128-processor MPP system, only two such jobs could be supported by this system. The job is essentially using 50% of all available resources since the system can only be scheduled to its most constrained resource - memory in this case. The processor equivalents for this job should be 50% of the processors, or PE = 64.

Another example: Assume a homogeneous 100-node system with 4 processors and 1 GB of memory per node. A job is submitted requesting 2 processors and 768 MB of memory. The PE for this job would be calculated as follows:

```
 PE = MAX(2/(100*4), 768/(100*1024)) * (100*4) = 3.
```

This result makes sense since the job would be consuming 3/4 of the memory on a 4-processor node.

The calculation works equally well on homogeneous or heterogeneous systems, uniprocessor or large SMP systems.

Class (or Queue)

A class (or queue) is a logical container object that implicitly or explicitly applies policies to jobs. In most cases, a class is defined and configured within the resource manager and associated with one or more of the following attributes or constraints:

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

As stated previously, most resource managers allow full class configuration within the resource manager. Where additional class configuration is required, the CLASSCFG parameter may be used.

Moab tracks class usage as a consumable resource allowing sites to limit the number of jobs using a particular class. This is done by monitoring class initiators that may be considered to be a ticket to run in a particular class. Any compute node may simultaneously support several types of classes and any number of initiators of each type. By default, nodes will have a one-to-one mapping between class initiators and configured processors. For every job task run on the node, one class initiator of the appropriate type is consumed. For example, a 3-processor job submitted to the class "batch" consumes three batch class initiators on the nodes where it runs.

Using queues as consumable resources allows sites to specify various policies by adjusting the class initiator to node mapping. For example, a site running serial jobs may want to allow a particular 8-processor node to run any combination of batch and special jobs subject to the following constraints:

- Only 8 jobs of any type allowed simultaneously.
- No more than 4 special jobs allowed simultaneously.

To enable this policy, the site may set the node's MAXJOB policy to 8 and configure the node with 4 special class initiators and 8 batch class initiators.

In virtually all cases, jobs have a one-to-one correspondence between processors requested and class initiators required. However, this is not a requirement, and with special configuration, sites may choose to associate job tasks with arbitrary combinations of class initiator requirements.

In displaying class initiator status, Moab signifies the type and number of class initiators available using the format [<CLASSNAME>:<CLASSCOUNT>]. This is most commonly seen in the output of node status commands indicating the number of configured and available class initiators, or in job status commands when displaying class initiator requirements.

Resource Manager (RM)

While other systems may have more strict interpretations of a resource manager and its responsibilities, Moab's multi-resource manager support allows a much more liberal interpretation. In essence, any object that can provide environmental information and environmental control can be used as a resource manager, including sources of resource, workload, credential, or policy information such as scripts, peer services, databases, web services, hardware monitors, or even flats files. Likewise, Moab considers 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, to be a resource 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*.

3.2.1 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 QoSes. ACLs may contain both allow and deny statements, include wildcards, and contain rules based on multiple object types.
Example	Reservation META1 contains 4 access statements. • Allow jobs owned by user "john" or "bob " • Allow jobs with QoS "premium" • Deny jobs in class "debug" • Allow jobs with a duration of less than 1 hour

Allocation

Definition	A logical, scalar unit assigned to users on a credential basis, providing access to a particular quantity of compute resources. Allocations are consumed by jobs associated with those credentials.
Example	`ALLOCATION=30000`

Class

Definition	(See Queue) A class is a logical container object that holds jobs allowing a site to associate various constraints and defaults to these jobs. Class access can also be tied to individual nodes defining whether a particular node will accept a job associated with a given class. Class based access to a node is denied unless explicitly allowed via resource manager configuration. Within Moab, classes are tied to jobs as a credential.
Example	job "cw.073" is submitted to class batch node "cl02" accepts jobs in class batch reservation weekend allows access to jobs in class batch

CPU

Definition	A single processing unit. A CPU is a consumable resource. Nodes typically consist of one or more CPUs. (same as processor)

Credential

Definition	An attribute associated with jobs and other objects that determines object identity. In the case of schedulers and resource managers, credential based policies and limits are often established. At submit time, jobs are associated with a number of credentials such as user, group , account , QoS, and class. These job credentials subject the job to various polices and grant it various types of access. In most cases, credentials set both the privileges of the job and the ID of the actual job executable.

Example	Job "cw.24001" possesses the following credentials:
	`USER=john;GROUP=staff;ACCOUNT=[NONE];`
	`QOS=[DEFAULT];CLASS=batch`

Disk

Definition	A quantity of local disk available for use by batch jobs. Disk is a consumable resource.

Execution Environment	
Definition	A description of the environment in which the executable is launched. This environment may include attributes such as the following: • an executable • command line arguments • input file • output file • local user ID • local group ID • process resource limits

Example	Job "cw.24001" possesses the following execution environment:
	`EXEC=/bin/sleep;ARGS="60";`
	`INPUT=[NONE];OUTPUT=[NONE];`
	`USER=loadl;GROUP=staff;`

Fairshare	
Definition	A mechanism that allows historical resource utilization information to be incorporated into job priority decisions.

Fairness	
Definition	The access to shared compute resources that each user is granted. Access can be equal or based on factors such as historical resource usage, political issues, and job value.

Group	
Definition	A credential typically directly mapping to a user's UNIX group ID.

Job	
Definition	The fundamental object of resource consumption. A job contains the following components: • A list of required consumable resources • A list of resource constraints controlling which resources may be allocated to the job • A list of job constraints controlling where, when, and how the job should run • A list of credentials • An execution environment

Job Constraints

Definition

A set of conditions that must be fulfilled for the job to start. These conditions are far reaching and may include one or more of the following:

- When the job may run. (After time X, within Y minutes.)
- Which resources may be allocated. (For example, node must possess at least 512 MB of RAM, run only in partition or Partition C, or run on HostA and HostB.)
- Starting job relative to a particular event. (Start after job X successfully completes.)

Example

```
RELEASETIME>='Tue Feb 12, 11:00AM'
DEPEND=AFTERANY:cw.2004
NODEMEMORY==256MB
```

Memory

Definition

A quantity of physical memory (RAM). Memory is provided by compute nodes. It is required as a constraint or consumed as a consumable resource by jobs. Within Moab, memory is tracked and reported in megabytes (MB).

Example

Node "node001" provides the following resources:

`PROCS=1,MEMORY=512,SWAP=1024`

"Job cw.24004" consumes the following resources per task:

`PROCS=1,MEMORY=256`

Node

Definition

A node is the fundamental object associated with compute resources. Each node contains the following components:

- A list of consumable resources
- A list of node attributes

Node Attribute

Definition

A node attribute is a non-quantitative aspect of a node. Attributes typically describe the node itself or possibly aspects of various node resources such as processors or memory. While it is probably not optimal to aggregate node and resource attributes together in this manner, it is common practice. Common node attributes include processor architecture, operating system, and processor speed. Jobs often specify that resources be allocated from nodes possessing certain node attributes.

Example

```
ARCH=AMD,OS=LINUX24,PROCSPEED=950
```

Node Feature	
Definition	A node feature is a node attribute that is typically specified locally via a configuration file. Node features are opaque strings associated with the node by the resource manager that generally only have meaning to the end-user, or possibly to the scheduler. A node feature is commonly associated with a subset of nodes allowing end-users to request use of this subset by requiring that resources be allocated from nodes with this feature present. In many cases, node features are used to extend the information provided by the resource manager.
Example	`FEATURE=s950,pIII,geology` *This may be used to indicate that the node possesses a 950 MHz Pentium III processor and that the node is owned by the Geology department.*

Processor	
Definition	A processing unit. A processor is a consumable resource. Nodes typically consist of one or more processors. (same as CPU)

Quality of Service (QoS)	
Definition	An object that provides special services, resources, and so forth.

Queue	
Definition	(see Class)

Reservation	
Definition	An object that reserves a specific collection or resources for a specific timeframe for use by jobs that meet specific conditions.
Example	Reserve 24 processors and 8 GB of memory from time T1 to time T2 for use by user X or jobs in the class batch.

Resource	
Definition	Hardware, generic resources such as software, and features available on a node, including memory, disk, swap, and processors.

Resource, Available	
Definition	A compute node's configured resources minus the *maximum* of the sum of the resources utilized by all job tasks running on the node and the resources dedicated; that is, R.Available = R.Configure - MAX(R.Dedicated,R.Utilized). In most cases, resources utilized will be associated with compute jobs that the batch system has started on the compute nodes, although resource consumption may also come from the operating system or *rogue* processes outside of the batch system's knowledge or control. Further, in a well-managed system, utilized resources are less than or equal to dedicated resources and when exceptions are detected, one or more usage-based limits are activated to preempt the jobs violating their requested resource usage.
Example	Node "cl003" has 4 processors and 512 MB of memory. It is executing 2 tasks of job "clserver.0041" that are using 1 processor and 60 MB of memory each. One processor and 250 MB of memory are reserved for user "jsmith" but are not currently in use. Resources available to user jsmith on node "cl003": • 2 processors • 392 MB memory Resources available to a user other than jsmith on node "cl003": • 1 processor • 142 MB memory

Resource, Configured	
Definition	The total amount of consumable resources available on a compute node for use by job tasks.
Example	Node "cl003" has 4 processors and 512 MB of memory. It is executing 2 tasks of job "clserver.0041" that are using 1 processor and 60 MB of memory each. One processor and 250 MB of memory are reserved for user "jsmith" but are not currently in use. Configured resources for node "cl003": • 4 processors • 512 MB memory

Resource, Consumable	
Definition	Any object that can be used (that is, consumed and thus made unavailable to another job) by, or dedicated to a job is considered to be a resource. Common examples of resources are a node's physical memory or local disk. As these resources may be given to one job and thus become unavailable to another, they are considered to be consumable. Other aspects of a node, such as its operating system, are not considered to be consumable since its use by one job does not preclude its use by another. Note that some node objects, such as a network adapter, may be dedicated under some operating systems and resource managers and not under others. On systems where the network adapter cannot be dedicated and the network usage per job cannot be specified or tracked, network adapters are not considered to be resources, but rather attributes. Nodes possess a specific quantity of consumable resources such as real memory, local disk, or processors. In a resource management system, the node manager may choose to report only those configured resources available to batch jobs. For example, a node may possess an 80-GB hard drive but may have only 20 GB dedicated to batch jobs. Consequently, the resource manager may report that the node has 20 GB of local disk available when idle. Jobs may explicitly request a certain quantity of consumable resources.

Resource, Constraint	
Definition	A resource constraint imposes a rule on which resources can be used to match a resource request. Resource constraints either specify a required quantity and type of resource or a required node attribute. All resource constraints must be met by any given node to establish a match.

Resource, Dedicated	
Definition	A job may request that a block of resources be dedicated while the job is executing. At other times, a certain number of resources may be reserved for use by a particular user or group. In these cases, the scheduler is responsible for guaranteeing that these resources, utilized or not, are set aside and made unavailable to other jobs.
Example	Node " cl003" has 4 processors and 512 MB of memory. It is executing 2 tasks of job "clserver.0041" that are using 1 processor and 60 MB of memory each. One processor and 250 MB of memory are reserved for user "jsmith" but are not currently in use. Dedicated resources for node "cl003": • 1 processor • 250 MB memory

Resource, Utilized	
Definition	All consumable resources actually used by all job tasks running on the compute node.

Resource, Utilized	
Example	Node "cl003" has 4 processors and 512 MB of memory. It is executing 2 tasks of job "clserver.0041" that are using 1 processor and 60 MB of memory each. One processor and 250 MB of memory are reserved for user "jsmith" but are not currently in use. Utilized resources for node "cl003": • 2 processors • 120 MB memory

Swap	
Definition	A quantity of virtual memory available for use by batch jobs. Swap is a consumable resource provided by nodes and consumed by jobs.

Task	
Definition	An atomic collection of consumable resources.

User, Global	
Definition	The user credential used to provide access to functions and resources. In local scheduling, global user IDs map directly to local user IDs.

User, Local	
Definition	The user credential under which the job executable will be launched.

Workload	
Definition	Generalized term.

3.3 Scheduling Iterations and Job Flow

- Scheduling Iterations
  - Update State Information
  - Handle User Requests

    - Perform Next Scheduling Cycle
- Detailed Job Flow
    - Determine Basic Job Feasibility
    - Prioritize Jobs
    - Enforce Configured Throttling Policies
    - Determine Resource Availability
    - Allocate Resources to Job
    - Launch Job

Scheduling Iterations

In any given scheduling iteration, many activities take place, examples of which are listed below:

- Refresh reservations
- Schedule reserved jobs
- Schedule priority jobs
- Backfill jobs
- Update statistics
- Update State Information
- Handle User Requests
- Perform Next Scheduling Cycle

Update State Information

Each iteration, the scheduler contacts the resource manager(s) and requests up-to-date information on compute resources, workload, and policy configuration. On most systems, these calls are to a centralized resource manager daemon that possesses all information. Jobs may be reported as being in any of the following states listed in the job state table.

Handle User Requests

User requests include any call requesting state information, configuration changes, or job or resource manipulation commands. These requests may come in the form of user client calls, peer daemon calls, or process signals.

Perform Next Scheduling Cycle

Moab operates on a polling/event driven basis. When all scheduling activities complete, Moab processes user requests until a new resource manager event is received or an internal event is generated. Resource manager events include activities such as a new job submission or completion of an active job, addition of new node resources, or changes in resource manager policies. Internal events include administrator schedule requests, reservation activation/deactivation, or the expiration of the RMPOLLINTERVAL timer.

Detailed Job Flow

Determine Basic Job Feasibility

The first step in scheduling is determining which jobs are feasible. This step eliminates jobs that have job holds in place, invalid job states (such as Completed, Not Queued, Deferred), or unsatisfied preconditions. Preconditions may include stage-in files or completion of preliminary job steps.

Prioritize Jobs

With a list of feasible jobs created, the next step involves determining the relative priority of all jobs within that list. A priority for each job is calculated based on job attributes such as job owner, job size, and length of time the job has been queued.

Enforce Configured Throttling Policies

Any configured throttling policies are then applied constraining how many jobs, nodes, processors, and so forth are allowed on a per credential basis. Jobs that violate these policies are not considered for scheduling.

Determine Resource Availability

For each job, Moab attempts to locate the required compute resources needed by the job. For a match to be made, the node must possess all node attributes specified by the job and possess adequate available resources to meet the "TasksPerNode" job constraint. (Default "TasksPerNode" is 1.) Normally, Moab determines that a node has adequate resources if the resources are *neither utilized by nor dedicated to* another job using the calculation.

R.Available = R.Configured - MAX(R.Dedicated,R.Utilized).

The **NODEAVAILABILITYPOLICY** on page 873 parameter can be modified to adjust this behavior.

Allocate Resources to Job

If adequate resources can be found for a job, the node allocation policy is then applied to select the best set of resources. These allocation policies allow selection criteria such as speed of node, type of reservations, or excess node resources to be figured into the allocation decision to improve the performance of the job and maximize the freedom of the scheduler in making future scheduling decisions.

Launch Job

With the resources selected and task distribution mapped, the scheduler then contacts the resource manager and informs it where and how to launch the job. The resource manager then initiates the actual job executable.

3.4 Configuring the Scheduler

- Adjusting Server Behavior
  - Logging
  - Checkpointing

- Client Interface
- 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 3will temporarily adjust the scheduler log level. When the scheduler restarts, the log level restores to the value stored in the Moab configuration files. To adjust a parameter permanently, the option `--flags=persistent` should be set.

At any time, the current server parameter settings may be viewed using the mschedctl -l command.

Adjusting Server Behavior

Most aspects of Moab behavior are configurable. This includes both scheduling policy behavior and daemon behavior. In terms of configuring server behavior, the following realms are most commonly modified.

Logging

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

Parameter	Description
LOGDIR	Indicates directory for log files.
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.

Parameter	Description
LOGLEVEL	Indicates verbosity of logging.

Checkpointing

Moab checkpoints its internal state. The checkpoint file records statistics and attributes for jobs, nodes, reservations, users, groups, classes, and almost every other scheduling object.

Parameter	Description
CHECKPOINTEXPIRATIONTIME	Indicates how long unmodified data should be kept after the associated object has disappeared; that is, job priority for a job no longer detected.
CHECKPOINTFILE	Indicates path name of checkpoint file.
CHECKPOINTINTERVAL	Indicates interval between subsequent checkpoints.

Client Interface

The Client interface is configured using the SCHEDCFG parameter. Most commonly, the attributes **SERVER** and **PORT** must be set to point client commands to the appropriate Moab server. Other parameters such as CLIENTTIMEOUT may also be set.

Scheduler Mode

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

Mode	Description
INTERACTIVE	Moab interactively confirms each scheduling action before taking any steps. (See interactive mode overview for more information.)
MONITOR	Moab observes cluster and workload performance, collects statistics, interacts with allocation management services, and evaluates failures, but it does not actively alter the cluster, including job migration, workload scheduling, and resource provisioning. (See monitor mode overview for more information.)
NORMAL	Moab actively schedules workload according to mission objectives and policies; it creates reservations; starts, cancels, preempts, and modifies jobs; and takes other scheduling actions.
SIMULATION	Moab obtains workload and resource information from specified simulation trace files and schedules the defined virtual environment.

Mode	Description
SINGLESTEP	Moab behaves as in **NORMAL** mode but will only schedule a single iteration and then exit.
SLAVE	Moab behaves as in **NORMAL** mode but will only start a job when explicitly requested by a trusted administrator.
TEST	Moab behaves as in **NORMAL** mode, will make reservations, and scheduling decisions, but will then only log scheduling actions it would have taken if running in **NORMAL** mode. In most cases, "TEST" mode is identical to **MONITOR** mode. (See test mode overview for more information.)

Related topics

- 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.

- General Credential Attributes
- User Credential
- Group Credential
- Account (or Project) Credential
- Class (or Queue) Credential
- QoS Credential

General Credential Attributes

Internally, credentials are maintained as objects. Credentials can be created, destroyed, queried, and modified. They are associated with jobs and requests providing access and privileges. Each credential type has the following attributes:

- Priority Settings
- Usage Limits
- Service Targets
- Credential and Partition Access

- Statistics
- Credential Defaults, State and Configuration Information

All credentials represent a form of identity, and when applied to a job, express ownership. Consequently, jobs are subject to policies and limits associated with their owners.

Credential Priority Settings

Each credential may be assigned a priority using the **PRIORITY** attribute. This priority affects a job's total credential priority factor as described in the Priority Factors section. In addition, each credential may also specify priority weight offsets, which adjust priority weights that apply to associated jobs. These priority weight offsets include FSWEIGHT (See Priority-Based Fairshare for more information.), QTWEIGHT, and XFWEIGHT.

For example:

```
# set priority weights
CREDWEIGHT      1
USERWEIGHT      1
CLASSWEIGHT     1
SERVICEWEIGHT   1
XFACTORWEIGHT   10
QUEUETIMEWEIGHT 1000
# set credential priorities
USERCFG[john] PRIORITY=200
CLASSCFG[batch] PRIORITY=15
CLASSCFG[debug] PRIORITY=100
QOSCFG[bottomfeeder] QTWEIGHT=-50 XFWEIGHT=100
ACCOUNTCFG[topfeeder] PRIORITY=100
```

Credential Usage Limits

Usage limits constrain which jobs may run, which jobs may be considered for scheduling, and what quantity of resources each individual job may consume. With usage limits, policies such as MAXJOB, MAXNODE, and MAXMEM may be enforced against both idle and active jobs. Limits may be applied in any combination as shown in the example below where usage limits include 32 active processors per group and 12 active jobs for user *john*. For a job to run, it must satisfy the most limiting policies of all associated credentials. The Throttling Policy section documents credential usage limits in detail.

```
GROUPCFG[DEFAULT] MAXPROC=32 MAXNODE=100
GROUPCFG[staff]   MAXNODE=200
USERCFG[john]     MAXJOB=12
```

Service Targets

Credential service targets allow jobs to obtain special treatment to meet usage or response time based metrics. Additional information about service targets can be found in the Fairshare section.

Credential and Partition Access

Access to partitions and to other credentials may be specified on a per credential basis with credential access lists, default credentials, and credential membership lists.

Credential Access Lists

You can use the **ALIST**, **PLIST**, and **QLIST** attributes (shown in the following table) to specify the list of credentials or partitions that a given credential may access.

Credential	Attribute
Account	**ALIST** (allows credential to access specified list of accounts
Partition	**PLIST** (allows credential to access specified list of partitions)
QoS	QLIST (allows credential to access specified list of QoSes)

Example 3-1:

```
USERCFG[bob]   ALIST=jupiter,quantum
USERCFG[steve] ALIST=quantum
```

Account-based access lists are only enforced if using an allocation manager or if the ENFORCEACCOUNTACCESS parameter is set to "TRUE."

Assigning Default Credentials

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

Credential	Attribute
Account	**ADEF** (specifies default account)
Class	**CDEF** (specifies default class)
QoS	**QDEF** (specifies default QoS)

Example 3-2:

```
# user bob can access accounts a2, a3, and a6. If no account is explicitly requested,
# his job will be assigned to account a3
USERCFG[bob]   ALIST=a2,a3,a6 ADEF=a3
# user steve can access accounts a14, a7, a2, a6, and a1. If no account is explicitly
# requested, his job will be assigned to account a2
USERCFG[steve] ALIST=a14,a7,a2,a6,a1 ADEF=a2
```

Specifying Credential Membership Lists

As an alternate to specifying access lists, administrators may also specify membership lists. This allows a credential to specify who can access it rather than allowing each credential to specify which

credentials it can access. Membership lists are controlled using the **MEMBERULIST**, **EXCLUDEUSERLIST** and **REQUIREDUSERLIST** attributes, shown in the following table:

Credential	Attribute
User	---
Account, Group, QoS	**MEMBERULIST**
Class	EXCLUDEUSERLIST and REQUIREDUSERLIST

Example 3-3:

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

Example 3-4: Controlling Partition Access on a Per User Basis

A site may specify the user `john` may access partitions `atlas`, `pluto`, and `zeus` and will default to partition `pluto`. To do this, include the following line in the configuration file:

```
USERCFG[john] PLIST=atlas,pluto,zeus
```

Example 3-5: Controlling QoS Access on a Per Group Basis

A site may also choose to allow everyone in the group `staff` to access QoS `standard` and `special` with a default QoS of `standard`. To do this, include the following line in the configuration file:

```
GROUPCFG[staff] QLIST=standard,special QDEF=standard
```

Example 3-6: Controlling Resource Access on a Per Account Basis

An organization wants to allow everyone in the account `omega3` to access nodes 20 through 24. To do this, include the following in the configuration file:

```
ACCOUNTCFG[omega3] MEMBERULIST=johnh,stevek,jenp
SRCFG[omega3]   HOSTLIST=r:20-24 ACCOUNTLIST=omega3
```

Credential Statistics

Full statistics are maintained for each credential instance. These statistics record current and historical resource usage, level of service delivered, accuracy of requests, and many other aspects of workload. Note, though, that you must explicitly enable credential statistics as they are not tracked by default. You can enable credential statistics by including the following in the configuration file:

```
USERCFG[DEFAULT]        ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]       ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]     ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]       ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]         ENABLEPROFILING=TRUE
```

Job Defaults, Credential State, and General Configuration

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

COMMENT	
Description	Associates a comment string with the target credential.
Example	`USERCFG[steve] COMMENT='works for boss, provides good service'` `CLASSCFG[i3]  COMMENT='queue for I/O intensive workload'`

HOLD	
Description	Specifies a hold should be placed on all jobs associated with the target credential. The order in which this **HOLD** attribute is evaluated depends on the following credential precedence: USERCFG, GROUPCFG, ACCOUNTCFG, CLASSCFG, QOSCFG, USERCFG *[DEFAULT]*, GROUPCFG*[DEFAULT]*, ACCOUNTCFG*[DEFAULT]*, CLASSCFG*[DEFAULT]*, QOSCFG*[DEFAULT]*.
Example	`GROUPCFG[bert] HOLD=yes`

JOBFLAGS	
Description	Assigns the specified job flag to all jobs with the associated credential.
Example	`CLASSCFG[batch] JOBFLAGS=suspendable` `QOSCFG[special] JOBFLAGS=restartable`

NOSUBMIT	
Description	Specifies whether jobs belonging to this credential can submit jobs using msub.
Example	`ACCOUNTCFG[general]  NOSUBMIT=TRUE` `CLASSCFG[special]  NOSUBMIT=TRUE`

OVERRUN	
Description	Specifies the amount of time a job may exceed its wallclock limit before being terminated. (Only applies to user and class credentials.)
Example	`CLASSCFG[bigmem] OVERRUN=00:15:00`

VARIABLE	
Description	Specifies attribute-value pairs associated with the specified credential. These variables may be used in triggers and other interfaces to modify system behavior.
Example	`GROUPCFG[staff] VARIABLE='nocharge=true'`

Credentials may carry additional configuration information. They may specify that detailed statistical profiling should occur, that submitted jobs should be held, or that corresponding jobs should be marked as preemptible.

User Credential

The user credential is the fundamental credential within a workload manager; each job requires an association with exactly one user. In fact, the user credential is the only required credential in Moab; all others are optional. In most cases, the job's user credential is configured within or managed by the operating system itself, although Moab may be configured to obtain this information from an independent security and identity management service.

As the fundamental credential, the user credential has a number of unique attributes.

- Role
- Email Address
- Disable Moab User Email

Role

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

Email Address

Facilities exist to allow user notification in the event of job or system failures or under other general conditions. This attribute allows these notifications to be mailed directly to the target user.

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

Disable Moab User Email

You can disable Moab email notifications for a specific user.

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

Group Credential

The group credential represents an aggregation of users. User-to-group mappings are often specified by the operating system or resource manager and typically map to a user's UNIX group ID. However, user-to-group mappings may also be provided by a security and identity management service, or you can specify such directly within Moab.

With many resource managers such as TORQUE, PBSPro, and LSF, the group associated with a job is either the user's active primary group as specified within the operating system or a group that is explicitly requested at job submission time. When a secondary group is requested, the user's default group and associated policies are not taken into account. Also note that a job may only run under one group. If more constraining policies are required for these systems, an alternate aggregation scheme such as the use of Account or QoS credentials is recommended.

To submit a job as a secondary group, refer to your local resource manager's job submission options. For TORQUE users, see the `group_list=g_list` option of the qsub -W command.

Account Credential

The account credential is also referred to as the project. This credential is generally associated with a group of users along the lines of a particular project for accounting and billing purposes. User-to-accounting mapping may be obtained from a resource manager or allocation manager, or you can configure it directly within Moab. Access to an account can be controlled via the **ALIST** and **ADEF** credential attributes specified via the Identity Manager or the `moab.cfg` file.

The MANAGERS attribute (applicable only to the account and class credentials) allows an administrator to assign a user the ability to manage jobs inside the credential, as if the user is the job owner.

Example 3-7: MANAGERS Attribute

```
ACCOUNTCFG[general]  MANAGERS=ops
ACCOUNTCFG[special]  MANAGERS=stevep
```

If a user is able to access more than one account, the desired account can be specified at job submission time using the resource-manager specific attribute. For example, with TORQUE this is accomplished using the -A argument to the qsub command.

Example 3-8: Enforcing Account Usage

Job-to-account mapping can be enforced using the **ALIST** attribute and the ENFORCEACCOUNTACCESS parameter.

```
USERCFG[john]    ALIST=proj1,proj3
USERCFG[steve]   ALIST=proj2,proj3,proj4
USERCFG[brad]    ALIST=proj1
USERCFG[DEFAULT] ALIST=proj2
ENFORCEACCOUNTACCESS TRUE
...
```

Class Credential

- Class Job Defaults
- Per Job Min/Max Limits
- Resource Access
- Class Membership Constraints
- Attributes Enabling Class Access to Other Credentials
- Special Class Attributes (such as Managers and Job Prologs)
- Setting Default Classes
- Creating a Remap Class
- Class Attribute Overview
- Enabling Queue Complex Functionality

The concept of the class credential is derived from the resource manager class or queue object. Classes differ from other credentials in that they more directly impact job attributes. In standard HPC usage, a user submits a job to a class and this class imposes a number of factors on the job. The attributes of a class may be specified within the resource manager or directly within Moab. Class attributes include the following:

- Job Defaults
- Per Job Min/Max Limits
- Resource Access Constraints
- Class Membership Constraints
- Attributes Enabling Class Access to Other Credentials
- Special Class Attributes

When using SLURM, Moab classes have a one-to-one relationship with SLURM partitions of the same name.

For all classes configured in Moab, a resource manager queue with the same name should be created.

When TORQUE reports a new queue to Moab a class of the same name is automatically applied to all nodes.

Class Job Defaults

Classes can be assigned to a default job template that can apply values to job attributes not explicitly specified by the submitter. Additionally, you can specify shortcut attributes from the table that follows:

Attribute	Description
DEFAULT.ATTR	Job Attribute
DEFAULT.DISK	Required Disk (in MB)
DEFAULT.EXT	Job RM Extension
DEFAULT.FEATURES	Required Node Features/Properties
DEFAULT.GRES	Required Consumable Generic Resources
DEFAULT.MEM	Required Memory/RAM (in MB)
DEFAULT.NODESET	Node Set Specification
DEFAULT.PROC	Required Processor Count
DEFAULT.TPN	Tasks Per Node
DEFAULT.WCLIMIT	Wallclock Limit

Defaults set in a class/queue of the resource manager will override the default values of the corresponding class/queue specified in Moab.

RESOURCELIMITPOLICY must be configured in order for the CLASSCFG limits to take effect.

Example 3-9:

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

Per Job Min/Max Limits

Classes can be assigned a minimum and a maximum job template that constrains resource requests. Jobs submitted to a particular queue must meet the resource request constraints of these templates. If a job submission exceeds these limits, the entire job submit fails.

Limit	Description
MAX.ARRAYSUBJOBS	Max Allowed Jobs in an Array

Limit	Description
MAX.CPUTIME	Max Allowed Utilized CPU Time
MAX.NODE	Max Allowed Node Count
MAX.PROC	Max Allowed Processor Count
MAX.PS	Max Requested Processor-Seconds
MIN.NODE	Min Allowed Node Count
MIN.PROC	Min Allowed Processor Count
MIN.PS	Min Requested Processor-Seconds
MIN.TPN	Min Tasks Per Node
MIN.WCLIMIT	Min Requested Wallclock Limit
MAX.WCLIMIT	Max Requested Wallclock Limit

The parameters listed in the preceding table are for classes 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.

Resource Access

Classes may be associated with a particular set of compute resources. Consequently, jobs submitted to a given class may only use listed resources. This may be handled at the resource manager level or via the CLASSCFG HOSTLIST attribute.

Class Membership Constraints

Classes may be configured at either the resource manager or scheduler level to only allow select users and groups to access them. Jobs that do not meet these criteria are rejected. If specifying class membership/access at the resource manager level, see the respective resource manager documentation. Moab automatically detects and enforces these constraints. If specifying class membership/access at the scheduler level, use the **REQUIREDUSERLIST** or **EXCLUDEUSERLIST** attributes of the CLASSCFG parameter.

Under most resource managers, jobs must always be a member of one and only one class.

Attributes Enabling Class Access to Other Credentials

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

Special Class Attributes

The class object also possesses a few unique attributes including JOBPROLOG, **JOBEPILOG**, RESFAILPOLICY, and DISABLEAM attributes described in what follows:

MANAGERS

Users listed via the **MANAGERS** attribute are granted full control over all jobs submitted to or running within the specified class.

```
# allow john and steve to cancel and modify all jobs submitted to the class/queue
special
CLASSCFG[special] MANAGERS=john,steve
```

In particular, a class manager can perform the following actions on jobs within a class/queue:

- view/diagnose job (checkjob)
- cancel, requeue, suspend, resume, and checkpoint job (mjobctl)
- modify job (mjobctl)

JOBPROLOG

The **JOBPROLOG** class performs a function similar to the resource manager level job prolog feature; however, there are some key differences:

- Moab prologs execute on the head node; resource manager prologs execute on the nodes allocated to the job.
- Moab prologs execute as the primary Moab administrator, resource manager prologs execute as root.
- Moab prologs can incorporate cluster environment information into their decisions and actions. (See Valid Variables.)
- Unique Moab prologs can be specified on a per class basis.
- Job start requests are not sent to the resource manager until the Moab job prolog is successfully completed.
- Error messages generated by a Moab prolog are attached to jobs and associated objects; stderr from prolog script is attached to job.
- Moab prologs have access to Moab internal and peer services.

Valid epilog and prolog variables are:

Variable	Description
$TIME	Time that the trigger launches
$HOME	Moab home directory
$USER	User name the job is running under
$JOBID	Unique job identifier
$HOSTLIST	Entire host list for job
$MASTERHOST	Master host for job

The **JOBPROLOG** class attribute allows a site to specify a unique per-class action to take before a job is allowed to start. This can be used for environmental provisioning, pre-execution resource checking, security management, and other functions. Sample uses may include enabling a VLAN, mounting a global file system, installing a new application or virtual node image, creating dynamic storage partitions, or activating job specific software services.

A prolog is considered to have failed if it returns a negative number. If a prolog fails, the associated job will not start.

If a prolog executes successfully, the associated epilog is guaranteed to start, even if the job fails for any reason. This allows the epilog to undo any changes made to the system by the prolog.

Job Prolog Examples

```
# explicitly specify prolog arguments for special epilog
CLASSCFG[special] JOBPROLOG='$TOOLSDIR/specialprolog.pl $JOBID $HOSTLIST'
# use default prolog arguments for batch prolog
CLASSCFG[batch]   JOBPROLOG=$TOOLSDIR/batchprolog.pl
```

JOBEPILOG

The Moab epilog is nearly identical to the prolog in functionality except that it runs after the job completes within the resource manager but before the scheduler releases the allocated resources for use by subsequent jobs. It is commonly used for job clean-up, file transfers, signaling peer services, and undoing other forms of resource customization.

An epilog is considered to have failed if it returns a negative number. If an epilog fails, the associated job will be annotated and a message will be sent to administrators.

RESFAILPOLICY

This policy allows specification of the action to take on a per-class basis when a failure occurs on a node allocated to an actively running job. See the Node Availability Overview for more information.

DISABLEAM

You can disable allocation management for jobs in specific classes by setting the **DISABLEAM** class attribute to *TRUE*. For all jobs outside of the specified classes, allocation enforcement will continue to be enforced.

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

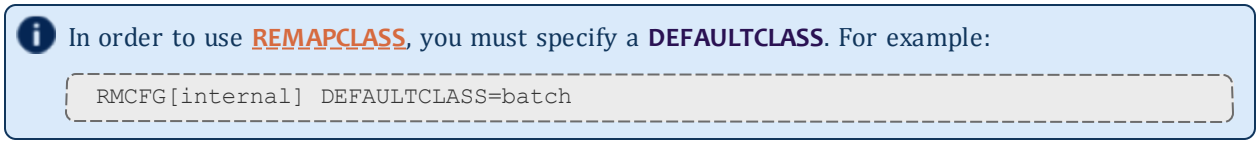
Setting Default Classes

In many cases, end-users do not want to be concerned with specifying a job class/queue. This is often handled by defining a default class. Whenever a user does not explicitly submit a job to a particular class, a default class, if specified, is used. In resource managers such as TORQUE, this can be done at the resource manager level and its impact is transparent to the scheduler. The default class can also be enabled within the scheduler on a per resource manager or per user basis. To set a resource manager default class within Moab, use the **DEFAULTCLASS** attribute of the RMCFG parameter. For per user defaults, use the **CDEF** attribute of the USERCFG parameter.

Creating a Remap Class

If a single default class is not adequate, Moab provides more flexible options with the REMAPCLASS parameter. If this parameter is set and a job is submitted to the remap class, Moab attempts to determine the final class to which a job belongs based on the resources requested. If a remap class is specified, Moab compares the job's requested nodes, processors, memory, and node features with the class's corresponding minimum and maximum resource limits. Classes are searched in the order in which they are defined; when the first match is found, Moab assigns the job to that class.

Because Moab remaps at job submission, updates you make to job requirements after submission will not cause any class changes. Moab does not restart the process.


In order to use REMAPCLASS, you must specify a **DEFAULTCLASS**. For example:

```
RMCFG[internal] DEFAULTCLASS=batch
```

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

```
# You must specify a default class in order to use remap classes
RMCFG[internal]   DEFAULTCLASS=batch

# Jobs submitted to 'batch' should be remapped
REMAPCLASS        batch

# stevens only queue
CLASSCFG[stevens] REQ.FEATURES=stevens REQUIREDUSERLIST=stevens,stevens2
```

```
# Special queue for I/O nodes
CLASSCFG[io]      MAX.PROC=8 REQ.FEATURES=io

# General access queues
CLASSCFG[quick]   MIN.PROC=2 MAX.PROC=8 REQ.FEATURES=fast|short
CLASSCFG[medium]  MIN.PROC=2 MAX.PROC=8
CLASSCFG[DEFAULT] MAX.PROC=64
...
```

The following attributes can be used to remap jobs to different classes:

- **MIN.PROC**
- **MAX.PROC**
- **MIN.WCLIMIT**
- **MAX.WCLIMIT**
- **REQ.FEATURES**
- **REQ.FLAGS=INTERACTIVE**
- **REQUIREDUSERLIST**

If the parameter REMAPCLASSLIST is set, then only the listed classes are searched and they are searched in the order specified by this parameter. If none of the listed classes are valid for a particular job, that job retains its original class.

The remap class only works with resource managers that allow dynamic modification of a job's assigned class/queue.

If default credentials are specified on a remap class, a job submitted to that class will inherit those credentials. If the destination class has different defaults credentials, the new defaults override the original settings. If the destination class does not have default credentials, the job maintains the defaults inherited from the remap class.

Class Attribute Overview

The following table enumerates the different parameters for CLASSCFG.

Setting DEFAULT.* on a class does not assign resources or features to that class. Rather, it specifies resources that jobs will inherit when they are submitted to the class without their own resource requests. To configure features, use NODECFG.

DEFAULT.ATTR	
Format	<ATTRIBUTE>[,<ATTRIBUTE>]...
Description	One or more comma-delimited generic job attributes.

DEFAULT.ATTR	
Example	---

DEFAULT.DISK	
Format	<INTEGER>
Description	Default amount of requested disk space.
Example	---

DEFAULT.EXT	
Format	<STRING>
Description	Default job RM extension.
Example	---

DEFAULT.FEATURES	
Format	Comma-delimited list of features.
Description	Default list of requested node features (a.k.a, node properties). This only applies to compute resource reqs.
Example	---

DEFAULT.GRES	
Format	<STRING>[<COUNT>][,<STRING>[<COUNT>]]...
Description	Default list of per task required consumable generic resources.
Example	`CLASSCFG[viz] DEFAULT.GRES=viz:2`

DEFAULT.MEM	
Format	<INTEGER> (in MB)
Description	Default amount of requested memory.
Example	---

DEFAULT.NODE	
Format	<INTEGER>
Description	Default required node count.
Example	`CLASSCFG[viz] DEFAULT.NODE=5` When a user submits a job to the `viz` class without a specified node count, the job is assigned 5 nodes.

DEFAULT.NODESET	
Format	<SETTYPE>:<SETATTR>[:<SETLIST>[,<SETLIST>]...]
Description	Default node set.
Example	`CLASSCFG[amd]` `DEFAULT.NODESET=ONEOF:FEATURE:ATHLON,OPTERON`

DEFAULT.PROC	
Format	<INTEGER>
Description	Default number of requested processors.
Example	---

DEFAULT.TPN	
Format	<INTEGER>
Description	Default number of tasks per node.
Example	---

DEFAULT.WCLIMIT	
Format	<INTEGER>
Description	Default wallclock limit.
Example	---

EXCL.FEATURES	
Format	Comma- or pipe-delimited list of node features.
Description	Set of excluded (disallowed) features. If delimited by commas, reject job if all features are requested; if delimited by the pipe symbol (|), reject job if at least one feature is requested.
Example	`CLASSCFG[intel] EXCL.FEATURES=ATHLON,AMD`

EXCL.FLAGS	
Format	Comma-delimited list of job flags.
Description	Set of excluded (disallowed) job flags. Reject job if any listed flags are set.
Example	`CLASSCFG[batch] EXCL.FLAGS=INTERACTIVE`

EXCLUDEUSERLIST	
Format	Comma-delimited list of users.

EXCLUDEUSERLIST	
Description	List of users not permitted access to class.
Example	---

FORCENODEACCESSPOLICY	
Format	one of *SINGLETASK*, *SINGLEJOB*, *SINGLEUSER*, or *SHARED*
Description	Node access policy associated with queue. If set, this value overrides any per job settings specified by the user at the job level. (See Node Access Policy overview for more information.)
Example	`CLASSCFG[batch] FORCENODEACCESSPOLICY=SINGLEJOB`

FSCAP	
Format	<DOUBLE>[%]
Description	See fairshare policies specification.
Example	---

FSTARGET	
Format	<DOUBLE>[%]
Description	See fairshare policies specification.
Example	---

HOSTLIST	
Format	Host expression, or comma-delimited list of hosts or host ranges.
Description	List of hosts associated with a class. If specified, Moab constrains the availability of a class to only nodes listed in the class host list.

HOSTLIST	
Example	`CLASSCFG[batch] HOSTLIST=r:abs[45-113]`

JOBEPILOG	
Format	<STRING>
Description	Scheduler level job epilog to be run after job is completed by resource manager. (See special class attributes.)
Example	---

JOBFLAGS	
Format	Comma-delimited list of job flags.
Description	See the flag overview for a description of legal flag values.
Example	`CLASSCFG[batch] JOBFLAGS=restartable`

JOBPROLOG	
Format	<STRING>
Description	Scheduler level job prolog to be run before job is started by resource manager. (See special class attributes.)
Example	---

MANAGERS	
Format	<USER>[,<USER>]...
Description	Users allowed to control, cancel, preempt, and modify jobs within class/queue. (See special class attributes.)
Example	`CLASSCFG[fast] MANAGERS=root,kerry,e43`

MAXJOB	
Format	<INTEGER>
Description	Maximum number of jobs allowed in the class.
Example	---

MAXPROCPERNODE	
Format	<INTEGER>
Description	Maximum number of processors requested per node. May optionally include node names to articulate which nodes have a specific limit.

Example	`CLASSCFG[cpu] MAXPROCPERNODE=20  # When using this class, limit 20 for all nodes`
	`CLASSCFG[cpu] MAXPROCPERNODE[n1,n2]=20 MAXPROCPERNODE[n3]=10  # When using this class, limit 20 for n1 & n2 and limit 10 for n3`
	`CLASSCFG[cpu] MAXPROCPERNODE[n1,n2]=20 MAXPROCPERNODE=10 # When using this class, limit 20 for n1 & n2 and limit 10 for all other nodes`

MAX.CPUTIME	
Format	<INTEGER>
Description	Maximum allowed utilized CPU time.
Example	---

MAX.NODE	
Format	<INTEGER>
Description	Maximum number of requested nodes per job. (Also used when REMAPCLASS is set to correctly route the job.)

MAX.NODE	
Example	`CLASSCFG[batch] MAX.NODE=64` Deny jobs requesting over 64 nodes access to the class *batch*.

MAX.PROC	
Format	<INTEGER>
Description	Maximum number of requested processors per job. (Also used when REMAPCLASS is set to correctly route the job.)
Example	`CLASSCFG[small] MAX.PROC[USER]=3,6`

MAX.PS	
Format	<INTEGER>
Description	Maximum requested processor-seconds.
Example	---

MAX.WCLIMIT	
Format	[[[DD:]HH:]MM:]SS
Description	Maximum allowed wallclock limit per job. (Also used when REMAPCLASS is set to correctly route the job.)
Example	`CLASSCFG[long] MAX.WCLIMIT=96:00:00`

MIN.NODE	
Format	<INTEGER>
Description	Minimum number of requested nodes per job. (Also used when REMAPCLASS is set to correctly route the job.)

MIN.NODE	
Example	`CLASSCFG[dev] MIN.NODE=16` Jobs must request at least 16 nodes to be allowed to access the class.

MIN.PROC	
Format	<INTEGER>
Description	Minimum number of requested processors per job. (Also used when REMAPCLASS is set to correctly route the job.)
Example	`CLASSCFG[dev] MIN.PROC=32` Jobs must request at least 32 processors to be allowed to access the class.

MIN.PS	
Format	<INTEGER>
Description	Minimum requested processor-seconds.
Example	---

MIN.TPN	
Format	<INTEGER>
Description	Minimum required tasks per node per job.
Example	---

MIN.WCLIMIT	
Format	[[[DD:]HH:]MM:]SS
Description	Minimum required wallclock limit per job. (Also used when REMAPCLASS is set to correctly route the job.)

MIN.WCLIMIT	
Example	---

NODEACCESSPOLICY	
Format	one of *SINGLETASK*, *SINGLEJOB*, *SINGLEUSER*, or *SHARED*
Description	Default node access policy associated with queue. This value will be overridden by any per job settings specified by the user at the job level. (See Node Access Policy overview.)
Example	`CLASSCFG[batch] NODEACCESSPOLICY=SINGLEJOB`

PARTITION	
Format	<STRING>
Description	Partition name where jobs associated with this class must run.
Example	`CLASSCFG[batch] PARTITION=p12`

PRIORITY	
Format	<INTEGER>
Description	Priority associated with the class. (See Priority overview.)
Example	`CLASSCFG[batch] PRIORITY=1000`

QDEF	
Format	<QOSID>
Description	Default QoS for jobs submitted to this class. You may specify a maximum of four **QDEF** entries per credential. Any QoSes specified after the fourth will not be accepted. In addition to classes, you may also specify **QDEF** for accounts, groups, and users.

QDEF	
Example	`CLASSCFG[batch] QDEF=base` Jobs submitted to class *batch* that do not explicitly request a QoS will have the QoS *base* assigned.

QLIST	
Format	<QOSID>[,<QOSID>]...
Description	List of accessible QoSes for jobs submitted to this class.
Example	`CLASSCFG[batch] QDEF=base` `QLIST=base,fast,special,bigio`

REQ.FEATURES	
Format	Comma- or pipe-delimited list of node features.
Description	Set of required features. If delimited by commas, all features are required; if delimited by the pipe symbol (|), at least one feature is required.
Example	`CLASSCFG[amd] REQ.FEATURES=ATHLON,AMD`

REQ.FLAGS	
Format	REQ.FLAGS can be used with only the INTERACTIVE flag.
Description	Sets the INTERACTIVE flag on jobs in this class.
Example	`CLASSCFG[orion] REQ.FLAGS=INTERACTIVE`

REQUIREDACCOUNTLIST	
Format	Comma-delimited list of accounts.
Description	List of accounts allowed to access and use a class (analogous to ***LIST** for other credentials).

REQUIREDACCOUNTLIST	
Example	`CLASSCFG[jasper] REQUIREDACCOUNTLIST=testers,development`

REQUIREDUSERLIST	
Format	Comma-delimited list of users.
Description	List of users allowed to access and use a class (analogous to ***LIST** for other credentials).
Example	`CLASSCFG[jasper] REQUIREDUSERLIST=john,u13,steve,guest`

REQUIREDQOSLIST	
Format	Comma-delimited list of QoSes
Description	List of QoSes allowed to access and use a class (analogous to ***LIST** for other credentials). The number of unique QoSes is limited by the Moab Maximum ACL limit, which defaults to 32.
Example	`CLASSCFG[jasper] REQUIREDQOSLIST=hi,lo`

SYSPRIO	
Format	<INTEGER>
Description	Value of system priority applied to every job submitted to this class.
Example	`CLASSCFG[special] SYSPRIO=100`

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

WCOVERRUN	
Description	Tolerated amount of time beyond the specified wallclock limit.
Example	---

Enabling Queue Complex Functionality

Queue complexes allow an organization to build a hierarchy of queues and apply certain limits and rules to collections of these queues. Moab supports this functionality in two ways. The first way, queue mapping, is very simple but limited in functionality. The second method provides very rich functionality but requires more extensive configuration using the Moab hierarchical fairshare facility.

Queue Mapping

Queue mapping allows collections of queues to be mapped to a parent credential object against which various limits and policies can be applied, as in the following example.

```
QOSCFG[general]   MAXIJOB[USER]=14  PRIORITY=20
QOSCFG[prio]      MAXIJOB[USER]=8   PRIORITY=2000
# group short, med, and long jobs into 'general' QoS
CLASSCFG[short]   QDEF=general FSTARGET=30
CLASSCFG[med]     QDEF=general FSTARGET=40
CLASSCFG[long]    QDEF=general FSTARGET=30 MAXPROC=200
# group interactive and debug jobs into 'prio' QoS
CLASSCFG[inter]   QDEF=prio
CLASSCFG[debug]   QDEF=prio
CLASSCFG[premier] PRIORITY=10000
```

QoS Credential

The concept of a quality of service (QoS) credential is unique to Moab and is not derived from any underlying concept or peer service. In most cases, the QoS credential is used to allow a site to set up a selection of service levels for end-users to choose from on a long-term or job-by-job basis. QoSes differ from other credentials in that they are centered around special access where this access may allow use of additional services, additional resources, or improved responsiveness. Unique to this credential, organizations may also choose to apply different charge rates to the varying levels of service available within each QoS. As QoS is an internal credential, all QoS configuration occurs within Moab.

QoS access and QoS defaults can be mapped to users, groups, accounts, and classes, allowing limited service offering for key users. As mentioned, these services focus around increasing access to special scheduling capabilities & additional resources and improving job responsiveness. At a high level, unique QoS attributes can be broken down into the following:

- Usage Limit Overrides
- Service Targets
- Privilege Flags

- Charge Rate
- Access Controls

QoS Usage Limit Overrides

All credentials allow specification of job limits. In such cases, jobs are constrained by the most limiting of all applicable policies. With QoS override limits, however, jobs are limited by the override, regardless of other limits specified.

QoS Service Targets

Service targets cause the scheduler to take certain job-related actions as various responsiveness targets are met. Targets can be set for either job queue time or job expansion factor and cause priority adjustments, reservation enforcement, or preemption activation. In strict service centric organizations, Moab can be configured to trigger various events and notifications in the case of failure by the cluster to meet responsiveness targets.

QoS Privilege Flags

QoSes can provide access to special capabilities. These capabilities include preemption, job deadline support, backfill, next to run priority, guaranteed resource reservation, resource provisioning, dedicated resource access, and many others. See the complete list in the QoS Facility Overview section.

QoS Charge Rate

Associated with the QoSes many privileges is the ability to assign end-users costs for the use of these services. This charging can be done on a per-QoS basis and may be specified for both dedicated and use-based resource consumption. The Per QoS Charging section covers more details on QoS level costing configuration while the Charging and Allocation Management section provides more details regarding general single cluster and multi-cluster charging capabilities.

QoS Access Controls

QoS access control can be enabled on a per QoS basis using the MEMBERULIST attribute or specified on a *per-requestor* basis using the **QDEF** and **QLIST** attributes of the USERCFG, GROUPCFG, ACCOUNTCFG, and CLASSCFG parameters. See Managing QoS Access for more detail.

Related topics

- Identity Manager Interface
- Usage Limits

3.5.1 Job Attributes/Flags Overview

Job Attributes

FLAGS	
Format:	<FLAG>[:<FLAG>]...
Default:	---
Description:	Specifies job specific flags.
Example:	`FLAGS=ADVRES:RESTARTABLE` *The job can be restarted and should only utilize reserved resources.*

PLIST*	
Format:	<PARTITION_NAME>[^|&] [:<PARTITION_NAME>[^|&]]...
Default:	*[ALL]*
Description:	Specifies the list of partitions the object can access. If no partition list is specified, the object is granted default access to all partitions.
Example:	`PLIST=OldSP:Cluster1:O3K` *The object can access resources located in the OldSP, Cluster1, and/or O3K partitions.*

QDEF	
Format:	<QOS_NAME>
Default:	*[DEFAULT]*

QDEF	
Description:	Specifies the default QoS associated with the object.
Example:	`QDEF=premium` *The object is assigned the default QoS premium.*

QLIST*	
Format:	<QOS_NAME>[^|&] [:<QOS_NAME>[^|&]]...
Default:	*<QDEF>*
Description:	Specifies the list of QoSes the object can access. If no QoS list is specified, the object is granted access only to its default partition.
Example:	`QLIST=premium:express:bottomfeeder` *The object can access any of the 3 QoSes listed.*

By default, jobs may access QoSes 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 QoSes accessible by any of the individual credentials. Thus the job's QoS access list, or QLIST, equals the 'or' of the user, group, and class QLIST's. (i.e., JOBQLIST = USERQLIST | GROUPQLIST | CLASSQLIST). If the ampersand symbol, '&', is associated with any list, this list is logically ANDed with the other lists. If the carat symbol, '^', is associated with any object QLIST, this list is exclusively set, regardless of other object access lists using the following order of precedence user, group, account, QoS, and class. These special symbols affect the behavior of both QoS and partition access lists.

Job Flags

ADVRES	
Format:	ADVRES[:<RESID>]
Default:	Use available resources where ever found, whether inside a reservation or not.

ADVRES	
Description:	Specifies the job may only utilize accessible, reserved resources. If <RESID> is specified, only resources in the specified reservation may be utilized.
Example:	`FLAGS=ADVRES:META.1` *The job may only utilize resources located in the META.1 reservation.*

ARRAYJOBPARLOCK	
Format:	---
Default:	---
Description:	Specifies that the job array being submitted should not span across multiple partitions. This locks all sub jobs of the array to a single partition. If you want to lock all job arrays to a single partition, specify the ARRAYJOBPARLOCK parameter in `moab.cfg` to force this behavior on a global scale.
Example:	`> msub -t moab.[1-5]%3 -l walltime=30,flags=arrayjobparlock`

ARRAYJOBPARSPAN	
Format:	---
Default:	---
Description:	Specifies that the job array being submitted should span across multiple partitions. This is the default behavior in Moab, unless the ARRAYJOBPARLOCK parameter is specified in `moab.cfg`. This job flag overrides the **ARRAYJOBPARLOCK** parameter so that job arrays can be allowed to span multiple partitions at submit time.
Example:	`> msub -t moab.[1-5]%3 -l walltime=30,flags=arrayjobparspan`

GRESONLY	
Format:	GRESONLY

GRESONLY	
Default:	False
Description:	Uses no compute resources such as processors, memory, and so forth; uses only generic resources.
Example:	`> msub -l gres=matlab,walltime=300`

IGNIDLEJOBRSV	
Format:	IGNIDLEJOBRSV
Default:	N/A
Description:	Only applies to QoS. **IGNIDLEJOBRSV** allows jobs to start without a guaranteed walltime. Instead, it overlaps the idle reservations of real jobs and is preempted 2 minutes before the real job starts.
Example:	`QOSCFG[standby] JOBFLAGS=IGNIDLEJOBRSV`

NOQUEUE	
Format:	NOQUEUE
Default:	Jobs remain queued until they are able to run.
Description:	Specifies that the job should be removed it is unable to allocate resources and start execution immediately.
Example:	`FLAGS=NOQUEUE` *The job should be removed unless it can start running at submit time.* This functionality is identical to the resource manager extension QUEUEJOB:FALSE.

NORMSTART	
Format:	NORMSTART

NORMSTART	
Default:	Moab passes jobs to a resource manager to schedule.
Description:	Specifies that the job is an internal system job and will not be started via an RM.
Example:	`FLAGS=NORMSTART` *The job begins running in Moab without a corresponding RM job.*

PREEMPTEE	
Format:	PREEMPTEE
Default:	Jobs may not be preempted by other jobs
Description:	Specifies that the job may be preempted by other jobs which have the *PREEMPTOR* flag set.
Example:	`FLAGS=PREEMPTEE` *The job may be preempted by other jobs which have the PREEMPTOR flag set.*

PREEMPTOR	
Format:	PREEMPTOR
Default:	Jobs may not preempt other jobs
Description:	Specifies that the job may preempt other jobs which have the **PREEMPTEE** flag set.
Example:	`FLAGS=PREEMPTOR` *The job may preempt other jobs which have the PREEMPTEE flag set.*

RESTARTABLE	
Format:	RESTARTABLE

RESTARTABLE	
Default:	Jobs may not be restarted if preempted.
Description:	Specifies jobs can be *requeued* and later restarted if preempted.
Example:	`FLAGS=RESTARTABLE` *The associated job can be preempted and restarted at a later date.*

SUSPENDABLE	
Format:	SUSPENDABLE
Default:	Jobs may not be suspended if preempted.
Description:	Specifies jobs can be *suspended* and later resumed if preempted.
Example:	`FLAGS=SUSPENDABLE` *The associated job can be suspended and resumed at a later date.*

SYSTEMJOB	
Format:	SYSTEMJOB
Default:	N/A
Description:	Creates an internal system job that does not require resources.
Example:	`FLAGS=SYSTEMJOB`

WIDERSVSEARCHALGO	
Format:	<BOOLEAN>
Default:	---

WIDERSVSEARCHALGO	
Description:	When Moab is determining when and where a job can run, it either searches for the most resources or the longest range of resources. In almost all cases searching for the longest range is ideal and returns the soonest starttime. In some rare cases, however, a particular job may need to search for the most resources. In those cases this flag can be used to have the job find the soonest starttime. The flag can be specified at submit time, or you can use mjobctl -m to modify the job after it has been submitted. See the RSVSEARCHALGO parameter.

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

Related topics

- Setting Per-Credential Job Flags

4.0 Scheduler Commands

Moab Commands

Command	Description
checkjob	Provide detailed status report for specified job
checknode	Provide detailed status report for specified node
mcredctl	Controls various aspects about the credential objects within Moab
mdiag	Provide diagnostic reports for resources, workload, and scheduling
mjobctl	Control and modify job
mnodectl	Control and modify nodes
moab	Control the Moab daemon
mrmctl	Query and control resource managers
mrsvctl	Create, control and modify reservations
mschedctl	Modify scheduler state and behavior
mshow	Displays various diagnostic messages about the system and job queues
mshow -a	Query and show available system resources
msub	Scheduler job submission
mvcctl	Create, modify, and delete VCs
mvmctl	Create, control and modify VMs
showbf	Show current resource availability

Command	Description
showhist.moab.pl	Show past job information
showq	Show queued jobs
showres	Show existing reservations
showstart	Show estimates of when job can/will start
showstate	Show current state of resources
showstats	Show usage statistics
showstats -f	Show various tables of scheduling/system performance

Moab command options

For many Moab commands, you can use the following options to specify that Moab will run the command in a different way or different location from the configured default. These options do not change your settings in the configuration file; they override the settings for this single instance of the command.

Option	Description
--about	Displays build and version information and the status of your Moab license
--help	Displays usage information about the command
--host=*<server-HostName>*	Causes Moab to run the client command on the specified host
--loglevel=*<logLevel>*	Causes Moab to write log information to STDERR as the client command is running. For more information, see **Logging Facilities** on page 588.
--msg=*<message>*	Causes Moab to annotate the action in the event log
--port=*<server-Port>*	Causes Moab to run the command using the port specified

Option	Description
--timeout= ***<seconds>***	Sets the maximum time that the client command will wait for a response from the Moab server
--version	Displays version information
--xml	Causes Moab to return the command output in XML format

Commands Providing Maui Compatibility

The following commands are deprecated. Click the link for respective deprecated commands to see the updated replacement command for each.

Command	Description
canceljob	Cancel job
changeparam	Change in memory parameter settings
diagnose	Provide diagnostic report for various aspects of resources, workload, and scheduling
releasehold	Release job defers and holds
releaseres	Release reservations
runjob	Force a job to run immediately
sethold	Set job holds
setqos	Modify job QoS settings
setres	Set an admin/user reservation
setspri	Adjust job/system priority of job
showconfig	Show current scheduler configuration

4.1 Status Commands

The status commands organize and present information about the current state and historical statistics of the scheduler, jobs, resources, users, and accounts. The following table presents the primary status commands and flags.

Command	Description
checkjob	Displays detailed job information such as job state, resource requirements, environment, constraints, credentials, history, allocated resources, and resource utilization.
checknode	Displays detailed node information such as node state, resources, attributes, reservations, history, and statistics.
mdiag -f	Displays summarized fairshare information and any unexpected fairshare configuration.
mdiag -j	Displays summarized job information and any unexpected job state.
mdiag -n	Displays summarized node information and any unexpected node state.
mdiag -p	Displays summarized job priority information.
mschedctl -f	Resets internal statistics.
showstats -f	Displays various aspects of scheduling performance across a job duration/job size matrix.
showq [-r|-i]	Displays various views of currently queued active, idle, and non-eligible jobs.
showstats -g	Displays current and historical usage on a per group basis.
showstats -u	Displays current and historical usage on a per user basis.
showstats -v	Displays high level current and historical scheduling statistics.

4.2 Job Management Commands

Moab shares job management tasks with the resource manager. Typically, the scheduler only modifies scheduling relevant aspects of the job such as partition access, job priority, charge account, and hold state. The following table covers the available job management commands. The Commands Overview lists all available commands.

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

Related topics

- Job State Definitions

4.3 Reservation Management Commands

Moab exclusively controls and manages all advance reservation features including both standing and administrative reservations. The following table covers the available reservation management commands.

Command	Description
mdiag -r	Displays summarized reservation information and any unexpected state.

Command	Description
mrsvctl	Reservation control.
mrsvctl -r	Removes reservations.
mrsvctl -c	Creates an administrative reservation.
showres	Displays information regarding location and state of reservations.

4.4 Policy/Configuration Management Commands

Moab allows dynamic modification of most scheduling parameters allowing new scheduling policies, algorithms, constraints, and permissions to be set at any time. Changes made via Moab client commands are temporary and are overridden by values specified in Moab configuration files the next time Moab is shut down and restarted. The following table covers the available configuration management commands.

Command	Description
mschedctl -l	Displays triggers, messages, and settings of all configuration parameters.
mschedctl	Controls the scheduler (behavior, parameters, triggers, messages).
mschedctl -m	Modifies system values.

4.5 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 host name. This can be done by using aliases (adding all node names to the `/etc/hosts` file on the system where Moab is running) and ensuring that the Linux resource manager reports the node with its global name rather than the Linux-specific one (`node01` rather than `node01-l`).

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

Related topics

- Commands Overview

4.6 Commands

checkjob

Synopsis

checkjob [-l *policylevel*] [-n *nodeid*] [-q *qosid*] [-r *reservationid*] [-v] [--flags=future] [--blocking] *jobid*

Overview

checkjob displays detailed job state information and diagnostic output for a specified job. Detailed information is available for queued, blocked, active, and recently completed jobs. The checkjob command shows the master job of an array as well as a summary of array sub-jobs, but does not display all sub-jobs. Use checkjob -v to display all job-array sub-jobs.

Access

This command can be run by level 1-3 Moab administrators for any job. Also, end users can use checkjob to view the status of their own jobs.

Arguments

--blocking	
Format	`--blocking`
Description	Do not use cache information in the output. The `--blocking` flag retrieves results exclusively from the scheduler.
Example	`> checkjob -v --blocking 1234` *Display real time data about job `1234`.*

--flags	
Format	`--flags=future`
Description	Evaluates future eligibility of job (ignore current resource state and usage limitations).
Example	`> checkjob -v --flags=future 6235` *Display reasons why idle job is blocked ignoring node state and current node utilization constraints.*

-l (Policy level)	
Format	*<POLICYLEVEL>* *HARD*, *SOFT*, or *OFF*
Description	Reports job start eligibility subject to specified throttling policy level.
Example	`> checkjob -l SOFT 6235` `> checkjob -l HARD 6235`

-n (NodeID)	
Format	*<NODEID>*
Description	Checks job access to specified node and preemption status with regards to jobs located on that node.
Example	`> checkjob -n node113 6235`

-q (QoS)	
Format	*<QOSID>*
Description	Checks job access to specified QoS *<QOSID>*.
Example	`> checkjob -q special 6235`

-r (Reservation)	
Format	*<RSVID>*
Description	Checks job access to specified reservation *<RSVID>*.
Example:	`> checkjob -r orion.1 6235`

-v (Verbose)	
Description	Sets verbose mode. If the job is part of an array, the `-v` option shows pertinent array information before the job-specific information (see Example 2 and Example 3 for differences between standard output and -v output). Specifying the double verbose (`-v -v`) displays additional information about the job. See the Output table for details.
Example	`> checkjob -v 6235`

Details

This command allows any Moab administrator to check the detailed status and resource requirements of an active, queued, or recently completed job. Additionally, this command performs numerous diagnostic checks and determines if and where the job could potentially run. Diagnostic checks include policy violations, reservation constraints, preemption status, and job to resource mapping. If a job cannot run, a text reason is provided along with a summary of how many nodes are and are not available. If the `-v` flag is specified, a node by node summary of resource availability will be displayed for idle jobs.

Job Eligibility

If a job cannot run, a text reason is provided along with a summary of how many nodes are and are not available. If the `-v` flag is specified, a node by node summary of resource availability will be displayed for idle jobs. For job level eligibility issues, one of the following reasons will be given:

Reason	Description
job has hold in place	one or more job holds are currently in place
insufficient idle procs	there are currently not adequate processor resources available to start the job
idle procs do not meet requirements	adequate idle processors are available but these do not meet job requirements
start date not reached	job has specified a minimum *start date* which is still in the future
expected state is not idle	job is in an unexpected state
state is not idle	job is not in the idle state
dependency is not met	job depends on another job reaching a certain state
rejected by policy	job start is prevented by a throttling policy

If a job cannot run on a particular node, one of the following 'per node' reasons will be given:

Reason	Description
Class	Node does not allow required job class/queue
CPU	Node does not possess required processors

Reason	Description
Disk	Node does not possess required local disk
Features	Node does not possess required node features
Memory	Node does not possess required real memory
Network	Node does not possess required network interface
State	Node is not Idle or Running

Reservation Access

The -r flag can be used to provide detailed information about job access to a specific reservation

Preemption Status

If a job is marked as a preemptor and the -v and -n flags are specified, checkjob will perform a job by job analysis for all jobs on the specified node to determine if they can be preempted.

Output

The checkjob command displays the following job attributes:

Attribute	Value	Description
Account	*<STRING>*	Name of account associated with job
Actual Run Time	[[[DD:]HH:]MM:]SS	Length of time job actually ran. This info is only displayed in simulation mode.
Allocated Nodes	Square bracket delimited list of node and processor ids	List of nodes and processors allocated to job
Applied Nodeset**	*<STRING>*	Node set used for job's node allocation
Arch	*<STRING>*	Node architecture required by job
Attr	square bracket delimited list of job attributes	Job Attributes (i.e. `[BACKFILL][PREEMPTEE]`)

Attribute	Value	Description
Available Memory**	*<INTEGER>*	The available memory requested by job. Moab displays the relative or exact value by returning a comparison symbol (>, <, >=, <=, or ==) with the value (i.e. `Available Memory <= 2048`).
Available Swap**	*<INTEGER>*	The available swap requested by job. Moab displays the relative or exact value by returning a comparison symbol (>, <, >=, <=, or ==) with the value (i.e. `Available Swap >= 1024`).
Average Utilized Procs*	*<FLOAT>*	Average load balance for a job
Avg Util Resources Per Task*	*<FLOAT>*	
BecameEligible	*<TIMESTAMP>*	The date and time when the job moved from Blocked to Eligible.
Bypass	*<INTEGER>*	Number of times a lower priority job with a later submit time ran before the job
CheckpointStartTime**	`[[[DD:]HH:]MM:]SS`	The time the job was first checkpointed
Class	[*<CLASS NAME>* *<CLASS COUNT>*]	Name of class/queue required by job and number of class initiators required per task.
Dedicated Resources Per Task*	Space-delimited list of *<STRING>*:*<INTEGER>*	Resources dedicated to a job on a per-task basis
Disk	*<INTEGER>*	Amount of local disk required by job (in MB)
Estimated Walltime	[[[DD:]HH:]MM:]SS	The scheduler's estimated walltime. In simulation mode, it is the actual walltime.
EnvVariables**	Comma-delimited list of *<STRING>*	List of environment variables assigned to job
Exec Size*	*<INTEGER>*	Size of job executable (in MB)

Attribute	Value	Description
Executable	*<STRING>*	Name of command to run
Features	Square bracket delimited list of *<STRING>*s	Node features required by job
Flags		
Group	*<STRING>*	Name of UNIX group associated with job
Holds	Zero or more of User, System, and Batch	Types of job holds currently applied to job
Image Size	*<INTEGER>*	Size of job data (in MB)
IWD (Initial Working Directory)	*<DIR>*	Directory to run the executable in
Job Messages**	*<STRING>*	Messages attached to a job
Job Submission**	*<STRING>*	Job script submitted to RM
Memory	*<INTEGER>*	Amount of real memory required per node (in MB)
Max Util Resources Per Task*	*<FLOAT>*	
NodeAccess*		
Nodecount	*<INTEGER>*	Number of nodes required by job
Opsys	*<STRING>*	Node operating system required by job
Partition Mask	ALL or colon delimited list of partitions	List of partitions the job has access to
PE	*<FLOAT>*	Number of processor-equivalents requested by job
Per Partition Priority**	Tabular	Table showing job template priority for each partition

Attribute	Value	Description
Priority Analysis**	Tabular	Table showing how job's priority was calculated: `Job PRIORITY* Cred( User:Group:Class) Serv (QTime)`
QOS	*<STRING>*	Quality of Service associated with job
Reservation	*<RSVID>* (*<TIME1>* - *<TIME2>* Duration: *<TIME3>*)	RESID specifies the reservation id, TIME1 is the relative start time, TIME2 the relative end time, TIME3 the duration of the reservation
Req	[*<INTEGER>*] TaskCount: *<INTEGER>* Partition: *<partition>*	A job requirement for a single type of resource followed by the number of tasks instances required and the appropriate partition
StartCount	*<INTEGER>*	Number of times job has been started by Moab
StartPriority	*<INTEGER>*	Start priority of job
StartTime	*<TIME>*	Time job was started by the resource management system
State	One of Idle, Starting, Running, etc. See **Job States** on page 28 for all possible values.	Current Job State
SubmitTime	*<TIME>*	Time job was submitted to resource management system
Swap	*<INTEGER>*	Amount of swap disk required by job (in MB)
Task Distribution*	Square bracket delimited list of nodes	
Time Queued		
Total Requested Nodes**	*<INTEGER>*	Number of nodes the job requested
Total Requested Tasks	*<INTEGER>*	Number of tasks requested by job
User	*<STRING>*	Name of user submitting job

Attribute	Value	Description
Utilized Resources Per Task*	*<FLOAT>*	
WallTime	[[[DD:]HH:]MM:]SS of [[[DD:]HH:]MM:]SS	Length of time job has been running out of the specified limit

In the above table, fields marked with an asterisk (*) are only displayed when set or when the `-v` flag is specified. Fields marked with two asterisks (**) are only displayed when set or when the `-v -v` flag is specified.

Example 4-1: checkjob 717

```
> checkjob 717
job 717
State: Idle
Creds:  user:jacksond  group:jacksond  class:batch
WallTime: 00:00:00 of 00:01:40
SubmitTime: Mon Aug 15 20:49:41
  (Time Queued  Total: 3:12:23:13  Eligible: 3:12:23:11)
TerminationDate:   INFINITY  Sat Oct 24 06:26:40
Total Tasks: 1
Req[0]  TaskCount: 1  Partition: ALL
Network: ---  Memory >= 0  Disk >= 0  Swap >= 0
Opsys: ---  Arch: ---  Features: ---

IWD:            /home/jacksond/moab/moab-4.2.3
Executable:     STDIN
Flags:          RESTARTABLE,NORMSTART
StartPriority:  5063
Reservation '717' (  INFINITY ->   INFINITY  Duration: 00:01:40)
Note:  job cannot run in partition base (idle procs do not meet requirements : 0 of 1
procs found)
idle procs:   4  feasible procs:   0
Rejection Reasons: [State        :    3][ReserveTime  :    1]
cannot select job 717 for partition GM (partition GM does not support requested class
batch)
```

The example job cannot be started for two different reasons.

- It is temporarily blocked from partition *base* because of node state and node reservation conflicts.
- It is permanently blocked from partition *GM* because the requested class *batch* is not supported in that partition.

Example 4-2: Using checkjob (no -v) on a job array master job:

```
checkjob array.1
job array.1

AName: array
Job Array Info:
  Name: array.1
```

```
Sub-jobs:         10
  Active:          6 ( 60.0%)
  Eligible:        2 ( 20.0%)
  Blocked:         2 ( 20.0%)
  Complete:        0 (  0.0%)
```

Example 4-3: Using checkjob -v on a job array master job:

```
$ checkjob -v array.1
job array.1

AName: array
Job Array Info:
  Name: array.1
  1 : array.1.1 : Running
  2 : array.1.2 : Running
  3 : array.1.3 : Running
  4 : array.1.4 : Running
  5 : array.1.5 : Running
  6 : array.1.6 : Running
  7 : array.1.7 : Idle
  8 : array.1.8 : Idle
  9 : array.1.9 : Blocked
  10 : array.1.10 : Blocked

Sub-jobs:         10
  Active:          6 ( 60.0%)
  Eligible:        2 ( 20.0%)
  Blocked:         2 ( 20.0%)
  Complete:        0 (  0.0%)
```

Related topics

- showhist.moab.pl - explains how to query for past job information
- Moab Client Installation - explains how to distribute this command to client nodes
- mdiag -j command - display additional detailed information regarding jobs
- showq command - showq high-level job summaries
- JOBCPURGETIME parameter - specify how long information regarding completed jobs is maintained
- diagnosing job preemption

checknode

Synopsis

checknode *options* *nodeID*
ALL

Overview

This command shows detailed state information and statistics for nodes that run jobs.

The following information is returned by this command:

Name	Description
Disk	Disk space available
Memory	Memory available
Swap	Swap space available
State	Node state
Opsys	Operating system
Arch	Architecture
Adapters	Network adapters available
Features	Features available
Classes	Classes available
StateTime	Time node has been in current state in HH:MM:SS notation
Downtime	Displayed only if downtime is scheduled
Load	CPU Load (Berkley one-minute load average)
TotalTime	Total time node has been detected since statistics initialization expressed in *HH:MM:SS* notation
UpTime	Total time node has been in an available (Non-Down) state since statistics initialization expressed in *HH:MM:SS* notation (percent of time up: UpTime/TotalTime)
ActiveTime	Total time node has been busy (allocated to active jobs) since statistics initialization expressed in *HH:MM:SS* notation (percent of time busy: BusyTime/TotalTime)
EffNodeAccessPolicy	Configured effective node access policy

After displaying this information, some analysis is performed and any unusual conditions are reported.

Access

By default, this command can be run by any Moab Administrator (see ADMINCFG).

Parameters

Name	Description
NODE	Node name you want to check. Moab uses regular expressions to return any node that contains the provided argument. For example, if you ran `checknode node1`, Moab would return information about `node1`, `node10`, `node100`, etc. If you want to limit the results to `node1` only, you would run `checknode "^node1$"`.

Flags

Name	Description
ALL	Returns checknode output on all nodes in the cluster.
-h	Help for this command.
-v	Returns verbose output.
--xml	Output in XML format. Same as mdiag -n --xml.

Example 4-4: checknode

```
> checknode P690-032
node P690-032

State:      Busy  (in current state for 11:31:10)
Configured Resources: PROCS: 1  MEM: 16G  SWAP: 2000M  DISK: 500G
Utilized   Resources: PROCS: 1
Dedicated  Resources: PROCS: 1
Opsys:      AIX       Arch:      P690
Speed:      1.00      CPULoad:   1.000
Network:    InfiniBand,Myrinet
Features:   Myrinet
Attributes: [Batch]
Classes:    [batch]

Total Time: 5:23:28:36  Up: 5:23:28:36 (100.00%)  Active: 5:19:44:22 (97.40%)

Reservations:
  Job '13678'(x1)  10:16:12:22 -> 12:16:12:22 (2:00:00:00)
  Job '13186'(x1)  -11:31:10 -> 1:12:28:50 (2:00:00:00)
Jobs:  13186
```

Example 4-5: checknode ALL

```
> checknode ALL
node ahe

State:      Idle  (in current state for 00:00:30)
Configured Resources: PROCS: 12  MEM: 8004M  SWAP: 26G  DISK: 1M
Utilized   Resources: PROCS: 1  SWAP: 4106M
```

```
Dedicated  Resources: ---
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   1.400
Flags:      rmdetected
Classes:    [batch]
RM[ahe]*:   TYPE=PBS
EffNodeAccessPolicy: SHARED

Total Time: 00:01:44  Up: 00:01:44 (100.00%)  Active: 00:00:00 (0.00%)

Reservations:  ---
node ahe-ubuntu32

State:   Running  (in current state for 00:00:05)
Configured Resources: PROCS: 12  MEM: 2013M  SWAP: 3405M  DISK: 1M
Utilized   Resources: PROCS: 6  SWAP: 55M
Dedicated  Resources: PROCS: 6
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   2.000
Flags:      rmdetected
Classes:    [batch]
RM[ahe]*:   TYPE=PBS
EffNodeAccessPolicy: SHARED

Total Time: 00:01:44  Up: 00:01:44 (100.00%)  Active: 00:00:02 (1.92%)

Reservations:
  6x2  Job:Running  -00:00:07 -> 00:01:53 (00:02:00)
  7x2  Job:Running  -00:00:06 -> 00:01:54 (00:02:00)
  8x2  Job:Running  -00:00:05 -> 00:01:55 (00:02:00)
Jobs:        6,7,8
node ahe-ubuntu64

State:      Busy  (in current state for 00:00:06)
Configured Resources: PROCS: 12  MEM: 2008M  SWAP: 3317M  DISK: 1M
Utilized   Resources: PROCS: 12  SWAP: 359M
Dedicated  Resources: PROCS: 12
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   0.000
Flags:      rmdetected
Classes:    [batch]
RM[ahe]*:   TYPE=PBS
EffNodeAccessPolicy: SHARED

Total Time: 00:01:44  Up: 00:01:44 (100.00%)  Active: 00:00:55 (52.88%)

Reservations:
  0x2  Job:Running  -00:01:10 -> 00:00:50 (00:02:00)
  1x2  Job:Running  -00:00:20 -> 00:01:40 (00:02:00)
  2x2  Job:Running  -00:00:20 -> 00:01:40 (00:02:00)
  3x2  Job:Running  -00:00:17 -> 00:01:43 (00:02:00)
  4x2  Job:Running  -00:00:13 -> 00:01:47 (00:02:00)
  5x2  Job:Running  -00:00:07 -> 00:01:53 (00:02:00)
Jobs:        0,1,2,3,4,5
ALERT:  node is in state Busy but load is low (0.000)
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mdiag -n
- showstate

mcredctl

Synopsis

mcredctl [-d *credtype*[:*credid*]] [-h *credtype*:*credid*] [-l *credtype*] [-q {role|limit|profile|accessfrom|accessto|policies} *credtype*[:*credid*]] [--format=xml] [-r {stats|credits|fairshare} *credtype*[:*credid*]] [-t *<STARTTIME>*[,*<ENDTIME>*]

Overview

The mcredctl command controls various aspects about the credential objects within Moab. It can be used to display configuration, limits, roles, and relationships for various moab credential objects.

Arguments

In all cases *<CREDTYPE>* is one of `acct`, `group`, `user`, `class`, or `qos`.

In most cases it is necessary to use the `--format=xml` flag in order to print the output (see examples below for specific syntax requirements).

-d - DESTROY	
Format	*<TYPE>*:*<VAL>*
Description	Purge a credential from `moab.cfg` (does not delete credential from memory).
Example	`> mcredctl -d user:john` *All references to USERCFG[john] will be commented out of `moab.cfg`)*

-h - HOLD	
Format	*<TYPE>*:*<VAL>*
Description	Toggles whether a given credentials' jobs should be place on hold or not.

-h - HOLD	
Example	`> mcredctl -h user:john` *User [john] will be put on hold.*

-l - LIST	
Format	*<TYPE>*
Description	List the various sub-objects of the specified credential.

Example	`> mcredctl -l user --format=xml`
	List all users within Moab in XML.
	`> mcredctl -l group --format=xml`
	List all groups within Moab in XML.

-q - QUERY	
Format	{role | accessfrom | accessto | limit| profile | policies} limit *<TYPE>* policies *<TYPE>* role *<USER>*:*<USERID>* profile *<TYPE>*[:*<VAL>*] accessfrom *<TYPE>*[:*<VAL>*] accessto *<TYPE>*[:*<VAL>*]
Description:	Display various aspects of a credential (formatted in XML)

-q - QUERY	

Example:	`> mcredctl -q role user:bob --format=xml`
	View user bob's administrative role within Moab in XML
	`> mcredctl -q limit acct --format=xml`
	Display limits for all accounts in XML
	`> mcredctl -q policies user:bob`
	View limits organized by credential for user bob on each partition and resource manager

-r - RESET	
Format	*<TYPE>*
Description	Resets the credential within Moab.
Example	`> mcredctl -r user:john` *Resets the credential of user john*

-t - TIMEFRAME	
Format	*<STARTTIME>*[,*<ENDTIME>*]
Description	Can be used in conjunction with the `-q` profile option to display profiling information for the specified timeframe.
Example	`> mcredctl -q profile user -t 14:30_06/20`

Credential Statistics XML Output

Credential statistics can be requested as XML (via the `--format=xml` argument) and will be written to STDOUT in the following format:

```
> mcredctl -q profile user --format=xml -o time:1182927600,1183013999
<Data>
```

```
   <user ...>
     <Profile ...>
     </Profile>
   </user>
 </Data>
```

Example 4-6: Deleting a group

```
> mcredctl -d group:john
GROUPCFG[john] Successfully purged from config files
```

Example 4-7: List users in XML format

```
> mcredctl -l user --format=xml
<Data><user ID="john"</user><user ID="john"></user><user ID="root"></user><user
ID="dev"></user></Data>
```

Example 4-8: Display information about a user

```
> mcredctl -q role user:john --format=xml
<Data><user ID="test" role="admin5"></user></Data>
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes

mdiag

Synopsis

mdiag -a [*accountid*]

mdiag -b [-l *policylevel*] [-t *partition*]

mdiag -c [*classid*]

mdiag -C [*configfile*] // diagnose config file syntax

mdiag -e [-w *<starttime>*|*<endtime>*|*<eventtypes>*|*<oidlist>*|*<eidlist>*|*<objectlist>*] --xml

mdiag -f [-o user|group|acct|qos|class] [-v]

mdiag -g [*groupid*]

mdiag -G [*Green*]

mdiag -j [*jobid*] [-t *<partition>*] [-v] [--blocking]

mdiag -L [-v] // diagnose usage limits

mdiag -n [-A *<creds>*] [-t *partition*] [*nodeid*] [-v]

mdiag -p [-t *partition*] [-v] // diagnose job priority

mdiag -q [*qosid*]

mdiag -r [*reservationid*] [-v] [-w type=*<type>*] [--blocking]

mdiag -R [*resourcemanagername*] [-v]

mdiag -s [*standingreservationid*] [--blocking] mdiag -S [-v] // diagnose scheduler mdiag -t [-v] // diagnose partitions

mdiag -T [*triggerid*] [-v] [--blocking]

mdiag -u [*userid*]

mdiag [--format=xml]

Overview

The mdiag command is used to display information about various aspects of the cluster and the results of internal diagnostic tests. In summary, it provides the following:

- current object health and state information
- current object configuration (resources, policies, attributes, etc.)
- current and historical performance/utilization information
- reports on recent failure
- object messages

Some mdiag options gather information from the Moab cache which prevents them from interrupting the scheduler, but the `--blocking` option can be used to bypass the cache and interrupt the scheduler.

Arguments

Argument	Description
-a [accountid]	Display account information
-b	Display information on jobs blocked by policies, holds, or other factors. If blocked job diagnostics are specified, the `-t` option is also available to constrain the report to analysis of particular partition. Also, with blocked job diagnosis, the `-l` option can be used to specify the analysis policy level.
-c [classid]	Display class information
-C [file]	With the vast array of options in the configuration file, the `-C` option does not validate function, but it does analyze the configuration file for syntax errors including use of invalid parameters, deprecated parameters, and some illegal values. If you start Moab with the -e flag, Moab evaluates the configuration file at startup and quits if an error exists.

Argument	Description
-e	Moab will do a query for all events whose *eventtime* starts at <*starttime*> and matches the search criteria. This works only when Moab is configured with ODBC MySQL. The syntax is: mdiag -e[-w <*starttime*>|<*eventtypes*>| <*oidlist*>|<*eidlist*>|<*objectlist*>] --xml • *starttime* default is - • • *eventtypes* default is command delimited, the default is all event types (possible values can be found in the EventType table in the Moab database) • *oidlist* is a comma-delimited list of object ids, the default is all objects ids • *eidlist* is a comma-delimited list of specific event ids, the default is all event ids • *objectlist* is a comma-delimited list of object types, the default is all object types (possible values can be found in the ObjectType table in the Moab database)
-f	Display fairshare information
-g [groupid]	display group information
-G [Green]	display power management information
-j [jobid]	display job information
-L	display limits
-n [nodeid]	display nodes If node diagnostics are specified, the `-t` option is also available to constrain the report to a particular partition.
-p	display job priority. If priority diagnostics are specified, the `-t` option is also available to constrain the report to a particular partition.
-q [qosid]	display QoS information
-r [reservationid]	display reservation information
-R [rmid]	display resource manager information

Argument	Description
-s [srsv]	display standing reservation information
-S	display general scheduler information
-t	display configuration, usage, health, and diagnostic information about partitions maintained by Moab
-T [triggerid]	display trigger information
-u [userid]	display user information
--format=xml	display output in XML format

XML Output

Information for most of the options can be reported as XML as well. This is done with the command `mdiag -<option> <CLASS_ID> --format=xml`. For example, XML-based class information will be written to STDOUT in the following format:

```
<Data>
  <class <ATTR>="<VAL>" ... >
    <stats <ATTR>="<VAL>" ... >
      <Profile <ATTR>="<VAL>" ... >
      </Profile>
    </stats>
  </class>
<Data>
  ...
</Data>
```

Of the mdiag options, only `-G` and `-L` cannot be reported as XML.

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- checkjob
- checknode

mdiag -a

Synopsis

mdiag -a [*accountid*]

Overview

The mdiag -a command provides detailed information about the accounts (aka projects) Moab is currently tracking. This command also allows an administrator to verify correct throttling policies and access provided to and from other credentials.

Example 4-9: Generating information about accounts

```
> mdiag -a
evaluating acct information
Name         Priority        Flags         QDef      QOSList*
PartitionList Target  Limits
engineering       100            -         high      high,urgent,low             [A]
[B]      30.00  MAXJOB=50,75  MAXPROC=400,500
marketing           1            -          low      low                         [A]
       5.00  MAXJOB=100,110  MAXPS=54000,54500
it                 10            -      DEFAULT      DEFAULT,high,urgent,low     [A]
     100.00  MAXPROC=100,1250 MAXPS=12000,12500
  FSWEIGHT=1000
development       100            -        high       high,urgent,low             [A]
[B]      30.00  MAXJOB=50,75 MAXNODE=100,120
research          100            -        high       DEFAULT,high,low            [A]
[B]      30.00  MAXNODE=400,500 MAXPS=900000,1000000
DEFAULT             0            -           -       -                           -
       0.00  -
```

Related topics

- Account credential

mdiag -b

Synopsis

mdiag -b [-l *policylevel*] [-t *partition*]

Overview

The mdiag -b command returns information about blocked jobs.

mdiag -c

Synopsis

mdiag -c [-v] [*classid*]

Overview

The mdiag -c command provides detailed information about the classes Moab is currently tracking. This command also allows an administrator to verify correct throttling policies and access provided to and from other credentials.

The term class is used interchangeably with the term queue and generally refers to resource manager queue.

XML Attributes

Name	Description
ADEF	Accounts a class has access to.
CAPACITY	Number of procs available to the class.
DEFAULT.ATTR	Default attributes attached to a job.
DEFAULT.DISK	Default required disk attached to a job.
DEFAULT.FEATURES	Default required node features attached to a job.
DEFAULT.GRES	Default generic resources attached to a job.
DEFAULT.MEM	Default required memory attached to a job.
DEFAULT.NODESET	Default specified node set attached to a job.
DEFAULT.WCLIMIT	Default wallclock limit attached to a job.
EXCL.FEATURES	List of excluded (disallowed) node features.
EXCL.FLAGS	List of excluded (disallowed) job flags.
FSTARGET	The class' fairshare target.
HOLD	If TRUE this credential has a hold on it, FALSE otherwise.
HOSTLIST	The list of hosts in this class.
JOBEPILOG	Scheduler level job epilog to be run after job is completed by resource manager (script path).
JOBFLAGS	Default flags attached to jobs in the class.
JOBPROLOG	Scheduler level job prolog to be run before job is started by resource manager (script path).
ID	The unique ID of this class.

Name	Description
LOGLEVEL	The log level attached to jobs in the class.
MAX.PROC	The max processors per job in the class.
MAX.PS	The max processor-seconds per job in the class.
MAX.WCLIMIT	The max wallclock limit per job in the class.
MAXIJOB	The max idle jobs in the class.
MAXIPROC	The max idle processors in the class.
MAXJOBPERUSER	The max jobs per user.
MAXNODEPERJOB	The max nodes per job.
MAXNODEPERUSER	The max nodes per user.
MAXPROCPERJOB	The max processors per job.
MAXPROCPERNODE	The max processors per node.
MAXPROCPERUSER	The max processors per user.
MIN.NODE	The minimum nodes per job in the class.
MIN.PROC	The minimum processors per job in the class.
MIN.WCLIMIT	The minimum wallclock limit per job in the class.
NODEACCESSPOLICY	The node access policy associated with jobs in the class.
OCDPROCFACTOR	Dedicated processor factor.
OCNODE	Overcommit node.
PRIORITY	The class' associated priority.
PRIORITYF	Priority calculation function.

Name	Description
REQ.FEATURES	Required features for a job to be considered in the class.
REQ.FLAGS	Required flags for a job to be considered in the class.
REQ.IMAGE	Required image for a job to be considered in the class.
REQUIREDUSERLIST	The list of users who have access to the class.
RM	The resource manager reporting the class.
STATE	The class' state.
WCOVERRUN	Tolerated amount of time beyond the specified wallclock limit.

Example 4-10: Generating information about classes

```
> mdiag -c
Class/Queue Status
ClassID        Priority Flags        QDef              QOSList* PartitionList
Target Limits
DEFAULT               0 ---          ---                   ---  ---
0.00  ---
batch                 1 ---          ---                   ---  [A][B]
70.00  MAXJOB=33:200,250
  MAX.WCLIMIT=10:00:00  MAXPROCPERJOB=128
long                  1 ---          low                   low  [A]
10.00  MAXJOB=3:100,200
  MAX.WCLIMIT=1:00:00:00  MAXPROCPERJOB=128
fast                100 ---          high                 high  [B]
10.00  MAXJOB=8:100,150
  MAX.WCLIMIT=00:30:00  MAXPROCPERJOB=128
bigmem                1 ---          low,high              low  ---
10.00  MAXJOB=1:100,200
  MAXPROCPERJOB=128
```

*In the example above, class fast has **MAXJOB** soft and hard limits of 100 and 150 respectively and is currently running 8 jobs.*

The `Limits` column will display limits in the following format:
<USAGE>:<HARDLIMIT>[,<SOFTLIMIT>]

Related topics

- showstats command - display general statistics

mdiag -f

Synopsis

mdiag -f [-o user|group|acct|qos|class] [--flags=relative] [-w par=*<PARTITIONID>*]

Overview

The mdiag -f command is used to display at a glance information about the fairshare configuration and historic resource utilization. The fairshare usage may impact job prioritization, job eligibility, or both based on the credential **FSTARGET** and **FSCAP** attributes and by the fairshare priority weights as described in the Job Prioritization Overview. The information presented by this command includes fairshare configuration and credential fairshare usage over time.

The command hides information about credentials which have no fairshare target and no fairshare cap.

If an object type (*<OTYPE>*) is specified, then only information for that credential type (user, group, acct, class, or qos) will be displayed. If the `relative` flag is set, then per user fairshare usage will be displayed relative to each non-user credential (see the second example below).

Relative output is only displayed for credentials which have user mappings. For example, if there is no association between classes and users, no relative per user fairshare usage class breakdown will be provided.

Example 4-11: Standard Fairshare Output

```
> mdiag -f
FairShare Information
Depth: 6 intervals   Interval Length: 00:20:00   Decay Rate: 0.50
FS Policy: DEDICATEDPES
System FS Settings:  Target Usage: 0.00
FSInterval        %     Target        0       1       2       3       4       5
FSWeight       ------- -------  1.0000  0.5000  0.2500  0.1250  0.0625  0.0312
TotalUsage      100.00 -------    85.3   476.1   478.9   478.5   475.5   482.8
USER
-------------
mattp             2.51 -------    2.20    2.69    2.21    2.65    2.65    3.01
jsmith           12.82 -------   12.66   15.36   10.96    8.74    8.15   13.85
kyliem            3.44 -------    3.93    2.78    4.36    3.11    3.94    4.25
tgh               4.94 -------    4.44    5.12    5.52    3.95    4.66    4.76
walex             1.51 -------    3.14    1.15    1.05    1.61    1.22    1.60
jimf              4.73 -------    4.67    4.31    5.67    4.49    4.93    4.92
poy               4.64 -------    4.43    4.61    4.58    4.76    5.36    4.90
mjackson          0.66 -------    0.35    0.78    0.67    0.77    0.55    0.43
tfw              17.44 -------   16.45   15.59   19.93   19.72   21.38   15.68
gjohn             2.81 -------    1.66    3.00    3.16    3.06    2.41    3.33
ljill            10.85 -------   18.09    7.23   13.28    9.24   14.76    6.67
kbill            11.10 -------    7.31   14.94    4.70   15.49    5.42   16.61
stevei            1.58 -------    1.41    1.34    2.09    0.75    3.30    2.15
gms               1.54 -------    1.15    1.74    1.63    1.40    1.38    0.90
patw              5.11 -------    5.22    5.11    4.85    5.20    5.28    5.78
wer               6.65 -------    5.04    7.03    7.52    6.80    6.43    2.83
anna              1.97 -------    2.29    1.68    2.27    1.80    2.37    2.17
susieb            5.69 -------    5.58    5.55    5.57    6.48    5.83    6.16
GROUP
-------------
dallas           13.25  15.00    14.61   12.41   13.19   13.29   15.37   15.09
sanjose*          8.86  15.00     6.54    9.55    9.81    8.97    8.35    4.16
```

```
seattle          10.05  15.00     9.66   10.23   10.37    9.15    9.94   10.54
austin*          30.26  15.00    29.10   30.95   30.89   28.45   29.53   29.54
boston*           3.44  15.00     3.93    2.78    4.36    3.11    3.94    4.25
orlando*         26.59  15.00    29.83   26.77   22.56   29.49   25.53   28.18
newyork*          7.54  15.00     6.33    7.31    8.83    7.54    7.34    8.24
ACCT
-------------
engineering      31.76  30.00    32.25   32.10   31.94   30.07   30.74   31.14
marketing         8.86   5.00     6.54    9.55    9.81    8.97    8.35    4.16
it                9.12   5.00     7.74    8.65   10.92    8.29   10.64   10.40
development*     24.86  30.00    24.15   24.76   25.00   24.84   26.15   26.78
research         25.40  30.00    29.32   24.94   22.33   27.84   24.11   27.53
QOS
-------------
DEFAULT*          0.00  50.00  ------- ------- ------- ------- ------- -------
high*            83.69  90.00    86.76   83.20   81.71   84.35   83.19   88.02
urgent            0.00   5.00  ------- ------- ------- ------- ------- -------
low*             12.00   5.00     7.34   12.70   14.02   12.51   12.86    7.48
CLASS
-------------
batch*           51.69  70.00    53.87   52.01   50.80   50.38   48.67   52.65
long*            18.75  10.00    16.54   18.36   20.89   18.36   21.53   16.28
fast*            15.29  10.00    18.41   14.98   12.58   16.80   15.15   18.21
bigmem           14.27  10.00    11.17   14.65   15.73   14.46   14.65   12.87
```

An asterisk (*) next to a credential name indicates that that credential has exceeded its fairshare target.

Example 4-12: Grouping User Output by Account

```
> mdiag -f -o acct --flags=relative
FairShare Information
Depth: 6 intervals   Interval Length: 00:20:00   Decay Rate: 0.50
FS Policy: DEDICATEDPES
System FS Settings:  Target Usage: 0.00
FSInterval        %     Target       0       1       2       3       4       5
FSWeight       ------- -------  1.0000  0.5000  0.2500  0.1250  0.0625  0.0312
TotalUsage      100.00 -------    23.8   476.1   478.9   478.5   475.5   482.8
ACCOUNT
-------------
dallas           13.12  15.00    15.42   12.41   13.19   13.29   15.37   15.09
  mattp          19.47 -------   15.00   21.66   16.75   19.93   17.26   19.95
  walex           9.93 -------   20.91    9.28    7.97   12.14    7.91   10.59
  stevei         12.19 -------    9.09   10.78   15.85    5.64   21.46   14.28
  anna           14.77 -------   16.36   13.54   17.18   13.55   15.44   14.37
  susieb         43.64 -------   38.64   44.74   42.25   48.74   37.92   40.81
sanjose*          9.26  15.00     8.69    9.55    9.81    8.97    8.35    4.16
  mjackson        7.71 -------    6.45    8.14    6.81    8.62    6.54   10.29
  gms            17.61 -------   21.77   18.25   16.57   15.58   16.51   21.74
  wer            74.68 -------   71.77   73.61   76.62   75.80   76.95   67.97
seattle          10.12  15.00    10.16   10.23   10.37    9.15    9.94   10.54
  tgh            49.56 -------   46.21   50.05   53.26   43.14   46.91   45.13
  patw           50.44 -------   53.79   49.95   46.74   56.86   53.09   54.87
austin*          30.23  15.00    25.58   30.95   30.89   28.45   29.53   29.54
  jsmith         42.44 -------   48.77   49.62   35.47   30.70   27.59   46.90
  tfw            57.56 -------   51.23   50.38   64.53   69.30   72.41   53.10
boston*           3.38  15.00     3.78    2.78    4.36    3.11    3.94    4.25
  kyliem         100.00 -------  100.00  100.00  100.00  100.00  100.00  100.00
orlando*         26.20  15.00    30.13   26.77   22.56   29.49   25.53   28.18
  poy            17.90 -------   16.28   17.22   20.30   16.15   20.98   17.39
```

```
   ljill          37.85 -------   58.60   26.99   58.87   31.33   57.79   23.67
   kbill          44.25 -------   25.12   55.79   20.83   52.52   21.23   58.94
 newyork*          7.69  15.00     6.24    7.31    8.83    7.54    7.34    8.24
   jimf           61.42 -------   69.66   58.94   64.20   59.46   67.21   59.64
   gjohn          38.58 -------   30.34   41.06   35.80   40.54   32.79   40.36
```

Related topics

- Fairshare Overview

mdiag -g

Synopsis

mdiag -g [*groupid*]

Overview

The mdiag -g command is used to present information about groups.

mdiag -j

Synopsis

mdiag -j [*jobid*] [-t <*partition*>] [-v] [-w] [--flags=policy] [--xml] [--blocking]

Overview

The mdiag -j command provides detailed information about the state of jobs Moab is currently tracking. This command also performs a large number of sanity and state checks. The job configuration and status information, as well as the results of the various checks, are presented by this command. The command gathers information from the Moab cache which prevents it from interrupting the scheduler, but the `--blocking` option can be used to bypass the cache and interrupt the scheduler. If the `-v` (verbose) flag is specified, additional information about less common job attributes is displayed. If `--flags=policy` is specified, information about job templates is displayed.

If used with the `-t <partition>` option on a running job, the only thing mdiag -j shows is if the job is running on the specified partition. If used on job that is not running, it shows if the job is able to run on the specified partition.

The `-w` flag enables you to specify specific job states (Such as Running, Completed, Idle, or ALL. See **Job States** on page 28 for all valid options.) or jobs associated with a given credential (user, acct, class, group, qos). For example:

```
mdiag -j -w user=david            # Displays only David's jobs
mdiag -j -w state=Idle,Running    # Displays only idle or running jobs
```

The mdiag -j command does not show all subjobs of an array unless you use `mdiag -j --xml`. In the XML, the master job element contains a child element called `ArraySubJobs` that contains the subjobs in the array. Using `mdiag -j -v --xml` shows the completed sub-jobs as well.

XML Output

If XML output is requested (via the --format=xml argument), XML based node information will be written to STDOUT in the following format:

```
<Data>
  <job ATTR="VALUE" ... > </job>
  ...
</Data>
```

For information about legal attributes, refer to the XML Attributes table.

To show jobs in XML, use `mdiag -j --xml -w [completed=true|system=true|ALL=true]` to limit or filter jobs. This is for XML use only.

Related topics

- checkjob
- mdiag

mdiag -n

Synopsis

mdiag -n [-t *partitionid*] [-A *creds*] [-w <CONSTRAINT>] [-v] [--format=xml] [*nodeid*]

Overview

The mdiag -n command provides detailed information about the state of nodes Moab is currently tracking. This command also performs a large number of sanity and state checks. The node configuration and status information as well as the results of the various checks are presented by this command.

Arguments

Flag	Argument	Description
[-A]	{user|group|account|qos|class|job}: <OBJECTID>	report if each node is accessible by requested job or credential
[-t]	<partitionid>	report only nodes from specified partition
[-v]	---	show verbose output (do not truncate columns and add columns for additional node attributes)
[-w]	nodestate=drained	display only jobs associated with the specified constraint: nodestate (See DISPLAYFLAGS for more information.)

Output

This command presents detailed node information in whitespace-delineated fields.

The output of this command can be extensive and the values for a number of fields may be truncated. If truncated, the -v flag can be used to display full field content.

Column	Format
Name	*<NODE NAME>*
State	*<NODE STATE>*
Procs	*<AVAILABLE PROCS>:<CONFIGURED PROCS>*
Memory	*<AVAILABLE MEMORY>:<CONFIGURED MEMORY>*
Disk	*<AVAILABLE DISK>:<CONFIGURED DISK>*
Swap	*<AVAILABLE SWAP>:<CONFIGURED SWAP>*
Speed	*<RELATIVE MACHINE SPEED>*
Opsys	*<NODE OPERATING SYSTEM>*
Arch	*<NODE HARDWARE ARCHITECTURE>*
Par	*<PARTITION NODE IS ASSIGNED TO>*
Load	*<CURRENT 1 MINUTE BSD LOAD>*
Rsv	*<NUMBER OF RESERVATIONS ON NODE>*
Classes	*<CLASS NAME>*
Network	*<NETWORK NAME>...*
Features	*<NODE FEATURE>...*

Examples

Example 4-13:

```
> mdiag -n

compute node summary
Name                    State   Procs      Memory         Opsys

opt-001                  Busy    0:2      2048:2048        SuSE
```

```
opt-002                  Busy    0:2      2048:2048        SuSE
opt-003                  Busy    0:2      2048:2048        SuSE
opt-004                  Busy    0:2      2048:2048        SuSE
opt-005                  Busy    0:2      2048:2048        SuSE
opt-006                  Busy    0:2      2048:2048        SuSE
WARNING:   swap is low on node opt-006
opt-007                  Busy    0:2      2048:2048        SuSE
opt-008                  Busy    0:2      2048:2048        SuSE
opt-009                  Busy    0:2      2048:2048        SuSE
opt-010                  Busy    0:2      2048:2048        SuSE
opt-011                  Busy    0:2      2048:2048        SuSE
opt-012                  Busy    0:2      2048:2048        SuSE
opt-013                  Busy    0:2      2048:2048        SuSE
opt-014                  Busy    0:2      2048:2048        SuSE
opt-015                  Busy    0:2      2048:2048        SuSE
opt-016                  Busy    0:2      2048:2048        SuSE
x86-001                  Busy    0:1       512:512       Redhat
x86-002                  Busy    0:1       512:512       Redhat
x86-003                  Busy    0:1       512:512       Redhat
x86-004                  Busy    0:1       512:512       Redhat
x86-005                  Idle    1:1       512:512       Redhat
x86-006                  Idle    1:1       512:512       Redhat
x86-007                  Idle    1:1       512:512       Redhat
x86-008                  Busy    0:1       512:512       Redhat
x86-009                  Down    1:1       512:512       Redhat
x86-010                  Busy    0:1       512:512       Redhat
x86-011                  Busy    0:1       512:512       Redhat
x86-012                  Busy    0:1       512:512       Redhat
x86-013                  Busy    0:1       512:512       Redhat
x86-014                  Busy    0:1       512:512       Redhat
x86-015                  Busy    0:1       512:512       Redhat
x86-016                  Busy    0:1       512:512       Redhat
P690-001                 Busy    0:1     16384:16384        AIX
P690-002                 Busy    0:1     16384:16384        AIX
P690-003                 Busy    0:1     16384:16384        AIX
P690-004                 Busy    0:1     16384:16384        AIX
P690-005                 Busy    0:1     16384:16384        AIX
P690-006                 Busy    0:1     16384:16384        AIX
P690-007                 Idle    1:1     16384:16384        AIX
P690-008                 Idle    1:1     16384:16384        AIX
WARNING:   node P690-008 is missing ethernet adapter
P690-009                 Busy    0:1     16384:16384        AIX
P690-010                 Busy    0:1     16384:16384        AIX
P690-011                 Busy    0:1     16384:16384        AIX
P690-012                 Busy    0:1     16384:16384        AIX
P690-013                 Busy    0:1     16384:16384        AIX
P690-014                 Busy    0:1     16384:16384        AIX
P690-015                 Busy    0:1     16384:16384        AIX
P690-016                 Busy    0:1     16384:16384        AIX
-----                     ---    6:64   745472:745472     -----

Total Nodes: 36  (Active: 30  Idle: 5  Down: 1)
```

Warning messages are interspersed with the node configuration information with all warnings preceded by the keyword WARNING.

XML Output

If XML output is requested (via the --format=xml argument), XML based node information will be written to STDOUT in the following format:

```
mdiag -n --format=xml
<Data>
  <node> <ATTR>="<VAL>" ... </node>
  ...
</Data>
```

XML Attributes

Name	Description
AGRES	Available generic resources
ALLOCRES	Special allocated resources (like vlans)
ARCH	The node's processor architecture.
AVLCLASS	Classes available on the node.
AVLETIME	Time when the node will no longer be available (used in Utility centers)
AVLSTIME	Time when the node will be available (used in Utility centers)
CFGCLASS	Classes configured on the node
ENABLEPROFILING	If true, a node's state and usage is tracked over time.
FEATURES	A list of comma-separated custom features describing a node.
GEVENT	A user-defined event that allows Moab to perform some action.
GMETRIC	A list of comma-separated consumable resources associated with a node.
GRES	generic resources on the node
ISDELETED	Node has been deleted
ISDYNAMIC	Node is dynamic (used in Utility centers)
JOBLIST	The list of jobs currently running on a node.
LOAD	Current load as reported by the resource manager
LOADWEIGHT	Load weight used when calculating node priority

Name	Description
MAXJOB	See Node Policies for details.
MAXJOBPERUSER	See Node Policies for details.
MAXLOAD	See Node Policies for details.
MAXPROC	See Node Policies for details.
MAXPROCPERUSER	See Node Policies for details.
NETWORK	The ability to specify which networks are available to a given node is limited to only a few resource managers. Using the **NETWORK** attribute, administrators can establish this node to network connection directly through the scheduler. The NODECFG parameter allows this list to be specified in a comma-delimited list.
NODEID	The unique identifier for a node.
NODESTATE	The state of a node.
OS	A node's operating system.
OSLIST	Operating systems the node can run
OSMODACTION	URL for changing the operating system
OWNER	Credential type and name of owner
PARTITION	The partition a node belongs to. See Node Location for details.
POWER	The state of the node's power. Either ON or OFF.
PRIORITY	The fixed node priority relative to other nodes.
PROCSPEED	A node's processor speed information specified in MHz.
RACK	The rack associated with a node's physical location.
RADISK	The total available disk on a node.

Name	Description
RAMEM	The total available memory available on a node.
RAPROC	The total number of processors available on a node.
RASWAP	The total available swap on a node.
RCMEM	The total configured memory on a node.
RCPROC	The total configured processors on a node.
RCSWAP	The total configured swap on a node.
RESCOUNT	Number of reservations on the node
RSVLIST	List of reservations on the node
RESOURCES	Deprecated (use GRES)
RMACCESSLIST	A comma-separated list of resource managers who have access to a node.
SIZE	The number of slots or size units consumed by the node.
SLOT	The first slot in the rack associated with the node's physical location.
SPEED	A node's relative speed.
SPEEDWEIGHT	speed weight used to calculate node's priority
STATACTIVETIME	Time node was active
STATMODIFYTIME	Time node's state was modified
STATTOTALTIME	Time node has been monitored
STATUPTIME	Time node has been up
TASKCOUNT	The number of tasks on a node.

Related topics

- checknode

mdiag -t

Synopsis

mdiag -t [-v] [-v] [*partitionid*]

Overview

The mdiag -t command is used to present configuration, usage, health, and diagnostic information about partitions maintained by Moab. The information presented includes partition name, limits, configured and available resources, allocation weights and policies.

Examples

Example 4-14: Standard partition diagnostics

```
> mdiag -t
Partition Status
...
```

mdiag -p

Synopsis

mdiag -p [-t *partition*] [-v]

Overview

The mdiag -p command is used to display at a glance information about the job priority configuration and its effects on the current eligible jobs. The information presented by this command includes priority weights, priority components, and the percentage contribution of each component to the total job priority.

The command hides information about priority components which have been deactivated (i.e. by setting the corresponding component priority weight to 0). For each displayed priority component, this command gives a small amount of context sensitive information. The following table documents this information. In all cases, the output is of the form *<PERCENT>*(*<CONTEXT INFO>*) where *<PERCENT>* is the percentage contribution of the associated priority component to the job's total priority.

By default, this command only shows information for jobs which are eligible for immediate execution. Jobs which violate soft or hard policies, or have holds, job dependencies, or other job constraints in place will not be displayed. If priority information is needed for any of these jobs, use the -v flag or the checkjob command.

Format

Flag	Name	Format	Default	Description	Example
-v	VERBOSE	---	---	Display verbose priority information. If specified, display priority breakdown information for blocked, eligible, and active jobs. By default, only information for eligible jobs is displayed. To view blocked jobs in addition to eligible, run mdiag -p -v -v.	`> mdiag -p -v` *Display priority summary information for eligible and active jobs*

Output

Priority Component	Format	Description
Target	*<PERCENT>*()	
QoS	*<PERCENT>*(*<QOS>*:*<QOSPRI>*)	*QOS* — QoS associated with job *QOSPRI* — Priority assigned to the QoS
FairShare	*<PERCENT>* (*<USR>* :*<GRP>*:*<ACC>*:*<QOS>*:*<CLS>*)	*USR* — user fs usage - user fs target *GRP* — group fs usage - group fs target *ACC* — account fs usage - account fs target *QOS* — QoS fs usage - QoS fs target *CLS* — class fs usage - class fs target
Service	*<PERCENT>*(*<QT>*:*<XF>*:*<Byp>*)	*QTime* — job queue time which is applicable towards priority (in minutes) *XF* — current theoretical minimum XFactor is job were to start immediately *Byp* — number of times job was bypassed by lower priority jobs via backfill

Priority Component	Format	Description
Resource	*<PERCENT>* (*<NDE>*:*<PE>*:*<PRC>*:*<MEM>*)	*NDE* — nodes requested by job *PE* — Processor Equivalents as calculated by all resources requested by job *PRC* — processors requested by job *MEM* — real memory requested by job

Examples

Example 4-15: mdiag -p

```
diagnosing job priority information (partition: ALL)

Job                     PRIORITY*     Cred(QOS)     FS(Accnt)   Serv(QTime)
Weights    --------      1(    1)      1(    1)      1(    1)

13678                      1321*    7.6(100.0)    0.2(  2.7)   92.2(1218.)
13698                       235*   42.6(100.0)    1.1(  2.7)   56.3(132.3)
13019                      8699     0.6( 50.0)    0.3( 25.4)   99.1(8674.)
13030                      8699     0.6( 50.0)    0.3( 25.4)   99.1(8674.)
13099                      8537     0.6( 50.0)    0.3( 25.4)   99.1(8512.)
13141                      8438     0.6( 50.0)    0.2( 17.6)   99.2(8370.)
13146                      8428     0.6( 50.0)    0.2( 17.6)   99.2(8360.)
13153                      8360     0.0(  1.0)    0.1( 11.6)   99.8(8347.)
13177                      8216     0.0(  1.0)    0.1( 11.6)   99.8(8203.)
13203                      8127     0.6( 50.0)    0.3( 25.4)   99.1(8102.)
13211                      8098     0.0(  1.0)    0.1( 11.6)   99.8(8085.)
...
13703                       137    36.6( 50.0)   12.8( 17.6)   50.6( 69.2)
13702                        79     1.3(  1.0)    5.7(  4.5)   93.0( 73.4)

Percent Contribution    --------     0.9(  0.9)    0.4(  0.4)   98.7( 98.7)

* indicates system prio set on job
```

The `mdiag -p` command only displays information for priority components actually utilized. In the above example, QOS, Account Fairshare, and QueueTime components are utilized in determining a job's priority. Other components, such as Service Targets, and Bypass are not used and thus are not displayed. (See the Priority Overview for more information) The output consists of a header, a job by job analysis of jobs, and a summary section.

*The header provides column labeling and provides configured priority component and subcomponent weights. In the above example, **QOSWEIGHT** is set to `1000` and **FSWEIGHT** is set to `100`. When configuring fairshare, a site also has the option of weighting the individual components of a job's overall fairshare, including its user, group, and account fairshare components. In this output, the 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 `qtime` which is contributing `99.1%` of the total priority since the job has been in the queue for approximately 8600 minutes.

At the end of the job by job description, a Totals line is displayed which documents the average percentage contributions of each priority component to the current idle jobs. In this example, the QOS, Fairshare, and Service components contributed an average of `0.9%`, `0.4%`, and `98.7%` to the jobs' total priorities.

Related topics

- Job Priority Overview
- Moab Cluster Manager - Priority Manager

mdiag -q

Synopsis

mdiag -q [*qosid*]

Overview

The mdiag -q command is used to present information about each QoS maintained by Moab. The information presented includes QoS name, membership, scheduling priority, weights and flags.

Examples

Example 4-16: Standard QoS Diagnostics

```
> mdiag -q
QOS Status
System QOS Settings:  QList: DEFAULT (Def: DEFAULT)  Flags: 0
Name                * Priority QTWeight QTTarget XFWeight XFTarget     QFlags
JobFlags Limits
DEFAULT                      1        1        3        1     5.00  PREEMPTEE
[NONE] [NONE]
  Accounts:  it research
  Classes:  batch
[ALL]                        0        0        0        0     0.00     [NONE]
[NONE] [NONE]
high                      1000        1        2        1    10.00  PREEMPTOR
[NONE] [NONE]
  Accounts:  engineering it development research
  Classes:  fast
urgent                   10000        1        1        1     7.00  PREEMPTOR
[NONE] [NONE]
  Accounts:  engineering it development
low                        100        1        5        1     1.00  PREEMPTEE
[NONE] [NONE]
  Accounts:  engineering marketing it development research
  Classes:  long bigmem
```

mdiag -r

Synopsis

mdiag -r [*reservationid*] [-v] [-w type=<*type*>]

Overview

The mdiag -r command allows administrators to look at detailed reservation information. It provides the name, type, partition, starttime and endtime, proc and node counts, as well as actual utilization figures. It also provides detailed information about which resources are being used, how many nodes, how much memory, swap, and processors are being associated with each task. Administrators can also view the Access Control Lists for each reservation as well as any flags that may be active in the reservation. The

command gathers information from the Moab cache which prevents it from waiting for the scheduler, but the `--blocking` option can be used to bypass the cache and allow waiting for the scheduler.

The `-w` flag filters the output according to the type of reservation. The allowable reservation types are Job, and User.

Examples

Example 4-17:

```
> mdiag -r
Diagnosing Reservations
RsvID                     Type Par   StartTime     EndTime     Duration Node Task
Proc
-----                     ---- ---   ---------     -------     -------- ---- ---- ---
-
engineer.0.1              User   A    -6:29:00    INFINITY     INFINITY    0    0
7
    Flags: STANDINGRSV IGNSTATE OWNERPREEMPT
    ACL:   CLASS==batch+:==long+:==fast+:==bigmem+ QOS==low-:==high+ JATTR==PREEMPTEE+
    CL:    RSV==engineer.0.1
    Task Resources: PROCS: [ALL]
    Attributes (HostExp='fr10n01 fr10n03 fr10n05 fr10n07 fr10n09 fr10n11 fr10n13
fr10n15')
    Active PH: 43.77/45.44 (96.31%)
    SRAttributes (TaskCount: 0  StartTime: 00:00:00  EndTime: 1:00:00:00  Days: ALL)
research.0.2              User   A    -6:29:00    INFINITY     INFINITY    0    0
8
    Flags: STANDINGRSV IGNSTATE OWNERPREEMPT
    ACL:   CLASS==batch+:==long+:==fast+:==bigmem+ QOS==high+:==low- JATTR==PREEMPTEE+
    CL:    RSV==research.0.2
    Task Resources: PROCS: [ALL]
    Attributes (HostExp='fr3n01 fr3n03 fr3n05 fr3n07 fr3n07 fr3n09 fr3n11 fr3n13
fr3n15')
    Active PH: 51.60/51.93 (99.36%)
    SRAttributes (TaskCount: 0  StartTime: 00:00:00  EndTime: 1:00:00:00  Days: ALL)
fast.0.3                  User   A    00:14:05     5:14:05      5:00:00    0    0
16
    Flags: STANDINGRSV IGNSTATE OWNERPREEMPT
    ACL:   CLASS==fast+ QOS==high+:==low+:==urgent+:==DEFAULT+ JATTR==PREEMPTEE+
    CL:    RSV==fast.0.3
    Task Resources: PROCS: [ALL]
    Attributes (HostExp='fr12n01 fr12n02 fr12n03 fr12n04 fr12n05 fr12n06 fr12n07
fr12n08 fr12n09 fr12n10 fr12n11 fr12n12 fr12n13 fr12n14 fr12n15 fr12n16')
    SRAttributes (TaskCount: 0  StartTime: 00:00:00  EndTime: 5:00:00  Days:
Mon,Tue,Wed,Thu,Fri)
fast.1.4                  User   A  1:00:14:05  1:05:14:05      5:00:00    0    0
16
    Flags: STANDINGRSV IGNSTATE OWNERPREEMPT
    ACL:   CLASS==fast+ QOS==high+:==low+:==urgent+:==DEFAULT+ JATTR==PREEMPTEE+
    CL:    RSV==fast.1.4
    Task Resources: PROCS: [ALL]
    Attributes (HostExp='fr12n01 fr12n02 fr12n03 fr12n04 fr12n05 fr12n06 fr12n07
fr12n08 fr12n09 fr12n10 fr12n11 fr12n12 fr12n13 fr12n14 fr12n15 fr12n16')
    SRAttributes (TaskCount: 0  StartTime: 00:00:00  EndTime: 5:00:00  Days:
Mon,Tue,Wed,Thu,Fri)
job2411                    Job   A   -00:01:00    00:06:30     Each tile contains a
summary information about the service it represents, including the following:
    ACL:   JOB==job2411=
    CL:    JOB==job2411 USER==jimf GROUP==newyork ACCT==it CLASS==bigmem QOS==low
JATTR==PREEMPTEE DURATION==00:07:30 PROC==6 PS==2700
```

```
job1292                     Job   A    00:00:00    00:07:30     00:07:30    0    0
4
    ACL:   JOB==job1292=
    CL:    JOB==job1292 USER==jimf GROUP==newyork ACCT==it CLASS==batch QOS==DEFAULT
JATTR==PREEMPTEE DURATION==00:07:30 PROC==4 PS==1800
```

Example 4-18:

With the -v option, a nodes line is included for each reservation and shows how many nodes are in the reservation as well as how many tasks are on each node.

```
> mdiag -r -v
Diagnosing Reservations
RsvID                      Type Par   StartTime     EndTime     Duration Node Task
Proc
-----                      ---- ---   ---------     -------     -------- ---- ---- ---
-
Moab.6                      Job   B   -00:01:05    00:00:35     00:01:40    1    1
1
    Flags: ISACTIVE
    ACL:   JOB==Moab.6=
    CL:    JOB==Moab.6 USER==tuser1 GROUP==tgroup1 CLASS==fast QOS==starter
JPRIORITY<=0 DURATION==00:01:40 PROC==1 PS==100
    SubType: JobReservation
    Nodes='node002:1'
    Rsv-Group: Moab.6

Moab.4                      Job   B   -00:01:05    00:00:35     00:01:40    1    1
1
    Flags: ISACTIVE
    ACL:   JOB==Moab.4=
    CL:    JOB==Moab.4 USER==tuser1 GROUP==tgroup1 CLASS==batch QOS==starter
JPRIORITY<=0 DURATION==00:01:40 PROC==1 PS==100
    SubType: JobReservation
    Nodes='node002:1'
    Rsv-Group: Moab.4

Moab.5                      Job   A   -00:01:05    00:00:35     00:01:40    3    3
6
    Flags: ISACTIVE
    ACL:   JOB==Moab.5=
    CL:    JOB==Moab.5 USER==tuser1 GROUP==tgroup1 ACCT==marketing CLASS==long
QOS==low JPRIORITY<=0 DURATION==00:01:40 PROC==6 PS==600
    Task Resources: PROCS: [ALL]
    SubType: JobReservation
    Nodes='node008:1,node007:1,node006:1'
    Rsv-Group: Moab.5

Moab.7                      Job   A   -00:01:04    00:00:36     00:01:40    1    1
1
    Flags: ISACTIVE
    ACL:   JOB==Moab.7=
    CL:    JOB==Moab.7 USER==tuser1 GROUP==tgroup1 CLASS==bigmen QOS==starter
JPRIORITY<=0 DURATION==00:01:40 PROC==1 PS==100
    SubType: JobReservation
    Nodes='node005:1'
    Rsv-Group: Moab.7

Moab.2                      Job   A   -00:01:07     3:58:53      4:00:00    1    2
2
    Flags: ISACTIVE
    ACL:   JOB==Moab.2=
```

```
    CL:    JOB==Moab.2 USER==tuser1 GROUP==tgroup1 QOS==starter JPRIORITY<=0
DURATION==4:00:00 PROC==2 PS==28800
    SubType: JobReservation
    Nodes='node009:1'
    Rsv-Group: Moab.2

Moab.8                      Job   A     3:58:53     7:58:53      4:00:00    8   16
16
    Flags: PREEMPTEE
    ACL:   JOB==Moab.8=
    CL:    JOB==Moab.8 USER==tuser1 GROUP==tgroup1 ACCT==development CLASS==bigmen
QOS==starter JPRIORITY<=0 DURATION==4:00:00 PROC==16 PS==230400
    SubType: JobReservation

Nodes='node009:1,node008:1,node007:1,node006:1,node005:1,node004:1,node003:1,node001:
1'
    Attributes (Priority=148)
    Rsv-Group: idle

system.3                  User bas   -00:01:08    INFINITY     INFINITY    1    1
2
    Flags: ISCLOSED,ISACTIVE
    ACL:   RSV==system.3=
    CL:    RSV==system.3
    Accounting Creds:  User:root
    Task Resources: PROCS: [ALL]
    SubType: Other
    Nodes='node254:1'
    Attributes (HostExp='node254')
    Active PH: 0.00/0.01 (0.00%)
    History: 1322773208:PROCS=2


system.2                  User bas   -00:01:08    INFINITY     INFINITY    1    1
2
    Flags: ISCLOSED,ISACTIVE
    ACL:   RSV==system.2=
    CL:    RSV==system.2
    Accounting Creds:  User:root
    Task Resources: PROCS: [ALL]
    SubType: Other
    Nodes='node255:1'
    Attributes (HostExp='node255')
    Active PH: 0.00/0.01 (0.00%)
    History: 1322773208:PROCS=2


system.1                  User bas   -00:01:08    INFINITY     INFINITY    1    1
2
    Flags: ISCLOSED,ISACTIVE
    ACL:   RSV==system.1=
    CL:    RSV==system.1
    Accounting Creds:  User:root
    Task Resources: PROCS: [ALL]
    SubType: Other
    Nodes='node256:1'
    Attributes (HostExp='node256')
    Active PH: 0.00/0.01 (0.00%)
    History: 1322773208:PROCS=2
```

mdiag -R

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.

Examples

Example 4-19:

```
> $ mdiag -R -v
diagnosing resource managers

RM[internal]  State: ---  Type: SSS  ResourceType: COMPUTE
  Max Fail/Iteration: 0
  JobCounter:         6
  Partition:          SHARED
  RM Performance:     AvgTime=0.00s  MaxTime=0.00s  (55353 samples)
  RM Languages:       -
  RM Sub-Languages:   -

RM[torque]    State: Active  Type: PBS  ResourceType: COMPUTE
  Timeout:            30000.00 ms
  Version:            '4.2.4'
  Job Submit URL:     exec:///opt/torque-4.2/bin/qsub
  Objects Reported:   Nodes=1 (12 procs)  Jobs=1
  Nodes Reported:     1 (N/A)
  Flags:              executionServer
  Partition:          torque
  Event Management:   EPORT=15004  (last event: 00:03:07)
  NOTE:  SSS protocol enabled
  Submit Command:     /opt/torque-4.2/bin/qsub
  DefaultClass:       batch
  Total Jobs Started: 1
  RM Performance:     AvgTime=0.00s  MaxTime=35.00s  (220097 samples)
  RM Languages:       PBS
  RM Sub-Languages:   PBS

RM[torque] Failures:
  clusterquery     (683 of 55349 failed)
        -12days  'cannot connect to PBS server '' (pbs_errno=15033, 'Batch protocol
error')'

NOTE:  use 'mrmctl -f messages <RMID>' to clear stats/failures

RM[FLEXlm]    State: Active  Type: NATIVE  ResourceType: LICENSE
  Timeout:            30000.00 ms
  Cluster Query URL:  exec://$TOOLSDIR/flexlm/license.mon.flexLM.pl
  Licenses Reported:  6 types (250 of 282 available)
  Partition:          SHARED
  License Stats:      Avg License Avail:   239.01  (978 iterations)
  Iteration Summary:  Idle: 396.42  Active: 150.92  Busy: -447.34
  License biocol          50 of  50 available  (Idle: 100.00%  Active: 0.00%)
  License cloudform       100 of 100 available  (Idle: 100.00%  Active: 0.00%)
  License mathworks       8 of  25 available  (Idle: 52.00%  Active: 48.00%)
  License verity          25 of  25 available  (Idle: 100.00%  Active: 0.00%)
  Event Management:   (event interface disabled)
  RM Performance:     AvgTime=0.00s  MaxTime=0.61s  (1307618 samples)
        clusterquery:   AvgTime=0.02s  MaxTime=0.61s  (9465 samples)
          queuequery:   AvgTime=0.00s  MaxTime=0.00s  (1 samples)
        rminitialize:   AvgTime=0.00s  MaxTime=0.00s  (1 samples)
             getdata:   AvgTime=0.17s  MaxTime=0.60s  (978 samples)
  RM Languages:       NATIVE
  RM Sub-Languages:   NATIVE

AM[mam]  Type: MAM  State: 'Active'
  Host:                      localhost
  Port:                      7112
  Timeout:                   15
  Thread Pool Size:          2
  Charge Policy:             DEBITALLWC
  Validate Job Submission:   TRUE
  Create Failure Action:     CANCEL,HOLD
  Start Failure Action:      CANCEL,HOLD

AM[mam] Failures:
```

```
  Fri Jun 21 14:32:45  Create           'Failure registering job Create (1) with
accounting manager -- server rejected request with status code 740 - Insufficient
funds: There are no valid allocations to satisfy the quote'
```

mdiag -S

Synopsis

mdiag -S [-v]

Overview

The mdiag -S command is used to present information about the status of the scheduler.

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

Examples

Example 4-20:

```
> mdiag -S
Moab Server running on orion-1:43225  (Mode: NORMAL)
  Load(5m)  Sched: 12.27%  RMAction: 1.16%  RMQuery: 75.30%  User: 0.29%  Idle: 10.98%
  Load(24h) Sched: 10.14%  RMAction: 0.93%  RMQuery: 74.02%  User: 0.11%  Idle: 13.80%
  HA Fallback Server:  orion-2:43225  (Fallback is Ready)
  Note:  system reservation blocking all nodes
  Message:  profiling enabled (531 of 600 samples/5:00 interval)
```

mdiag -s

Synopsis

mdiag -s [*reservationid*] [-v]>]

Overview

The mdiag -s command allows administrators to look at detailed standing reservation information. It provides the name, type, partition, starttime and endtime, period, task count, host list, and a list of child instances.

Examples

Example 4-21:

```
> mdiag -s
standing reservation overview
RsvID                   Type    Par     StartTime       EndTime         Duration        Period
-----                   ----    ---     ---------       -------         --------        ------

TestSR                  User    ---     00:00:00            ---         00:00:00          DAY
  Days:                 ALL
  Depth:        2
  RsvList:      testSR.1,testSR.2,testSR.3
  HostExp:      'node1,node2,node4,node8'

test2                   User    ---     00:00:00            ---         00:00:00          DAY
  Days:         ALL
  TaskCount:    4
  Depth:        1
  RsvList:      test2.4,test2.5
```

mdiag -T

Synopsis

mdiag -T [*triggerid*] [-v] [--blocking]

Overview

The mdiag -T command is used to present information about each Trigger. The information presented includes Name, State, Action, Event Time. The command gathers information from the Moab cache which prevents it from waiting for the scheduler, but the --blocking option can be used to bypass the cache and allow waiting for the scheduler.

Examples

Example 4-22:

```
> mdiag -T
TrigID                Object ID               Event  AType           ActionDate
State
--------------------- ------------------- -------- ------ -------------------- ------
-----
sched_trig.0          sched:Moab                end   exec                    -
```

```
Blocked
3                     node:node010         threshol   exec                     -
Blocked
5                     job:Moab.7            preempt   exec                     -
Blocked
6                     job:Moab.8            preempt   exec                     -
Blocked
4*                    job:Moab.5              start   exec            -00:00:36
Failure
* indicates trigger has completed
```

Example 4-23:

```
> mdiag -T -v
TrigID                Object ID               Event  AType           ActionDate
State
--------------------- ------------------- -------- ------ -------------------- ------
-----
sched_trig.0          sched:Moab                end   exec                     -
Blocked
  Name:          sched_trig
  Flags:         globaltrig
  BlockUntil:    INFINITY  ActiveTime:  ---
  Action Data:   date
  NOTE:  trigger can launch

3                     node:node010         threshol   exec                     -
Blocked
  Flags:         globaltrig
  BlockUntil:    INFINITY  ActiveTime:  ---
  Threshold:     CPULoad > 3.00  (current value: 0.00)
  Action Data:   date
  NOTE:  trigger cannot launch - threshold not satisfied - threshold type not
supported

5                     job:Moab.7            preempt   exec                     -
Blocked
  Flags:         user,globaltrig
  BlockUntil:    INFINITY  ActiveTime:  ---
  Action Data:   $HOME/tools/preemptnotify.pl $OID $OWNER $HOSTNAME

6                     job:Moab.8            preempt   exec                     -
Blocked
  Flags:         user,globaltrig
  BlockUntil:    INFINITY  ActiveTime:  ---
  Action Data:   $HOME/tools/preemptnotify.pl $OID $OWNER $HOSTNAME
  NOTE:  trigger cannot launch - parent job Moab.8 is in state Idle

4*                    job:Moab.5              start   exec  Mon Jan 16 12:33:00
Failure
  Launch Time:   -00:02:17
  Flags:         globaltrig
  Last Execution State: Failure (ExitCode: 0)
  BlockUntil:    00:00:00  ActiveTime:  00:00:00
  Action Data:   $HOME/tools/preemptnotify.pl $OID $OWNER $HOSTNAME
  ALERT:  trigger failure detected
  Message:       'exec '/usr/test/moab/tools/preemptnotify.pl' cannot be located or is
not executable'

* indicates trigger has completed
```

mdiag -u

Synopsis

mdiag -u [*userid*]

Overview

The mdiag -u command is used to present information about user records maintained by Moab. The information presented includes user name, UID, scheduling priority, default job flags, default QoS level, List of accessible QoS levels, and list of accessible partitions.

Examples

Example 4-24:

```
> mdiag -u
evaluating user information
Name         Priority        Flags         QDef      QOSList*        PartitionList
Target  Limits

jvella              0       [NONE]       [NONE]       [NONE]                [NONE]
0.00  [NONE]
  ALIST=Engineering
  Message:  profiling enabled (597 of 3000 samples/00:15:00 interval)
[NONE]              0       [NONE]       [NONE]       [NONE]                [NONE]
0.00  [NONE]
reynolds            0       [NONE]       [NONE]       [NONE]                [NONE]
0.00  [NONE]
  ALIST=Administration
  Message:  profiling enabled (597 of 3000 samples/00:15:00 interval)
mshaw               0       [NONE]       [NONE]       [NONE]                [NONE]
0.00  [NONE]
  ALIST=Test
  Message:  profiling enabled (584 of 3000 samples/00:15:00 interval)
kforbes             0       [NONE]       [NONE]       [NONE]                [NONE]
0.00  [NONE]
  ALIST=Shared
  Message:  profiling enabled (597 of 3000 samples/00:15:00 interval)
gastor              0       [NONE]       [NONE]       [NONE]                [NONE]
0.00  [NONE]
  ALIST=Engineering
  Message:  profiling enabled (597 of 3000 samples/00:15:00 interval)
```

Note that only users which have jobs which are currently queued or have been queued since Moab was most recently started are listed.

Related topics

- showstats command (display user statistics)

mjobctl

Synopsis

mjobctl -c *jobexp*

mjobctl -c -w *attr=val*

mjobctl -C *jobexp*

mjobctl -e jobid

mjobctl -F *jobexp*

mjobctl -h [User|System|Batch|Defer|All] *jobexp*

mjobctl -m *attr*{+=|=|-=}*val jobexp*

mjobctl -N [<SIGNO>] *jobexp*

mjobctl -n <JOBNAME>

mjobctl -p <PRIORITY> *jobexp*

mjobctl -q {diag|starttime|hostlist} *jobexp*

mjobctl -r *jobexp*

mjobctl -R *jobexp*

mjobctl -s

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
Description	Cancel a job. Use -w (following a -c flag) to specify job cancellation according to given credentials or job attributes. See -c -w for more information.
Example:	`> mjobctl -c job1045` *Cancel job `job1045`.*

-c -w - Cancel Where	
Format	*<ATTR>=<VALUE>* where *<ATTR>*=[user | account | qos | class | reqreservation(RsvName) | state (JobState) | job-name(JobName, not job ID)] | partition
Description	Cancel a job based on a given credential or job attribute. Use `-w` following a -c flag to specify job cancellation according to credentials or job attributes. (See examples.) See **Job States** on page 28 for a list of all valid job states. Also, you can cancel jobs from given partitions using `-w partition=<PAR1>[<PAR2>...]`; however, you must also either use another `-w` flag to specify a job or use the standard job expression.

Example	`> mjobctl -c -w state=USERHOLD`
	Cancels all jobs that currently have a `USERHOLD` on them.
	
	`> mjobctl -c -w user=user1 -w acct=acct1`
	Cancels all jobs assigned to `user1` or `acct1`.

-C - Checkpoint	
Format	JOBEXP
Description	Checkpoint a job. See **Checkpoint/Restart Facilities** on page 452 for more information.
Example	`> mjobctl -C job1045` *Checkpoint job `job1045`.*

-e - Rerun	
Format	JOBID
Description	Rerun the completed TORQUE job. This works only for jobs that are completed and show up in TORQUE as completed. This flag does not work with other resource managers.

-e - Rerun	
Example	`> mjobctl -e job1045` *Rerun job `job1045`.*

-F - Force Cancel	
Format	JOBEXP
Description	Forces a job to cancel and ignores previous cancellation attempts.
Example	`> mjobctl -F job1045` *Force cancel job `job1045`.*

-h - Hold	
Format	*<HOLDTYPE><JOBEXP>* *<HOLDTYPE>* = { user | batch | system | defer | ALL }
Default	user
Description	Set or release a job hold See Job Holds for more information

Example	`> mjobctl -h user job1045`
	Set a user hold on job `job1045`.
	`> mjobctl -u all job1045`
	Unset all holds on job `job1045`.

-m - Modify

Format

`<ATTR>{ += | =| -= } <VAL>`

<*ATTR*>={ account | arraylimit | awduration| class | deadline | depend | eeduration | env | features | feature | flags | gres | group | hold | hostlist | jobdisk | jobmem | jobname | jobswap | loglevel | messages | minstarttime | nodecount | notificationaddress | partition | priority | queue | qos | reqreservation | rmxstring | reqawduration | sysprio | trig | trigvar | userprio | var | wclimit}

Description

Modify a specific job attribute.

> If an mjobctl -m attribute can affect how a job starts, then it generally cannot affect a job that is already running. For example, it is not feasible to change the **hostlist** of a job that is already running.

The userprio attribute allows you to specify user priority. For job priority, use the '-p' flag.

Modification of the job dependency is also communicated to the resource manager in the case of SLURM and PBS/Torque.

Adding `--flags=warnifcompleted` causes a warning message to print when a job completes.

To define values for `awduration`, `eeduration`, `minstarttime` (Note that the `minstarttime` attribute performs the same function as msub -a.), `reqawduration`, and `wclimit`, use the time spec format.

A non-active job's partition list can be modified by adding or subtracting partitions. Note, though, that when adding or subtracting multiple partitions, each partition must have its own `-m partition{+= | = | -=}name` on the command line. (See example for adding multiple partitions.)

To modify a job's generic resources, use the following format: `gres{ += | = | -= } <gresName>[:<count>]`. <*gresName*> is a single resource, not a list. <*count*> is an integer that, if not specified, is assumed to be 1. Modifying a job's generic resources causes Moab to append the new gres (+=), subtract the specified gres (-=), or clear out all existing generic resources attached to the job and override them with the newly-specified one (=).

-m - Modify

Example

```
> mjobctl -m reqawduration+=600 1664
```

Add 10 minutes to the job walltime.

```
> mjobctl -m eeduration=-1 1664
```

Reset job's effective queue time, to when the job was submitted.

```
> mjobctl -m var=Flag1=TRUE 1664
```

Set the job variable `Flag1` to `TRUE`.

```
> mjobctl -m notificationaddress="name@server.com"
```

Sets the notification e-mail address associated with a job to `name@server.com`.

```
> mjobctl -m partition+=p3 -m partition+=p4 Moab.5
```

Adds multiple partitions (p3 and p4) to job `Moab.5`.

```
> mjobctl -m arraylimit=10 sim.25
```

Changes the concurrently running sub-job limit to 10 for array `sim.25`.

```
> mjobctl -m gres=matlab:1 job0201
```

Overrides all generic resources applied to job `job0201` and replaces them with 1 `matlab`.

```
> mjobctl -m userprio-=100 Moab.4
```

Reduces the user priority of `Moab.4` by 100.

-N - Notify

Format	[signal=]<*SIGID*>JOBEXP
Description	Send a signal to all jobs matching the job expression.

-N - Notify	

Example	`> mjobctl -N INT 1664`
	Send an interrupt signal to job 1664.
	`> mjobctl -N 47 1664`
	Send signal 47 to job `1664`.

-n - Name	
Format	
Description	Select jobs by job name.
Example	

-p - Priority	
Format	[+|+=|-=]<*VAL*><*JOBID*> [--flags=relative]
Description	Modify a job's system priority.

-p - Priority

Example

Priority is the job priority plus the system priority. Each format affects the job and system priorities differently. Using the format *<VAL><JOBID>* or *+<VAL><JOBID>* will set the system priority to the maximum system priority plus the specified value. Using *+=<VAL><JOBID>* or *<VAL><JOBID>* `--flags=relative` will relatively increase the job's priority and set the system priority. Using the format *−=<VAL> <JOBID>* sets the system priority to 0, and does not change priority based on *<VAL>* (it will not decrease priority by that number).

For the following example, `job1045` has a priority of 10, which is composed of a job priority of 10 and a system priority of 0.

```
> mjobctl -p +1000 job1045
```

The system priority changes to the max system priority plus `1000` points, ensuring that this job will be higher priority than all normal jobs. In this case, the job priority of 10 is not added, so the priority of `job1045` is now 1000001000.

```
> mjobctl -p -=1 job1045
```

The system priority of `job1045` resets to 0. The job priority is still 10, so the overall priority becomes 10.

```
> mjobctl -p 3 job1045 --flags=relative
```

Adds 3 points to the relative system priority. The priority for `job1045` changes from 10 to 13.

-q - Query

Format	[diag(ALL)| hostlist | starttime| template] *<JOBEXP>*
Description	Query a job.

-q - Query

Example

```
> mjobctl -q diag job1045
```

Query job job1045.

```
> mjobctl -q diag ALL --format=xml
```

Query all jobs and return the output in machine-readable XML.

```
> mjobctl -q starttime job1045
```

Query starttime of job job1045.

```
> mjobctl -q template <job>
```

Query job templates. If the <job> is set to ALL or empty, it will return information for all job templates.

```
> mjobctl -q wiki <jobName>
```

Query a job with the output displayed in a WIKI string. The job's name may be replaced with ALL.

`--flags=completed` will only work with the `diag` option.

-r - Resume

Format	JOBEXP
Description	Resume a job.
Example	`> mjobctl -r job1045` *Resume job job1045.*

-R - Requeue

Format	JOBEXP

-R - Requeue	
Description	Requeue a job.
Example	`> mjobctl -R job1045` *Requeue job job1045.*

-s - Suspend	
Format	JOBEXP
Description	Suspend a job. For more information, see Suspend/Resume Handling.
Example	`> mjobctl -s job1045` *Suspend job job1045.*

-u - Unhold	
Format	[<*TYPE*>[,<*TYPE*>]]JOBEXP <*TYPE*> = [user | system | batch | defer | ALL]
Default	ALL
Description	Release a hold on a job See **Job Holds** on page 449 for more information.
Example	`> mjobctl -u user,system scrib.1045` *Release user and system holds on job scrib.1045.*

-w - Where	
Format	[CompletionTime | StartTime][<= | = | >=]<*EPOCH_TIME*>
Description	Add a where constraint clause to the current command. As it pertains to `CompletionTime` | `StartTime`, the where constraint only works for completed jobs. CompletionTime filters according to the completed jobs' completion times; StartTime filters according to the completed jobs' start times.
Example	`> mjobctl -q diag ALL --flags=COMPLETED --format=xml -w CompletionTime>=1246428000 -w CompletionTime<=1254376800` *Prints all completed jobs still in memory that completed between July 1, 2009 and October 1, 2009.*

-x - Execute	
Format	JOBEXP
Description	Execute a job. The `-w` option allows flags to be set for the job. Allowable flags are, `ignore-policies`, `ignorenodestate`, and `ignorersv`.

Example	`> mjobctl -x job1045`
	Execute job `job1045`.
	`> mjobctl -x -w flags=ignorepolicies job1046`
	Execute job `job1046` and ignore policies, such as MaxJobPerUser.

Parameters

JOB EXPRESSION	
Format	<*STRING*>

JOB EXPRESSION

Description

The name of a job or a regular expression for several jobs. The flags that support job expressions can use node expression syntax as described in Node Selection. Using `x:` indicates the following string is to be interpreted as a regular expression, and using `r:` indicates the following string is to be interpreted as a range. Job expressions do not work for array sub-jobs.

Moab uses regular expressions conforming to the POSIX 1003.2 standard. This standard is somewhat different than the regular expressions commonly used for filename matching in Unix environments (see man 7 regex). To interpret a job expression as a regular expression, use `x:` or in the Moab configuration file (`moab.cfg`), set the parameter USEJOBREGEX to *TRUE* (and take note of the following caution).

If you set USEJOBREGEX to *TRUE*, Moab treats all mjobctl job expressions as regular expressions regardless of whether wildcards are specified. This should be used with extreme caution since there is high potential for unintended consequences. For example, specifying `canceljob m.1` will not only cancel `m.1`, but also `m.11`,`m.12`,`m13`, and so on.

In most cases, it is necessary to quote the job expression (for example, `job13[5-9]`) to prevent the shell from intercepting and interpreting the special characters.

The mjobctl command accepts a comma delimited list of job expressions. Example usage might be `mjobctl -r job[1-2],job4` or `mjobctl -c job1,job2,job4`.

Example:

```
> mjobctl -c "x:80.*"
job '802' cancelled
job '803' cancelled
job '804' cancelled
job '805' cancelled
job '806' cancelled
job '807' cancelled
job '808' cancelled
job '809' cancelled
```

Cancel all jobs starting with `80`.

```
> mjobctl -m priority+=200 "x:74[3-5]"
job '743' system priority modified
job '744' system priority modified
job '745' system priority modified
```

```
> mjobctl -h x:17.*
# This puts a hold on any job that has a 17 that is followed by an unlimited amount
of any
# character and includes jobs 1701, 17mjk10, and 17DjN_JW-07

> mjobctl -h r:1-17
# This puts a hold on jobs 1 through 17.
```

XML Output

mjobctl information can be reported as XML as well. This is done with the command `mjobctl -q diag <JOB_ID>`.

XML Attributes

Name	Description
Account	The account assigned to the job
AllocNodeList	The nodes allocated to the job
Args	The job's executable arguments
AWDuration	The active wall time consumed
BlockReason	The block message index for the reason the job is not eligible
Bypass	Number of times the job has been bypassed by other jobs
Calendar	The job's timeframe constraint calendar
Class	The class assigned to the job
CmdFile	The command file path
CompletionCode	The return code of the job as extracted from the RM
CompletionTime	The time of the job's completion
Cost	The cost of executing the job relative to an allocation manager
CPULimit	The CPU limit for the job
Depend	Any dependencies on the status of other jobs
DRM	The master destination RM
DRMJID	The master destination RM job ID
EEDuration	The duration of time the job has been eligible for scheduling

Name	Description
EFile	The stderr file
Env	The job's environment variables set for execution
EnvOverride	The job's overriding environment variables set for execution
EState	The expected state of the job
EstHistStartTime	The estimated historical start time
EstPrioStartTime	The estimated priority start time
EstRsvStartTime	The estimated reservation start time
EstWCTime	The estimated walltime the job will execute
ExcHList	The excluded host list
Flags	Command delimited list of Moab flags on the job
GAttr	The requested generic attributes
GJID	The global job ID
Group	The group assigned to the job
Hold	The hold list
Holdtime	The time the job was put on hold
HopCount	The hop count between the job's peers
HostList	The requested host list
IFlags	The internal flags for the job
IsInteractive	If set, the job is interactive
IsRestartable	If set, the job is restartable

Name	Description
IsSuspendable	If set, the job is suspendable
IWD	The directory where the job is executed
JobID	The job's batch ID.
JobName	The user-specified name for the job
JobGroup	The job ID relative to its group
LogLevel	The individual log level for the job
MasterHost	The specified host to run primary tasks on
Messages	Any messages reported by Moab regarding the job
MinPreemptTime	The minimum amount of time the job must run before being eligible for preemption
Notification	Any events generated to notify the job's user
OFile	The stdout file
OldMessages	Any messages reported by Moab in the old message style regarding the job
OWCLimit	The original wallclock limit
PAL	The partition access list relative to the job
QueueStatus	The job's queue status as generated this iteration
QOS	The QoS assigned to the job
QOSReq	The requested QoS for the job
ReqAWDuration	The requested active walltime duration
ReqCMaxTime	The requested latest allowed completion time
ReqMem	The total memory requested/dedicated to the job

Name	Description
ReqNodes	The number of requested nodes for the job
ReqProcs	The number of requested procs for the job
ReqReservation	The required reservation for the job
ReqRMType	The required RM type
ReqSMinTime	The requested earliest start time
RM	The master source resource manager
RMXString	The resource manager extension string
RsvAccess	The list of reservations accessible by the job
RsvStartTime	The reservation start time
RunPriority	The effective job priority
Shell	The execution shell's output
SID	The job's system ID (parent cluster)
Size	The job's computational size
STotCPU	The average CPU load tracked across all nodes
SMaxCPU	The max CPU load tracked across all nodes
STotMem	The average memory usage tracked across all nodes
SMaxMem	The max memory usage tracked across all nodes
SRMJID	The source RM's ID for the job
StartCount	The number of the times the job has tried to start
StartPriority	The effective job priority

Name	Description
StartTime	The most recent time the job started executing
State	The state of the job as reported by Moab
StatMSUtl	The total number of memory seconds utilized
StatPSDed	The total number of processor seconds dedicated to the job
StatPSUtl	The total number of processor seconds utilized by the job
StdErr	The path to the stderr file
StdIn	The path to the stdin file
StdOut	The path to the stdout file
StepID	StepID of the job (used with LoadLeveler systems)
SubmitHost	The host where the job was submitted
SubmitLanguage	The RM language that the submission request was performed
SubmitString	The string containing the entire submission request
SubmissionTime	The time the job was submitted
SuspendDuration	The amount of time the job has been suspended
SysPrio	The admin specified job priority
SysSMinTime	The system specified min. start time
TaskMap	The allocation taskmap for the job
TermTime	The time the job was terminated
User	The user assigned to the job
UserPrio	The user specified job priority

Name	Description
UtlMem	The utilized memory of the job
UtlProcs	The number of utilized processors by the job
Variable	
VWCTime	The virtual wallclock limit

Examples

Example 4-25:

```
> mjobctl -q diag ALL --format=xml
<Data><job AWDuration="346" Class="batch" CmdFile="jobsleep.sh" EEDuration="0"
EState="Running" Flags="RESTARTABLE" Group="test" IWD="/home/test" JobID="11578"
QOS="high"
RMJID="11578.lolo.icluster.org" ReqAWDuration="00:10:00" ReqNodes="1" ReqProcs="1"
StartCount="1"
StartPriority="1" StartTime="1083861225" StatMSUtl="903.570" StatPSDed="364.610"
StatPSUtl="364.610"
State="Running" SubmissionTime="1083861225" SuspendDuration="0" SysPrio="0"
SysSMinTime="00:00:00"
User="test"><req AllocNodeList="hana" AllocPartition="access" ReqNodeFeature="[NONE]"
ReqPartition="access"></req></job><job AWDuration="346" Class="batch"
CmdFile="jobsleep.sh"
EEDuration="0" EState="Running" Flags="RESTARTABLE" Group="test" IWD="/home/test"
JobID="11579"
QOS="high" RMJID="11579.lolo.icluster.org" ReqAWDuration="00:10:00" ReqNodes="1"
ReqProcs="1"
StartCount="1" StartPriority="1" StartTime="1083861225" StatMSUtl="602.380"
StatPSDed="364.610"
StatPSUtl="364.610" State="Running" SubmissionTime="1083861225" SuspendDuration="0"
SysPrio="0"
SysSMinTime="00:00:00" User="test"><req AllocNodeList="lolo" AllocPartition="access"
ReqNodeFeature="[NONE]" ReqPartition="access"></req></job></Data>
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- setspri
- canceljob
- runjob

mnodectl

Synopsis

mnodectl -m *attr*{=|-=}*val* *nodeexp*
mnodectl -q [*cat*|*diag*|*profile*|*wiki*] *nodeexp*

Overview

Change specified attributes for a given node expression.

Access

By default, this command can be run by any Moab Administrator.

Format

-m - Modify

Format	*<ATTR>*{=|-=}*<VAL>* Where *<ATTR>* is one of the following: FEATURES GEVENT, GMETRIC, MESSAGE, OS, POWER, STATE, VARIABLE and −=, in this case, clears the attribute; it does not decrement the attribute's value. Changing OS and POWER require a Moab Adaptive Computing Suite license and a provisioning resource manager.
Description	Modify the state or attribute of specified node(s)

Example	`> mnodectl -m features=fastio,highmem node1`
	`> mnodectl -m gevent=cpufail:'cpu02 has failed w/ec:0317' node1`
	`> mnodectl -m gmetric=temp:131.2 node1`
	`> mnodectl -m message='cpufailure:cpu02 has failed w/ec:0317' node1`
	`> mnodectl -m OS=RHAS30 node1`
	`> mnodectl -m power=off node1`
	`> mnodectl -m state=idle node1`
	`> mnodectl -m variable=IP=10.10.10.100,Location=R1S2 node1`

-q - Query

Format	{cat | diag | profile | wiki}
Description	Query node categories or node profile information (see ENABLEPROFILING for nodes). The `diag` and `profile` options must use `--xml`.

-q - Query

Example

```
> mnodectl -q cat ALL
node categorization stats from Mon Jul 10 00:00:00 to Mon Jul 10 15:30:00
Node: moab
 Categories:
                     busy: 96.88%
                     idle: 3.12%
Node: maka
 Categories:
                     busy: 96.88%
                     idle: 3.12%
Node: pau
 Categories:
                     busy: 96.88%
                     idle: 3.12%
Node: maowu
 Categories:
                     busy: 96.88%
                  down-hw: 3.12%
Cluster Summary:
                     busy: 96.88%
                  down-hw: 0.78%
                     idle: 2.34%
```

```
> mnodectl -v -q profile
...
```

```
> mnodectl -q wiki <ALL>
GLOBAL STATE=Idle PARTITION=SHARED
n0 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n1 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n2 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n3 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n4 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n5 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n6 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n7 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n8 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
n9 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED
```

Query a node with the output displayed in a WIKI string.

Parameters

FEATURES	
Format	*<STRING>* One of the following: • a comma-delimited list of features • [NONE] (to clear features on the node)
Description	Sets the features on a node. These node features will be overwritten when an RM reports features.
Example	`mnodectl -m features=fastio,highmem node1` `mnodectl -m features=[NONE] node1`

GEVENT	
Format	*<EVENT>:<MESSAGE>*
Description	Creates a generic event on the node to which Moab may respond (see Enabling Generic Events).
Example	`mnodectl -m gevent=powerfail:'power has failed' node1`

GMETRIC	
Format	*<ATTR>:<VALUE>*
Description	Sets the value for a generic metric on the node (see Enabling Generic Metrics). When a gmetric set in Moab conflicts with what the resource manager reports, Moab uses the set gmetric until the next time the resource manager reports a different number.
Example	`mnodectl -m gmetric=temp:120 node1`

MESSAGE	
Format	'*<MESSAGE>*'

MESSAGE	
Description	Sets a message to be displayed on the node.
Example	`mnodectl -m message='powerfailure: power has failed' node1`

NODEEXP	
Format	*<STRING>* Where *<NODEEXP>* is a node name, regex or ALL
Description	Identifies one or more nodes.

Example	`node1` — applies only to `node1`
	`fr10n*` - all nodes starting with `fr10n`
	`ALL` - all known nodes

OS	
Format	*<STRING>*
Description	Operating System (see Resource Provisioning).
Example	`mnodectl node1 -m OS=RHELAS30`

POWER	
Format	{off|on}

POWER	
Description	Set the power state of a node. Action will NOT be taken if the node is already in the specified state. If you power off a node, a green policy will try to turn it back on. If you want the node to remain powered off, you must associate a reservation with it. If you request to power off a node that has active work on it, Moab will return a status indicating that the node is busy (with a job or VM) and will not be powered off. You will see one of these messages: • `Ignoring node <name>: power ON in process` (indicates node is currently powering on) • `Ignoring node <name>: power OFF in process` (indicates node is currently powering off) • `Ignoring node <name>: has active VMs running` (indicates the node is currently running active VMs) • `Ignoring node <name>: has active jobs running` (indicates the node is currently running active jobs) Once you resolve the activity on the node (by preempting or migrating the jobs or VMs, for example), you can attempt to power the node off again. You can use the `--flags=force` option to cause a force override. However, doing this will power off the node regardless of whether or not its jobs get migrated or preempted (i.e., you run the risk of losing the VMs/jobs entirely). For example: `> mnodectl node1 -m power=off --flags=force`
Example	`> mnodectl node1 -m power=off`

STATE	
Format	{drained|idle}
Description	Remove (drained) or add (idle) a node from scheduling.
Example	`mnodectl node1 -m state=drained` *Moab ignores `node1` when scheduling.*

VARIABLE	
Format	*<name>*[=*<value>*],*<name>*[=*<value>*]...

VARIABLE	
Description	Set a list of variables for a node.
Example	`> mnodectl node1 -m variable=IP=10.10.10.100,Location=R1S2`

Related topics

- Moab Client Installation — explains how to distribute this command to client nodes
- mdiag -n
- showres -n
- checknode
- showstats -n — report current and historical node statistics

moab

Synopsis

moab --about --help --loglevel=<LOGLEVEL> --version [-c <CONFIG_FILE>] [-C] [-d] [-e] [-h] [-P [<PAUSEDURATION>]] [-R <RECYCLEDURATION>] [-s] [-S [<STOPITERATION>]] [-v]

Parameters

Parameter	Description
--about	Displays build environment and version information.
--loglevel	Sets the server loglevel to the specified value.
--version	Displays version information.
-c	Configuration file the server should use.
-C	Clears checkpoint files (`.moab.ck`, `.moab.ck.1`).
-d	Debug mode (does not background itself).

Parameter	Description
-e	Forces Moab to exit if there are any errors in the configuration file, if it can't connect to the configured database, or if it can't find these directories: • `statdir` • `logdir` • `spooldir` • `toolsdir`
-P	Starts Moab in a paused state for the duration specified.
-R	Causes Moab to automatically recycle every time the specified duration transpires.
-s	Starts Moab in the state that was most recently checkpointed.
-S	Suspends/stops scheduling at specified iteration (or at startup if no iteration is specified).
-v	Same as --version.

mrmctl

Synopsis

mrmctl -f [*fobject*] {*rmName*|am:[*amid*]} mrmctl -l [*rmid*|am:[*amid*]] mrmctl -m <attr>=<value> [*rmid*|am:[*amid*]] mrmctl -p {*rmid*|am:[*amid*]} mrmctl -R {am|id}[:*rmid*]}}

Overview

mrmctl allows an admin to query, list, modify, and ping the resource managers and allocation managers in Moab. mrmctl also allows for a queue (often referred to as a class) to be created for a resource manager.

Access

By default, this command can be run by level 1 and level 2 Moab administrators (see ADMINCFG).

Format

-f - Flush Statistics	
Format	[<*fobject*>] where *fobject* is optional and one of messages or stats.

-f - Flush Statistics	
Default	If no *fobject* is specified, then reported failures and performance data will be flushed. If no resource manager id is specified, the first resource manager will be flushed.
Description	Clears resource manager statistics. If messages is specified, then reported failures, performance data, and messages will be flushed.
Example	`> mrmctl -f base` *Moab will clear the statistics for RM `base`.*

-l - List	
Format	N/A
Default	All RMs and AMs (when no RM/AM is specified)
Description	List Resource and Allocation Manager(s)
Example	`> mrmctl -l` *Moab will list all resource and allocation managers.*

-m - Modify	
Format	N/A
Default	All RMs and AMs (when no RM/AM is specified).
Description	Modify Resource and Allocation Manager(s).
Example	`> mrmctl -m state=disabled peer13`

-p - Ping	
Format	N/A

-p - Ping	
Default	First RM configured.
Description	Ping Resource Manager.
Example	`> mrmctl -p base` *Moab will ping RM base.*

-R - Reload	
Format	{am|id}[:*rmid*]}
Description	Dynamically reloads server information for the identity manager service if `id` is specified; if `am` is specified, reloads the allocation manager service.
Example	`> mrmctl -R id` `Reloads the identity manager on demand.`

Resource manager interfaces can be enabled/disabled using the modify operation to change the resource manager state as in the following example:

```
# disable active resource manager interface
> mrmctl -m state=disabled torque
# restore disabled resource manager interface
> mrmctl -m state=enabled torque
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mdiag -R
- mdiag -c

mrsvctl

Synopsis

mrsvctl -c [-a *acl*] [-b *subtype*] [-d *duration*] [-D *description*] [-e *endtime*] [-E] [-f *features*] [-F *flags*] [-g *rsvgroup*] [-h *hostexp*] [-n *name*] [-o *owner*] [-p *partition*] [-P *profile*] [-R *resources*] [-s *starttime*] [-S *setvalue*] [-t *tasks*] [-T *trigger*] [-V *variable*] [-x joblist]

mrsvctl -C [-g *standing_reservationid*] {*reservationid*}

mrsvctl -l [{*reservationid* | -i *index*}]

mrsvctl -m <duration|endtime|reqtaskcount|starttime>{=|+=|-=}<VAL> <hostexp>{+=|-=}<VAL> <variable> {+=KEY=VAL|-=KEY_TO_REMOVE} {*reservationid* | -i *index*}

mrsvctl -q {*reservationid* | -i *index*} [--blocking]

mrsvctl -r {*reservationid* | -i *index*}

Overview

mrsvctl controls the creation, modification, querying, and releasing of reservations.

The timeframe covered by the reservation can be specified on either an absolute or relative basis. Only jobs with credentials listed in the reservation's access control list can utilize the reserved resources. However, these jobs still have the freedom to utilize resources outside of the reservation. The reservation will be assigned a name derived from the ACL specified. If no reservation ACL is specified, the reservation is created as a system reservation and no jobs will be allowed access to the resources during the specified timeframe (valuable for system maintenance, etc). See the Reservation Overview for more information.

Reservations can be viewed using the `-q` flag and can be released using the `-r` flag.

By default, reservations are not exclusive and may overlap with other reservations and jobs. Use the '-E' flag to adjust this behavior.

Access

By default, this command can be run by level 1 and level 2 Moab administrators (see ADMINCFG).

Format

-a	
Name	ACL
Format	*<TYPE>*==*<VAL>*[,*<TYPE>*==*<VAL>*]... Where *<TYPE>* is one of the following: ACCT, CLASS, DURATION, GROUP, JATTR, PROC, QOS, RSV, or USER

-a	
Description	List of limitations for access to the reserved resources (See also: ACL Modifiers).

Example	`> mrsvctl -c -h node01 -a USER==john+,CLASS==batch-`
	Moab will make a reservation on `node01` allowing access to user john and restricting access from class batch when other resources are available to class batch
	
	`> mrsvctl -m -a USER-=john system.1`
	Moab will remove user john from the `system.1` reservation

-a

Notes

- When you specify multiple credentials, a user must only match one of them in order to access the reservation. To require one or more of the listed limitations for reservation access, each required specification must end with an asterisk (*). If a user meets the required limitation(s), he or she has access to the reservation (without meeting any that are not marked required).
- There are three different assignment operators that can be used for modifying most credentials in the ACL. The operator == will reassess the list for that particular credential type. The += operator will append to the list for that credential type, and −= will remove from the list. Two other operators are used to specify DURATION and PROC: >= (greater than) and <= (less than).
- To add multiple credentials of the same type with one command, use a colon to separate them. To separate lists of different credential types, use commas. For example, to reassign the user list to consist of users Joe and Bob, and to append the group MyGroup to the groups list on the `system.1` reservation, you could use the command `mrsvctl -m -a USER==Joe:Bob,GROUP+=MyGroup system.1`.
- Any of the ACL modifiers may be used. When using them, it is often useful to put single quotes on either side of the assignment command. For example, `mrsvctl -m -a 'USER==&Joe' system.1`.
- Some flags are mutually exclusive. For example, the ! modifier means that the credential is blocked from the reservation and the & modifier means that the credential must run on that reservation. Moab will take the most recently parsed modifier. Modifiers may be placed on either the left or the right of the argument, so `USER==&JOE` and `USER==JOE&` are equivalent. Moab parses each argument starting from right to left on the right side of the argument, then from left to right on the left side. So, if the command was `USER==!JOE&`, Moab would keep the equivalent of `USER==!JOE` because the `!` would be the last one parsed.
- You can set a reservation to have a time limit for submitted jobs using DURATION and the `*` modifier. For example, `mrsvctl -m -a 'DURATION<=*1:00:00' system.1` would cause the `system.1` reservation to not accept any jobs with a walltime greater than one hour. Similarly, you can set a reservation to have a processor limit using PROC and the `*` modifier. `mrsvctl -a 'PROC>=2*' system.2` would cause the `system.2` reservation to only allow jobs requesting more than 2 procs to run on it.
- You can verify the ACL of a reservation using the mdiag -r command.

```
mrsvctl -m -a 'USER==Joe:Bob,GROUP-=BadGroup,ACCT+=GoodAccount,DURATION<=*1:00:00'
system.1
```

Moab will reassign the `USER` list to be `Joe` and `Bob`, will remove `BadGroup` from the `GROUP` list, append `GoodAccount` to the `ACCT` list, and only allow jobs that have a submitted walltime of an hour or less on the `system.1` reservation.

```
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`

-a	
	USER==Joe:Bob system.1 or *mrsvctl -m -a USER==Joe,USER+=Bob system.1.*

-b	
Name	SUBTYPE
Format	One of the node category values or node category shortcuts.
Description	Add subtype to reservation.
Example	`> mrsvctl -c -b SoftwareMaintenance -t ALL` *Moab will associate the reserved nodes with the node category SoftwareMaintenance.*

-c	
Name	CREATE
Format	*<ARGUMENTS>*
Description	Creates a reservation. The `-x` flag, when used with `-F ignjobrsv`, lets users create reservations but exclude certain nodes from being part of the reservation because they are running specific jobs. The `-F` flag instructs mrsvctl to still consider nodes with current running jobs.

-c	

Examples	`> mrsvctl -c -t ALL`
	Moab will create a reservation across all system resources.
	`> mrsvctl -c -t 5 -F ignjobrsv -x moab.5,moab.6`
	Moab will create the reservation while assigning the nodes. Nodes running jobs `moab5` and `moab6` will not be assigned to the reservation.
	`> mrsvctl -c -d INFINITY`
	Moab will create an infinite reservation.

-C	
Name	CLEAR
Format	*<RSVID>* | -g *<SRSVID>*
Description	Clears any disabled time slots from standing reservations and allows the recreation of disabled reservations
Example	`> mrsvctl -C -g testing` *Moab will clear any disabled timeslots from the standing reservation `testing`.*

-d	
Name	DURATION
Format	`[[[DD:]HH:]MM:]SS`
Default	INFINITY
Description	Duration of the reservation (not needed if ENDTIME is specified)

-d	

Example	`> mrsvctl -c -h node01 -d 5:00:00`
	Moab will create a reservation on `node01` lasting 5 hours.
	
	`mrsvctl -c -d INFINITY`
	Moab will create a reservation with a duration of INFINITY (no endtime).

-D	
Name	DESCRIPTION
Format	*<STRING>*
Description	Human-readable description of reservation or purpose
Example	`> mrsvctl -c -h node01 -d 5:00:00 -D 'system maintenance to test network'` *Moab will create a reservation on `node01` lasting 5 hours.*

-e	
Name	ENDTIME
Format	[HH[:MM[:SS]]][_MO[/DD[/YY]]] or +[[[DD:]HH:]MM:]SS
Default	INFINITY
Description	Absolute or relative time reservation will end (not required if Duration specified). ENDTIME also supports an epoch timestamp.

-e	
Example	`> mrsvctl -c -h node01 -e +3:00:00` *Moab will create a reservation on `node01` ending in 3 hours.*

-E	
Name	EXCLUSIVE
Description	When specified, Moab will only create a reservation if there are no other reservations (exclusive or otherwise) which would conflict with the time and space constraints of this reservation. If exceptions are desired, the rsvaccesslist attribute can be set or the ignrsv flag can be used.
Example	`> mrsvctl -c -h node01 -E` *Moab will only create a reservation on `node01` if no conflicting reservations are found.* This flag is only used at the time of reservation creation. Once the reservation is created, Moab allows jobs into the reservation based on the ACL. Also, once the exclusive reservation is created, it is possible that Moab will overlap it with jobs that match the ACL.

-f	
Name	FEATURES
Format	*<STRING>*[:*<STRING>*]...
Description	List of node features which must be possessed by the reserved resources. You can use a backslash and pipe to delimit features to indicate that Moab can use one or the other.
Example	`> mrsvctl -c -h node[0-9] -f fast|slow` *Moab will create a reservation on nodes matching the expression and which also have either the feature `fast` or the feature `slow`.*

-F	
Name	FLAGS
Format	*<flag>*[[,*<flag>*]...]
Description	Comma-delimited list of flags to set for the reservation (see Managing Reservations for flags).
Example	`> mrsvctl -c -h node01 -F ignstate` *Moab will create a reservation on `node01` ignoring any conflicting node states.*

-g	
Name	RSVGROUP
Format	*<STRING>*
Description	For a create operation, create a reservation in this reservation group. For list and modify operations, take actions on all reservations in the specified reservation group. The `-g` option can also be used in conjunction with the -r option to release a reservation associated with a specified group. See Reservation Group for more information.
Example	`> mrsvctl -c -g staff -h 'node0[1-9]'` *Moab will create a reservation on nodes matching the node expression given and assign it to the reservation group `staff`.*

-h	
Name	host list
Format	class:*<classname>*[,*<classname>*]... or *<STRING>* or 'r:*<nodeNameStart>*[*<beginRange>*-*<endRange>*]' or ALL

-h	
Description	Host expression or a class mapping indicating the nodes which the reservation will allocate. When you specify a *<STRING>*, the HOSTLIST attribute is always treated as a regular expression. `foo10` will map to `foo10`, `foo101`, `foo1006`, etc. To request an exact host match, the expression can be bounded by the carat and dollar op expression markers as in `^foo10$`.

Example	`> mrsvctl -c -h 'r:node0[1-9]'`
	
	Moab will create a reservation on nodes `node01`, `node02`, `node03`, `node04`, `node05`, `node06`, `node07`, `node08`, and `node09`.
	
	`> mrsvctl -c -h class:batch`
	
	Moab will create a reservation on all nodes which support class/queue `batch`.

-i	
Name	INDEX
Format	*<STRING>*
Description	Use the reservation index instead of full reservation ID.
Example	`> mrsvctl -m -i 1 starttime=+5:00` *Moab will create a reservation on nodes matching the expression given.*

-l	
Name	LIST
Format	*<RSV_ID>* or ALL *RSV_ID* can be the name of a reservation or a regular expression.
Default	ALL

-l	
Description	List reservation(s).
Example	`> mrsvctl -l system*` *Moab will list all of the reservations whose names start with `system`.*

-m	
Name	MODIFY
Format	*<ATTR>*=*<VAL>*[-m *<ATTR2>*=*<VAL2>*]... Where *<ATTR>* is one of the following:
Description	Modify aspects of a reservation.

flags	
duration	duration{+=|-=|=}*<RELTIME>*
endtime	endtime{+=|-=}*<RELTIME>* or endtime=*<ABSTIME>*
hostexp	hostexp[+=|-=]*<node>*[,*<node>*]
variable	variable[+=key1=val1|-=key_to_remove]
reqtaskcount	reqtaskcount{+=|-=|=}*<TASKCOUNT>*
starttime	starttime{+=|-=}*<RELTIME>* or starttime=*<ABSTIME>*

-m

Example

```
> mrsvctl -m duration=2:00:00 system.1
```

Moab sets the duration of reservation `system.1` to be exactly two hours, thus modifying the endtime of the reservation.

```
> mrsvctl -m starttime+=5:00:00 system.1
```

Moab advances the starttime of `system.1` five hours from its current starttime (without modifying the duration of the reservation).

```
> mrsvctl -m endtime-=5:00:00 system.1
```

Moab moves the endtime of reservation `system.1` ahead five hours from its current endtime (without modifying the starttime; thus, this action is equivalent to modifying the duration of the reservation).

```
> mrsvctl -m starttime=15:00:00_7/6/08 system.1
```

Moab sets the starttime of reservation `system.1` to 3:00 p.m. on July 6, 2008.

```
> mrsvctl -m starttime-=5:00:00 system.1
```

Moab moves the starttime of reservation `system.1` ahead five hours.

```
> mrsvctl -m starttime+=5:00:00 system.1
```

Moab moves the starttime of reservation `system.1` five hours from the current time.

```
> mrsvctl -m -duration+=5:00:00 system.1
```

Moab extends the duration of `system.1` by five hours.

```
> mrsvctl -m flags+=ADVRES system.1
```

Moab adds the flag `ADVRES` to reservation `system.1`.

```
> mrsvctl -m variable+key1=val1 system.1
```

-m

Moab adds the variable `key1` with the value `key2` to `system.1`.

```
> mrsvctl -m variable+=key1=val1 variable+=key2=val2 system.1
```

Moab adds the variable `key1` with the value `val1`, and variable `key2` with `val2` to `system.1`. (Note that each variable flag requires a distinct `-m` entry.)

```
> mrsvctl -m variable-=key1 system.1
```

Moab deletes the variable `key1` from `system.1`.

```
> mrsvctl -m variable-=key1 -m variable-=key2 system.1
```

Moab deletes the variables `key1` and `key2` from `system.1`.

-m	

Notes:	• Modifying the starttime does not change the duration of the reservation, so the endtime changes as well. The starttime can be changed to be before the current time, but if the change causes the endtime to be before the current time, the change is not allowed.
	• Modifying the endtime changes the duration of the reservation as well (and vice versa). An endtime *cannot* be placed before the starttime or before the current time.
	• Duration cannot be negative.
	• The += and -= operators operate on the time of the reservation (`starttime+=5` adds five seconds to the current reservation starttime), while + and - operate on the current time (`starttime+5` sets the starttime to five seconds from now).
	• If the starttime or endtime specified is before the current time without a date specified, it is set to the next time that fits the command. To force the date, add the date as well. For the following examples, assume that the current time is 9:00 a.m. on March 1, 2007.
	`> mrsvctl -m starttime=8:00:00_3/1/07 system.1`
	Moab moves `system.1`'s starttime to 8:00 a.m., March 1.
	`> mrsvctl -m starttime=8:00:00 system.1`
	Moab moves `system.1`'s starttime to 8:00 a.m., March 2.
	`> mrsvctl -m endtime=7:00:00 system.1`
	Moab moves `system.1`'s endtime to 7:00 a.m., March 3. This happens because the endtime must also be after the starttime, so Moab continues searching until it has found a valid time that is in the future and after the starttime.
	`> mrsvctl -m endtime=7:00:00_3/2/07 system.1`
	Moab will return an error because the endtime cannot be before the starttime.

-n	
Name	NAME
Format	*<STRING>*

-n	
Description	Name for new reservation. If no name is specified, the reservation name is set to first name listed in ACL or `SYSTEM` if no ACL is specified. Reservation names may not contain whitespace.
Example	`mrsvctl -c -h node01 -n John` *Moab will create a reservation on node01 with the name `John`.*

-o	
Name	OWNER
Format	*<CREDTYPE>:<CREDID>*
Description	Specifies the owner of a reservation. See Reservation Ownership for more information.
Example	`mrsvctl -c -h node01 -o USER:user1` *Moab creates a reservation on `node01` owned by `user1`.*

-p	
Name	PARTITION
Format	*<STRING>*
Description	Only allocate resources from the specified partition
Example	`mrsvctl -c -p switchB -t 14` *Moab will allocate 14 tasks from the `switchB` partition.*

-P	
Name	PROFILE
Format	*<STRING>*
Description	Indicates the reservation profile to load when creating this reservation
Example	`mrsvctl -c -P testing2 -t 14` *Moab will allocate 14 tasks to a reservation defined by the `testing2` reservation profile.*

-q	
Name	QUERY
Format	*<RSV_ID>* — The `-r` option accepts x: node regular expressions and r: node range expressions (asterisks (*) are supported wildcards as well).
Description	Get diagnostic information or list all completed reservations. The command gathers information from the Moab cache which prevents it from interrupting the scheduler, but the `--blocking` option can be used to bypass the cache and interrupt the scheduler.

Example	`mrsvctl -q ALL`
	Moab will query reservations.
	
	`mrsvctl -q system.1`
	Moab will query the reservation `system.1`.

-r	
Name	RELEASE
Format	*<RSV_ID>* — The `-r` option accepts x: node regular expressions and r: node range expressions (asterisks (`*`) are supported wildcards as well).

-r	
Description	Releases the specified reservation.

Example	`> mrsvctl -r system.1`
	Moab will release reservation `system.1`.
	
	`> mrsvctl -r -g idle`
	Moab will release all idle job reservations.

-R	
Name	RESOURCES
Format	*<tid>* or *<RES>*=*<VAL>*[{,|+|;}*<RES>*=*<VAL>*]... Where *<RES>* is one of the following: PROCS, MEM, DISK, SWAP, GRES
Default	PROCS=-1
Description	Specifies the resources to be reserved per task. (`-1` indicates all resources on node) For GRES resources, *<VAL>* is specified in the format *<GRESNAME>*[:*<COUNT>*]
Example	`> mrsvctl -c -R MEM=100;PROCS=2 -t 2` *Moab will create a reservation for two tasks with the specified resources.*

-s	
Name	STARTTIME

Format	`[HH[:MM[:SS]]][_MO[/DD[/YY]]]`
	or
	`+[[[DD:]HH:]MM:]SS`

Default	[NOW]
Description	Absolute or relative time reservation will start. STARTTIME also supports an epoch timestamp.

Example	`> mrsvctl -c -t ALL -s 3:00:00_4/4/04`
	Moab will create a reservation on all system resources at 3:00 am on April 4, 2004
	
	`> mrsvctl -c -h node01 -s +5:00`
	Moab will create a reservation in 5 minutes on `node01`
	
	`> mrsvctl -m -s -=5:00 system.1`
	This will decrement the start time by 5 minutes.

-S	
Name	SET ATTRIBUTE
Format	*<ATTR>=<VALUE>* where *<ATTR>* is one of aaccount — Accountable account agroup — accountable group aqos — accountable QoS auser — accountable user reqarch — required architecture reqmemory — required node memory - in MB reqos — required operating system rsvaccesslist — comma- delimited list of reservations or reservation groups which can be accessed by this reservation request. Because each reservation can access all other reservations by default, you should make any reservation with a specified rsvaccesslist exclusive by setting the **-E** on page 165 flag. This setting gives the otherwise exclusive reservation access to reservations specified in the list.

-S	
Description	Specifies a reservation attribute will be used to create this reservation
Example	`> mrsvctl -c -h node01 -S aqos=high` *Moab will create a reservation on `node01` and will use the QOS `high` as the accountable credential*

-t	
Name	TASKS
Format	*<INTEGER>*[-*<INTEGER>*]
Description	Specifies the number of tasks to reserve. ALL indicates all resources available should be reserved. If the task value is set to ALL, Moab applies the reservation regardless of existing reservations and exclusive issues. If an integer is used, Moab only allocates accessible resources. If a range is specified Moab attempts to reserve the maximum number of tasks, or at least the minimum.

Example	`> mrsvctl -c -t ALL`
	Moab will create a reservation on all resources.
	`> mrsvctl -c -t 3`
	Moab will create a reservation for three tasks.
	`> mrsvctl -c -t 3-10 -E`
	Moab will attempt to reserve 10 tasks but will fail if it cannot get at least three.

-T	
Name	TRIGGER

-T	
Format	*<STRING>*
Description	Comma-delimited reservation trigger list following format described in the trigger format section of the reservation configuration overview. See **About object triggers** on page 655 for more information. To cancel a standing reservation with a trigger, the SRCFG parameter's attribute DEPTH must be set to *0*.
Example	`> mrsvctl -c -h node01 -T offset=200,etype=start,atype=exec,action=/opt/moab/tools/support.diag.pl` *Moab will create a reservation on `node01` and fire the script `/tmp/email.sh 200` seconds after it starts*

-V	
Name	VARIABLE
Format	*<name>*[=*<value>*][[;*<name>*[=*<value>*]]...]
Description	Semicolon-delimited list of variables that will be set when the reservation is created (see **About trigger variables** on page 684). Names with no values will simply be set to *TRUE*.
Example	`> mrsvctl -c -h node01 -V $T1=mac;var2=18.19` *Moab will create a reservation on `node01` and set `$T1` to `mac` and `var2` to `18.19`.* For information on modifying a variable on a reservation, see MODIFY.

-x	
Name	JOBLIST
Format	-x *<jobs to be excluded>*

-x	
Description	The -x flag, when used with -F ignjobrsv, lets users create reservations but exclude certain nodes that are running the listed jobs. The -F flag instructs mrsvctl to still consider nodes with current running jobs. The nodes are not listed directly.
Example	`> mrsvctl -c -t 5 -F ignjobrsv -x moab.5,moab.6` *Moab will create the reservation while assigning the nodes. Nodes running jobs moab5 and moab6 will not be assigned to the reservation.*

Parameters

RESERVATION ID	
Format	<STRING>
Description	The name of a reservation or a regular expression for several reservations.
Example	`system*` *Specifies all reservations starting with system.*

Resource Allocation Details

When allocating resources, the following rules apply:

- When specifying tasks, each task defaults to one full compute node unless otherwise specified using the -R specification
- When specifying tasks, the reservation will not be created unless all requested resources can be allocated. (This behavior can be changed by specifying -F besteffort)
- When specifying tasks or hosts, only nodes in an idle or running state will be considered. (This behavior can be changed by specifying -F ignstate)

Reservation Timeframe Modification

Moab supports dynamically modifying the timeframe of existing reservations. This can be accomplished using the mrsvctl -m flag. By default, Moab will perform advanced boundary and resource access to verify that the modification does not result in an invalid scheduler state. However, in certain circumstances administrators may wish to FORCE the modification in spite of any access violations. This can be done using the switch `mrsvctl -m --flags=force` which forces Moab to bypass any access verification and force the change through.

Extending a reservation by modifying the endtime

The following increases the endtime of a reservation using the += tag:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:35:57  1:11:35:57  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m endtime+=24:00:00 system.1
endtime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:35:22  2:11:35:22  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

The following increases the endtime of a reservation by setting the endtime to an absolute time:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:33:18  1:11:33:18  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m endtime=0_11/20 system.1
endtime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:33:05  2:11:33:05  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

Extending a reservation by modifying the duration

The following increases the duration of a reservation using the += tag:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:28:46  1:11:28:46  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m duration+=24:00:00 system.1
duration for rsv 'system.1' changed
>$ showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:28:42  2:11:28:42  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

The following increases the duration of a reservation by setting the duration to an absolute time:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:26:41  1:11:26:41  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m duration=48:00:00 system.1
duration for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:26:33  2:11:26:33  2:00:00:00    1/2    Sat Nov 18
```

```
00:00:00
1 reservation located
```

Shortening a reservation by modifying the endtime

The following modifies the endtime of a reservation using the -= tag:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:15:51  2:11:15:51  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m endtime-=24:00:00 system.1
endtime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:15:48  1:11:15:48  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

The following modifies the endtime of a reservation by setting the endtime to an absolute time:

```
$ showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:14:00  2:11:14:00  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m endtime=0_11/19 system.1
endtime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:13:48  1:11:13:48  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

Shortening a reservation by modifying the duration

The following modifies the duration of a reservation using the -= tag:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:12:20  2:11:12:20  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m duration-=24:00:00 system.1
duration for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:12:07  1:11:12:07  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

The following modifies the duration of a reservation by setting the duration to an absolute time:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:10:57  2:11:10:57  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m duration=24:00:00 system.1
```

```
duration for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:10:50  1:11:10:50  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

Modifying the starttime of a reservation

The following increases the starttime of a reservation using the += tag:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:08:30  2:11:08:30  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m starttime+=24:00:00 system.1
starttime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -  1:11:08:22  3:11:08:22  2:00:00:00    1/2    Sun Nov 19
00:00:00
1 reservation located
```

The following decreases the starttime of a reservation using the -= tag:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:07:04  2:11:07:04  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m starttime-=24:00:00 system.1
starttime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -   -12:53:04  1:11:06:56  2:00:00:00    1/2    Fri Nov 17
00:00:00
1 reservation located
```

The following modifies the starttime of a reservation using an absolute time:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:05:31  2:11:05:31  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m starttime=0_11/19 system.1
starttime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -  1:11:05:18  3:11:05:18  2:00:00:00    1/2    Sun Nov 19
00:00:00
1 reservation located
```

The following modifies the starttime of a reservation using an absolute time:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:04:04  2:11:04:04  2:00:00:00    1/2    Sat Nov 18
00:00:00
```

```
1 reservation located
$> mrsvctl -m starttime=0_11/17 system.1
starttime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -   -12:56:02  1:11:03:58  2:00:00:00    1/2    Fri Nov 17
00:00:00
1 reservation located
```

Examples

- **Basic Reservation** on page 182
- **System Maintenance Reservation** on page 182
- **Explicit Task Description** on page 182
- **Dynamic Reservation Modification** on page 182
- **Reservation Modification** on page 182
- **Allocating Reserved Resources** on page 183
- **Modifying an Existing Reservation** on page 183

Example 4-26: Basic Reservation

Reserve two nodes for use by users `john` and `mary` for a period of 8 hours starting in 24 hours

```
> mrsvctl -c -a USER=john,USER=mary -starttime +24:00:00 -duration 8:00:00 -t 2
reservation 'system.1' created
```

Example 4-27: System Maintenance Reservation

Schedule a system wide reservation to allow a system maintenance on Jun 20, 8:00 AM until Jun 22, 5:00 PM.

```
% mrsvctl -c -s 8:00:00_06/20 -e 17:00:00_06/22 -h ALL
reservation 'system.1' created
```

Example 4-28: Explicit Task Description

Reserve one processor and `512` MB of memory on nodes `node003` through node `node006` for members of the group `staff` and jobs in the `interactive` class

```
> mrsvctl -c -R PROCS=1,MEM=512 -a GROUP=staff,CLASS=interactive -h 'node00[3-6]'
reservation 'system.1' created
```

Example 4-29: Dynamic Reservation Modification

Modify reservation `john.1` to start in 2 hours, run for 2 hours, and include `node02` in the host list.

```
> mrsvctl -m starttime=+2:00:00,duration=2:00:00,HostExp+=node02
Note:  hosts added to rsv system.3
```

Example 4-30: Reservation Modification

Remove user `John`'s access to reservation `system.1`

```
> mrsvctl -m -a USER=John system.1 --flags=unset
successfully changed ACL for rsv system.1
```

Example 4-31: Allocating Reserved Resources

Allocate resources for group `dev` which are exclusive except for resources found within reservations `myrinet.3` or `john.6`

```
> mrsvctl -c -E -a group=dev,rsv=myrinet.3,rsv=john.6 -h 'node00[3-6]'
reservation 'dev.14' created
```

Create exclusive `network` reservation on racks 3 and 4

```
> mrsvctl -c -E -a group=ops -g network -f rack3 -h ALL
reservation 'ops.1' created
> mrsvctl -c -E -a group=ops -g network -f rack4 -h ALL
reservation 'ops.2' created
```

Allocate `64` nodes for 2 hours to new reservation and grant access to reservation `system.3` and all reservations in the reservation group `network`

```
> mrsvctl -c -E -d 2:00:00 -a group=dev -t 64 -S rsvaccesslist=system.3,network
reservation 'system.23' created
```

Allocate `4` nodes for 1 hour to new reservation and grant access to idle job reservations

```
> mrsvctl -c -E -d 1:00:00 -t 4 -S rsvaccesslist=idle
reservation 'system.24' created
```

Example 4-32: Modifying an Existing Reservation

Remove user `john` from reservation ACL

```
> mrsvctl -m -a USER=john system.1 --flags=unset
successfully changed ACL for rsv system.1
```

Change reservation group

```
> mrsvctl -m RSVGROUP=network ops.4
successfully changed RSVGROUP for rsv ops.4
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- Admin Reservation Overview
- showres
- mdiag -r
- mshow -a command to identify available resources
- job to rsv binding

mschedctl

Synopsis

mschedctl -A '<MESSAGE>'

mschedctl -c message *messagestring* [-o *type*:*val*]

mschedctl -c trigger *triggerid* -o *type*:*val*

mschedctl -d trigger:*triggerid*

mschedctl -d message:*index*

mschedctl -f {all|fairshare|usage}

mschedctl -k

mschedctl -l {config|gmetric|gres|message|opsys|trigger|trans} [--flags=verbose] [--xml]

mschedctl -L [*LOGLEVEL*]

mschedctl -m config *string* [-e] [--flags=persistent]

mschedctl -m trigger *triggerid attr*=*val*[,*attr*=*val*...]

mschedctl -q mschedctl -q pactions --xml

mschedctl -p

mschedctl -r [*resumetime*]

mschedctl -R

mschedctl -s [*STOPITERATION*]

mschedctl -S [*STEPITERATION*]

Overview

The mschedctl command controls various aspects of scheduling behavior. It is used to manage scheduling activity, shutdown the scheduler, and create resource trace files. It can also evaluate, modify, and create parameters, triggers, and messages.

With many flags, the `--msg=<MSG>` option can be specified to annotate the action in the event log.

Format

-A - ANNOTATE	
Format	*<STRING>*
Description	Report the specified parameter modification to the event log and annotate it with the specified message. The RECORDEVENTLIST parameter must be set in order for this to work.

-A - ANNOTATE

Example

```
mschedctl -A 'increase logging' -m 'LOGLEVEL 6'
```

Adjust the LOGLEVEL parameter and record an associated message.

-c - CREATE

Format

One of:

- **message** <*STRING*> [-o <*TYPE*>:<*VAL*>]
- **trigger**<TRIGSPEC> -o <*OBJECTTYPE*>:<*OBJECTID*>
- **gevent** -n <*NAME*> [-m <*message*>]

  where <*ATTR*> is one of account, duration, ID, messages, profile, reqresources, resources, rsvprofile, starttime, user, or variables

Description

Create a message, trigger, or gevent and attach it to the specified object. To create a trigger on a default object, use the Moab configuration file (`moab.cfg`) rather than the mschedctl command.

Example

```
mschedctl -c message tell the admin to be nice
```

Create a message on the system table.

```
mschedctl -c trigger EType=start,AType=exec,Action="/tmp/email $OWNER $TIME" -o
rsv:system.1
```

Create a trigger linked to `system.1`.

Creating triggers on default objects via `mschedctl -c trigger` does not propagate the triggers to individual objects. To propagate triggers to all objects, the triggers must be created within the `moab.cfg` file; for example: `NODECFG[DEFAULT]TRIGGER`.

```
mschedctl -c gevent -n diskfailure -m "node=n4"
```

Create a gevent indicating a disk failure on the node labeled `n4`.

-d - DESTROY

Format	One of: • **trigger**:*<TRIGID>* • **message**:*<INDEX>*
Description	Delete a trigger or message.

Example	`mschedctl -d trigger:3`
	Delete trigger 3.
	
	`mschedctl -d message:5`
	Delete message with index 5.

-f - FLUSH

Format	{all|fairshare|usage}
Description	Reset all internally-stored Moab Scheduler statistics to the initial start-up state as of the time the command was executed.
Example	`mschedctl -f usage` *Flush usage statistics.*

-k - KILL

Description	Stop scheduling and exit the scheduler
Example	`mschedctl -k` *Kill the scheduler.*

-l - LIST

Format	{config |gmetric| gres | message |opsys| trans | trigger} [--flags=verbose] [--xml] Using the `--xml` argument with the trans option returns XML that states if the queried TID is valid or not.
Default	config
Description	List the generic metrics, generic resources, operating systems, scheduler configuration, system messages, triggers, or transactions.

Example	`mschedctl -l config`
	List system parameters.
	`mschedctl -l gmetric`
	List all configured generic metrics.
	`mschedctl -l gres`
	List all configured generic resources.
	`mschedctl -l message`
	List all system messages.
	`mschedctl -l opsys`
	List all operating systems.
	`mschedctl -l trans 1`
	List transaction id 1.
	`mschedctl -l trigger`
	List triggers.

-L - LOG

Format	*<INTEGER>*

-L - LOG

Default	7
Description	Create a temporary log file with the specified loglevel.
Example	`mschedctl -L 7` *Create temporary log file with naming convention <logfile>.YYYYMMDDHHMMSS.*

-m - MODIFY

Format	One of: • **config** [*<STRING>*] [`-e`] [`--flags=pers`] *<STRING>* is any string which would be acceptable in `moab.cfg` ◦ If no string is specified, *<STRING>* is read from STDIN. ◦ If `-e` is specified, the configuration string will be evaluated for correctness but no configuration changes will take place. Any issues with the provided string will be reported to STDERR. ◦ If `--flags=persistent` is specified, the Moab configuration files (`moab.cfg` and `moab.dat`) are modified. • **trigger**:*<TRIGID> <ATTR>=<VAL>* where *<ATTR>* is one of action, atype, etype, iscomplete, oid, otype, offset, or threshold
Description	Modify a system parameter or trigger.

Example	`mschedctl -m config LOGLEVEL 9`
	Change the system loglevel to 9.
	
	`mschedctl -m trigger:2 AType=exec,Offset=200,OID=system.1`
	Change aspects of trigger 2.

-p - PAUSE

Description	Disable scheduling but allow the scheduler to update its cluster and workload state information.
Example	`mschedctl -p`

-q QUERY PENDING ACTIONS

Default	`mschedctl -q pactions --xml`
Description	A way to view pending actions. Only an XML request is valid. Pending actions can be VMs or system jobs.
Example	`mschedctl -q pactions --xml`

-R - RECYCLE

Description	Recycle scheduler immediately (shut it down and restart it using the original execution environment and command line arguments).

Example	`mschedctl -R`
	Recycle scheduler immediately.
	
	To restart Moab with its last known scheduler state, use:
	`mschedctl -R savestate`

-r - RESUME

Format	mschedctl -r [[HH:[MM:]]SS]
Default	0
Description	Resume scheduling in the specified amount of time (or immediately if none is specified).

-r - RESUME	
Example	`mschedctl -r` *Resume scheduling immediately.*

-s - STOP	
Format	*<INTEGER>*
Default	0
Description	Suspend/stop scheduling at specified iteration (or at the end of the current iteration if none is specified). If the letter I follows *<ITERATION>*, Moab will not process client requests until this iteration is reached.
Example	`mschedctl -s 100I` *Stop scheduling at iteration 100 and ignore all client requests until then.*

-S - STEP	
Format	*<INTEGER>*
Default	0
Description	Step the specified number of iterations (or to the next iteration if none is specified) and suspend scheduling If the letter I follows *<ITERATION>*, Moab will not process client requests until this iteration is reached.
Example	`mschedctl -S` *Step to the next iteration and stop scheduling.*

Examples

Example 4-33: Shutting down the Scheduler

```
mschedctl -k
```

```
scheduler will be shutdown immediately
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes

mshow

Synopsis

mshow [-a] [-q *jobqueue*=active]

Overview

The mshow command displays various diagnostic messages about the system and job queues.

Arguments

Flag	Description
-a	AVAILABLE RESOURCES
-q [<QUEUENAME>]	Displays the job queues.

Format

AVAILABLE RESOURCES	
Format	Can be combined with `--flags=[tid|verbose|future]` `--format=xml` and/or `-w`
Description	Display available resources.
Example	`> mshow -a -w user=john --flags=tid --format=xml` *Show resources available to john in XML format with a transaction id. See mshow -a for details.*

JOB QUEUE	
Format	*<QUEUENAME>*, where the queue name is one of: active, eligible, or blocked. Job queue names can be delimited by a comma to display multiple queues. If no job queue name is specified, mshow displays all job queues.

JOB QUEUE	
Description	Displays the job queues. If a job queue name is specified, mshow shows only that job queue.

Example	`> mshow -q active,blocked`
	`  [Displays all jobs in the active and blocked queues]`
	`...`

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mshow -a command to show available resources

mshow -a

Synopsis

mshow -a [-i] [-o] [-T] [-w *where*] [-x] [--xml]

Overview

The mshow -a command allows for querying of available system resources.

Arguments

[-i]	INTERSECTION
[-o]	NO AGGREGATE
[-T]	TIMELOCK
[-w]	WHERE
[-x]	EXCLUSIVE

Table 4-1: Argument Format

--flags	
Name	Flags
Format	--flags=[future | policy | tid | summary | verbose]

--flags	
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. **verbose** will return diagnostic information.
Example	`> mshow -a -w user=john --flags=tid --xml` *Show resources available to john in XML format with a transaction ID.*

--xml	
Name	XML
Format	--xml
Description	Report results in XML format.
Example	`> mshow -a -w user=john --flags=tid --xml` *Show resources available to `john` in XML format with a transaction ID.*

-i	
Name	INTERSECTION
Description	Specifies that an intersection should be performed during an mshow -a command with multiple requirements.

-o	
Name	NO AGGREGATE
Description	Specifies that the results of the command mshow -a with multiple requirements should not be aggregated together.

-T	
Name	TIMELOCK
Description	Specifies that the multiple requirements of an mshow -a command should be timelocked.

Example	`> mshow -a -w minprocs=1,os=linux,duration=1:00:00 \`
	`  -w minprocs=1,os=aix,duration=10:00 \`
	`  --flags=tid,future -x -T`

-w	
Name	WHERE
Format	Comma delimited list of *<ATTR>=<VAL>* pairs: *<ATTR>=<VAL>* [,*<ATTR>=<VAL>*]... If any of the *<ATTR>=<VAL>* pairs contains a sub-list that is also comma delimited, the entire `-w` string must be wrapped in single quotations with the sub-list expression wrapped in double quotations. See the example below. Attributes are listed below in table 2.
Description	Add a Where clause to the current command (currently supports up to six co-allocation clauses).

Example	`> mshow -a -w minprocs=2,duration=1:00:00 -w nodemem=512,duration=1:00:00`
	Moab returns a list of all nodes with at least `2` processors and one hour duration or with a memory of `512` and a duration of one hour.
	`> mshow -a -w nodefeature=\!vmware:gpfs --flags=future`
	Moab returns a list of all nodes that do not contain the `vmware` feature but that do contain the `gpfs` feature.
	`> mshow -a -w 'duration=INFINITY,"excludehostlist=n01,n12,n23"'`
	Moab returns a list of all nodes with a duration of INFINITY, except for nodes named n01, n12, and n23.
	*Note the use of single quotations containing the entire `-w` string and the use of double quotations containing the **excludehostlist** attribute.*

-x	
Name	EXCLUSIVE
Description	Specifies that the multiple requirements of an mshow -a command should be exclusive (i.e. each node may only be allocated to a single requirement)
Example	`> mshow -a -w minprocs=1,os=linux -w minprocs=1,os=aix --flags=tid -x`

Table 4-2: Request Attributes

Name	Description
account	The account credential of the requestor
acl	ACL to attach to the reservation This ACL must be enclosed in quotation marks. For example: `$ mshow -a ... -w acl=\"user=john\" ...`
arch	Select only nodes with the specified architecture
cal	Select resources subject to the constraints of the specified global calendar
class	The class credential of the requestor
coalloc	The co-allocation group of the specific Where request (can be any string but must match co-allocation group of at least one other Where request) The number of tasks requested in each Where request must be equal whether this task count is specified via `minprocs`, `mintasks`, or `gres`.
count	The number of profiles to apply to the resource request
displaymode	Possible value is `future`. (Example: `displaymode=future`). Constrains how results are presented; setting `future` evaluates which resources are available now and which resources will be available in the future that match the requested attributes.
duration	The duration for which the resources will be required in format `[[[DD:]HH:]MM:]SS`

Name	Description
excludehostlist	Do not select any nodes from the given list. The list must be comma delimited. `> mshow -a -w 'duration=INFINITY,"excludehostlist=n01,n12,n23"'` *Moab returns a list of all nodes with a duration of INFINITY, except for nodes named n01, n12, and n23.* *Note the use of single quotations to contain the entire -w string, and the use of double quotations containing the **excludehostlist** attribute.*
gres	Select only nodes which possess the specified generic resource
group	The group credential of the requestor
hostlist	Select only the specified resources. The list must be comma delimited. `> mshow -a -w 'duration=INFINITY,"hostlist=n01,n12,n23"'` *Moab returns a list of nodes from the selected hostlist that have a duration of INFINITY.* *Note the use of single quotations to contain the entire -w string, and the use of double quotations containing the **hostlist** attribute.*
job	Use the resource, duration, and credential information for the job specified as a resource request template
jobfeature	Select only resources which would allow access to jobs with the specified job features
jobflags	Select only resources which would allow access to jobs with the specified job flags. The `jobflags` attribute accepts a colon delimited list of multiple flags.
minnodes	Return only results with at least the number of nodes specified. If used with TID's, return only solutions with exactly minnodes nodes available
minprocs	Return only results with at least the number of processors specified. If used with TID's, return only solutions with exactly minprocs processors available
mintasks	**FORMAT**: *<TASKCOUNT>*[@*<RESTYPE>*:*<COUNT>*[+*<RESTYPE>*:*<COUNT>*]...] where *<RESTYPE>* is one of `procs`, `mem`, `disk`, or `swap`. Return only results with at least the number of tasks specified. If used with TID's, return only solutions with exactly mintasks available
nodedisk	Select only nodes with at least nodedisk MB of local disk configured

Name	Description
nodefeature	Select only nodes with all specified features present and nodes without all `\!` specified features using format `[\!]<feature>[:[\!]<feature>]...` You must set the future flag when specifying node features.
nodemem	Select only nodes with at least `nodemem` MB of memory configured
offset	Select only resources which can be co-allocated with the specified time offset where offset is specified in the format `[[[DD:]HH:]MM:]SS`
os	Select only nodes with have, or can be provisioned to have, the specified operating system
partition	The partition in which the resources must be located
policylevel	Enable policy enforcement at the specified policy constraint level
qos	The QoS credential of the requestor
rsvprofile	Use the specified profile if committing a resulting transaction id directly to a reservation
starttime	Constrain the timeframe for the returned results by specifying one or more ranges using the format *<STIME>*[-*<ENDTIME>*][;*<STIME>*[-*<ENDTIME>*]] where each time is specified in the format in absolute, relative, or epoch time format (`[HH[:MM[:SS]]][_MO[/DD[/YY]]]` or `+[[[DD:]HH:]MM:]SS` or *<EPOCHTIME>*). The starttime specified is not the exact time at which the returned range must start, but is rather the earliest possible time the range may start.
taskmem	Require `taskmem` MB of memory per task located
tpn	Require exactly `tpn` tasks per node on all discovered resources
user	The user credential of the requestor
var	Use associated variables in generating per transaction charging quotes
variables	Takes a string of the format `variables='var[=attr]'[;'var[=attr]'` and passes the variables onto the reservation when used in conjunction with `--flags=tid` and `mrsvctl -c -R <tid>`.

Name	Description
vmusage	Possible value is `vmcreate`. Moab will find resources for the job assuming it is a vmcreate job, and if `os` is also specified, Moab will look for a hypervisor capable of running a VM with the requested OS.

Usage Notes

The mshow -a command allows for querying of available system resources. When combined with the `--flags=tid` option these available resources can then be placed into a packaged reservation (using mrsvctl -c -R). This allows system administrators to grab and reserve available resources for whatever reason, without conflicting with jobs or reservations that may be holding certain resources.

There are a few restrictions on which *<ATTR>* from the -w command can be placed in the same req: `minprocs`, `minnodes`, and `gres` are all mutually exclusive, only one may be used per `-w` request.

The allocation of available nodes will follow the global NODEALLOCATIONPOLICY.

When the '-o' flag is not used, multi-request results will be aggregated. This aggregation will negate the use of offsets and request-specific starttimes.

The config parameter RESOURCEQUERYDEPTH controls the maximum number of options that will be returned in response to a resource query.

Examples

Example 4-34: Basic Compute Node Query and Reservation

```
> mshow -a -w duration=10:00:00,minprocs=1,os=AIX53,jobfeature=shared --
flags=tid,future

Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1      10:00:00      00:00:00  13:28:09_04/27  TID=4  ReqID=0
ALL               1      1      10:00:00      10:00:00  17:14:48_04/28  TID=5  ReqID=0
ALL               1      1      10:00:00      20:00:00  21:01:27_04/29  TID=6  ReqID=0
> mrsvctl -c -R 4
Note:  reservation system.2 created
```

Example 4-35: Mixed Processor and License Query

Select one node with 4 processors and 1 matlab license where the matlab license is only available for the last hour of the reservation. Also, select 16 additional processors which are available during the same timeframe but which can be located anywhere in the cluster. Group the resulting transactions together using transaction dependencies so only the first transaction needs to be committed to reserve all associated resources.

```
> mshow -a -i -o -x -w mintasks=1@PROCS:4,duration=10:00:00,coalloc=a \
                    -w gres=matlab,offset=9:00:00,duration=1:00:00,coalloc=a \
                    -w minprocs=16,duration=10:00:00 --flags=tid,future,summary
Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1      10:00:00      00:00:00  13:28:09_04/27  TID=4  ReqID=0
ALL               1      1      10:00:00      10:00:00  17:14:48_04/28  TID=5  ReqID=0
ALL               1      1      10:00:00      20:00:00  21:01:27_04/29  TID=6  ReqID=0
```

```
> mrsvctl -c -R 4

Note:  reservation system.2 created
Note:  reservation system.3 created
Note:  reservation system.4 created
```

Example 4-36: Request for Generic Resources

Query for a generic resource on a specific host (no processors, only a generic resource).

```
> mshow -a -i -x -o -w gres=dvd,duration=10:00,hostlist=node03 --flags=tid,future
Partition     Tasks  Nodes    StartOffset      Duration       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1      00:00:00      00:10:00  11:33:25_07/27  TID=16
ReqID=0
ALL               1      1      00:10:00      00:10:00  11:43:25_07/27  TID=17
ReqID=0
ALL               1      1      00:20:00      00:10:00  11:53:25_07/27  TID=18
ReqID=0
> mrsvctl -c -R 16
Note:  reservation system.6 created
> mdiag -r system.6
Diagnosing Reservations
RsvID                     Type Par   StartTime     EndTime     Duration Node Task
Proc
-----                     ---- ---   ---------     -------     -------- ---- ---- ---
-
system.6                  User loc   -00:01:02    00:08:35     00:09:37    1    1
0
    Flags: ISCLOSED
    ACL:   RSV==system.6=
    CL:    RSV==system.6
    Accounting Creds:  User:test
    Task Resources: dvd: 1
    Attributes (HostExp='^node03$')
    Rsv-Group: system.6
```

Example 4-37: Allocation of Shared Resources

This example walks through a relatively complicated example in which a set of resources can be reserved to be allocated for shared requests. In the example below, the first mshow query looks for resources within an existing shared reservation. In the example, this first query fails because there is now existing reservation. The second query looks for resources within an existing shared reservation. In the example, this first query fails because there is now existing reservation. The second mshow request asks for resources outside of a shared reservation and finds the desired resources. These resources are then reserved as a shared pool. The third mshow request again asks for resources inside of a shared reservation and this time finds the desired resources.

```
> mshow -a -w duration=10:00:00,minprocs=1,os=AIX53,jobflags=ADVRES,jobfeature=shared
--flags=tid

Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
> mshow -a -w duration=100:00:00,minprocs=1,os=AIX53,jobfeature=shared --flags=tid
Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1     100:00:00      00:00:00  13:20:23_04/27  TID=1  ReqID=0
> mrsvctl -c -R 1
Note:  reservation system.1 created
```

```
> mshow -a -w duration=10:00:00,minprocs=1,os=AIX53,jobflags=ADVRES,jobfeature=shared
--flags=tid
Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1      10:00:00      00:00:00  13:20:36_04/27  TID=2  ReqID=0
> mrsvctl -c -R 2
Note:  reservation system.2 created
```

Example 4-38: Full Resource Query in XML Format

The following command will report information on all available resources which meet at least the minimum specified processor and walltime constraints and which are available to the specified user. The results will be reported in XML to allow for easy system processing.

```
> mshow -a -w class=grid,minprocs=8,duration=20:00 --format=xml --flags=future,verbose

<Data>
  <Object>cluster</Object>
  <job User="john" time="1162407604"></job>
  <par Name="template">
    <range duration="Duration" nodecount="Nodes" proccount="Procs"
starttime="StartTime"></range>
    </par>
  <par Name="ALL" feasibleNodeCount="131" feasibleTaskCount="163">
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
041:1,opt-042:1,x86-001:1,P690-001:1,P690-021:1,P690-022:1"
        index="0" nodecount="10" proccount="8" reqid="0"
starttime="1162407604"></range>
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
039:1,opt-041:1,opt-042:1,x86-001:1,P690-001:1,P690-021:1,P690-022:1"
        index="0" nodecount="11" proccount="8"reqid="0"
starttime="1162411204"></range>
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
039:1,opt-041:1,opt-042:1,x86-001:1,x86-002:1,x86-004:1,
       x86-006:1,x86-013:1,x86-014:1,x86-015:1,x86-016:1,x86-037:1,P690-001:1,P690-
021:1,P690-022:1"
        index="0" nodecount="19" proccount="8" reqid="0"
starttime="1162425519"></range>
    </par>
  <par Name="SharedMem">
    <range duration="1200" hostlist="P690-001:1,P690-002:1,P690-003:1,P690-004:1,P690-
005:1,P690-006:1,P690-007:1,P690-008:1,P690-009:1,
        P690-010:1,P690-011:1,P690-012:1,P690-013:1,P690-014:1,P690-015:1,P690-
016:1,P690-017:1,P690-018:1,P690-019:1,P690-020:1,P690-021:1,
        P690-022:1,P690-023:1,P690-024:1,P690-025:1,P690-026:1,P690-027:1,P690-
028:1,P690-029:1,P690-030:1,P690-031:1,P690-032:1"
        index="0" nodecount="32" proccount="8" reqid="0"
starttime="1163122507"></range>
    </par>
  <par Name="64Bit">
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
039:1,opt-041:1,opt-042:1"
        index="0" nodecount="7" proccount="8" reqid="0"
starttime="1162411204"></range>
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
039:1,opt-041:1,opt-042:1,opt-043:1,opt-044:1,opt-045:1,
        opt-046:1,opt-047:1,opt-048:1,opt-049:1,opt-050:1"
        index="0" nodecount="15" proccount="8" reqid="0"
starttime="1162428996"></range>
    <range duration="1200" hostlist="opt-001:1,opt-006:1,opt-007:2,opt-008:2,opt-
009:2,opt-010:2,opt-011:2,opt-012:2,opt-013:2,opt-014:2,
```

```
        opt-015:2,opt-016:2,opt-017:2,opt-018:2,opt-019:2,opt-020:2,opt-021:2,opt-
022:2,opt-023:2,opt-024:2,opt-025:1,opt-027:2,opt-039:1,
        opt-041:1,opt-042:1,opt-043:1,opt-044:1,opt-045:1,opt-046:1,opt-047:1,opt-
048:1,opt-049:1,opt-050:1"
        index="0" nodecount="33" proccount="8" reqid="0"
starttime="1162876617"></range>
    </par>
  <par Name="32Bit">
    <range duration="1200" hostlist="x86-001:1,x86-002:1,x86-004:1,x86-006:1,x86-
013:1,x86-014:1,x86-015:1,x86-016:1,x86-037:1"
        index="0" nodecount="9" proccount="8" reqid="0"
starttime="1162425519"></range>
    <range duration="1200" hostlist="x86-001:1,x86-002:1,x86-004:1,x86-006:1,x86-
013:1,x86-014:1,x86-015:1,x86-016:1,x86-037:1,x86-042:1,x86-043:1"
        index="0" nodecount="11" proccount="8" reqid="0"
starttime="1162956803"></range>
    <range duration="1200" hostlist="x86-001:1,x86-002:1,x86-004:1,x86-006:1,x86-
013:1,x86-014:1,x86-015:1,x86-016:1,x86-027:1,x86-028:1,
        x86-029:1,x86-030:1,x86-037:1,x86-041:1,x86-042:1,x86-043:1,x86-046:1,x86-
047:1,x86-048:1,x86-049:1"
        index="0" nodecount="20" proccount="8" reqid="0"
starttime="1163053393"></range>
    </par>
  </Data>
```

This command reports the original query, and the timeframe, resource size, and hostlist associated with each possible time slot.

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mshow in a hosting environment

mshow -a

Basic Current and Future Requests

The mshow command can report information on many aspects of the scheduling environment. To request information on available resources, the `-a` flag should be used. By default, the mshow command resource availability query only reports resources that are immediately available. To request information on specific resources, the type of resources required can be specified using the `-w` flag as in the following example:

```
> mshow -a -w taskmem=1500,duration=600
...
```

To view current and future resource availability, the `future` flag should be set as in the following example:

```
> mshow -a -w taskmem=1500,duration=600 --flags=future
...
```

Co-allocation Resources Queries

In many cases, a particular request will need simultaneous access to resources of different types. The mshow command supports a co-allocation request specified by using multiple `-w` arguments. For example, to request 16 nodes with feature `fastcpu` and 2 nodes with feature `fastio`, the following request might be used:

```
> mshow -a -w minprocs=16,duration=1:00:00,nodefeature=fastcpu -w
minprocs=2,nodefeature=fastio,duration=1:00:00 --flags=future
Partition     Procs  Nodes    StartOffset      Duration       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL              16      8      00:00:00       1:00:00  13:00:18_08/25  ReqID=0
ALL               2      1      00:00:00       1:00:00  13:00:18_08/25  ReqID=1
```

The mshow -a documentation contains a list of the different resources that may be queried as well as examples on using mshow.

Using Transaction IDs

By default, the mshow command reports simply when and where the requested resources are available. However, when the `tid` flag is specified, the mshow command returns both resource availability information and a handle to these resources called a Transaction ID as in the following example:

```
> mshow -a -w minprocs=16,nodefeature=fastcpu,duration=2:00:00 --flags=future,tid
Partition     Procs  Nodes    StartOffset      Duration       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL              16     16      00:00:00       2:00:00  13:00:18_08/25  TID=26 ReqID=0
```

In the preceding example, the returned transaction id (TID) may then be used to reserve the available resources using the mrsvctl -c -R command:

```
> mrsvctl -c -R 26
reservation system.1 successfully created
```

Any TID can be printed out using the mschedctl -l trans command:

```
Code example (replace with your own content)
```

> mschedctl -l trans 26 TID[26] A1='node01' A2='600' A3='1093465728' A4='ADVRES' A5='fastio'

Where `A1` is the host list, `A2` is the duration, `A3` is the starttime, `A4` are any flags, and `A5` are any features.

Using Reservation Profiles

Reservation profiles (RSVPROFILE) stand as templates against which reservations can be created. They can contain a host list, startime, endtime, duration, access-control list, flags, triggers, variables, and most other attributes of an Administrative Reservation. The following example illustrates how to create a reservation with the exact same trigger-set.

```
-----
# moab.cfg
-----
RSVPROFILE[test1] TRIGGER=Sets=$Var1.$Var2.$Var3.!Net,EType=start,AType=exec,
        Action=/tmp/host/triggers/Net.sh,
```

```
        Timeout=1:00:00
RSVPROFILE[test1]                                   TRIGGER=Requires=$Var1.$Var2.$Var3,
        Sets=$Var4.$Var5,EType=start,
        AType=exec,Action=/tmp/host/triggers/
        FS.sh+$Var1:$Var2:$Var3,Timeout=20:00
RSVPROFILE[test1]
TRIGGER=Requires=$Var1.$Var2.$Var3.$Var4.$Var5,
        Sets=!NOOSinit.OSinit,Etype=start,
        AType=exec,
        Action=/tmp/host/triggers/
        OS.sh+$Var1:$Var2:$Var3:$Var4:$Var5
RSVPROFILE[test1]
TRIGGER=Requires=NOOSini,AType=cancel,EType=start
RSVPROFILE[test1]
TRIGGER=EType=start,Requires=OSinit,AType=exec,
        Action=/tmp/host/triggers/success.sh
...
-----
```

To create a reservation with this profile the mrsvctl -c -P command is used:

```
> mrsvctl -c -P test1

reservation system.1 successfully created
```

Using Reservation Groups

Reservation groups are a way for Moab to tie reservations together. When a reservation is created using multiple Transaction IDs, these transactions and their resulting reservations are tied together into one group.

```
> mrsvctl -c -R 34,35,36
reservation system.99 successfully created
reservation system.100 successfully created
reservation system.101 successfully created
```

In the preceding example, these three reservations would be tied together into a single group. The mdiag -r command can be used to see which group a reservation belongs to. The mrsvctl -q diag -g command can also be used to print out a specific group of reservations. The mrsvctl -c -g command can also be used to release a group of reservations.

Related topics

- mshow

msub

Synopsis

msub [-a *datetime*][-A *account*][-c interval][-C directive_prefix][-d *path*] [-e *path*][-E][-F][-h][-I][-j *join*][-k keep][-K][-l *resourcelist*][-m *mailoptions*] [-M user_list][-N name][-o *path*][-p *priority*][-q *destination*][-r] [-S *pathlist*][-t *jobarrays*][-u *userlist*][-v *variablelist*][-V] [-W *additionalattributes*][-x][-z][*script*]

Overview

msub allows users to submit jobs directly to Moab. When a job is submitted directly to a resource manager (such as TORQUE), it is constrained to run on only those nodes that the resource manager is directly monitoring. In many instances, a site may be controlling multiple resource managers. When a job is submitted to Moab rather than to a specific resource manager, it is not constrained as to what nodes it is executed on. msub can accept command line arguments (with the same syntax as qsub), job scripts (in either PBS or LoadLeveler syntax), or the SSS Job XML specification.

Moab must run as a root user in order for msub submissions to work. Workload submitted via msub when Moab is running as a non-root user fail immediately.

Submitted jobs can then be viewed and controlled via the mjobctl command.

Flags specified in the following table are not necessarily supported by all resource managers.

Access

When Moab is configured to run as root, any user may submit jobs via msub.

Flags

-a	
Name	Eligible Date
Format	[[[[CC]YY]MM]DD]hhmm[.SS]
Description	Declares the time after which the job is eligible for execution.
Example	`> msub -a 12041300 cmd.pbs` *Moab will not schedule the job until 1:00 pm on December 4, of the current year.*

-A	
Name	Account
Format	*<ACCOUNT NAME>*
Description	Defines the account associated with the job.

-A	
Example	`> msub -A research cmd.pbs` *Moab will associate this job with account research.*

-c	
Name	Checkpoint Interval
Format	[n|s|c|c=*<minutes>*]
Description	Checkpoint of the will occur at the specified interval. **n** — No Checkpoint is to be performed. **s** — Checkpointing is to be performed only when the server executing the job is shut down. **c** — Checkpoint is to be performed at the default minimum time for the server executing the job. **c=*<minutes>*** — Checkpoint is to be performed at an interval of minutes.
Example	`> msub -c c=12 cmd.pbs` *The job will be checkpointed every 12 minutes.*

-C	
Name	Directive Prefix
Format	'*<PREFIX NAME>*'
Default	First known prefix (#PBS, #@, #BSUB, #!, #MOAB, #MSUB)
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.

-C	

Example	`> msub -C '#MYPREFIX' cmd.pbs`
	`#MYPREFIX -l walltime=5:00:00 (in cmd.pbs)`
	Moab will use the #MYPREFIX directive specified in cmd.pbs, setting the wallclock limit to five hours.

-d	
Name	Execution Directory
Format	*<path>*
Default	Depends on the RM being used. If using TORQUE, the default is $HOME. If using SLURM, the default is the submission directory.
Description	Specifies which directory the job should execute in.
Example	`> msub -d /home/test/job12 cmd.pbs` *The job will begin execution in the `/home/test/job12` directory.*

-e	
Name	Error Path
Format	[*<hostname>*:]*<path>*
Default	`$SUBMISSIONDIR/$JOBNAME.e$JOBID`
Description	Defines the path to be used for the standard error stream of the batch job.
Example	`> msub -e test12/stderr.txt` *The STDERR stream of the job will be placed in the relative (to execution) directory specified.*

-E	
Name	Environment Variables
Description	Moab adds the following variables, if populated, to the job's environment: • MOAB_ACCOUNT — Account name. • MOAB_BATCH — Set if a batch job (non-interactive). • MOAB_CLASS — Class name. • MOAB_DEPEND — Job dependency string. • MOAB_GROUP — Group name. • MOAB_JOBARRAYINDEX —For a job in an array, the index of the job. • MOAB_JOBARRAYRANGE — For a system with job arrays, the range of all job arrays. • MOAB_JOBID — Job ID. • MOAB_JOBNAME — Job name. • MOAB_MACHINE — Name of the machine (i.e. Destination RM) that the job is running on. • MOAB_NODECOUNT — Number of nodes allocated to job. • MOAB_NODELIST — Comma-separated list of nodes (listed singly with no ppn info). • MOAB_PARTITION — Partition name the job is running in. • MOAB_PROCCOUNT — Number of processors allocated to job. • MOAB_QOS — QoS name. • MOAB_TASKMAP — Node list with procs per node listed. <nodename>.<procs> • MOAB_USER — User name. In SLURM environments, not all variables will be populated since the variables are added at submission (such as NODELIST). With TORQUE/PBS, the variables are added just before the job is started. This feature only works with SLURM and TORQUE/PBS.
Example:	`> msub -E mySim.cmd` *The job `mySim` will be submitted with extra environment variables.*

-F	
Name	Script Flags
Format	"\"*<STRING>*\""
Description	Specifies the flags TORQUE will pass to the job script at execution time. The `-F` flag is only compatible with TORQUE resource managers.

-F	
Example	`> msub -F "\"arg1 arg2\"" -1 nodes=1,walltime=60 files/job.sh` *TORQUE will pass parameters `arg1` and `arg2` to the `job.sh` script when the job executes.*

-h	
Name	Hold
Description	Specifies that a user hold be applied to the job at submission time.
Example	`> msub -h cmd.ll` *The job will be submitted with a user hold on it.*

-I	
Name	Interactive
Description	Declares the job is to be run interactively. qsub must exist on the same host as msub if the interactive job is destined for a TORQUE cluster, because the interactive msub request will be converted to a qsub -I request.
Example	`> msub -I job117.sh` *The job will be submitted in interactive mode.*

-j	
Name	Join
Format	[eo|oe|n]
Default	n (not merged)

-j	
Description	If `eo` is specified, the error and output streams are merged into the *error* stream. If `oe` is specified, the error and output streams will be merged into the *output* stream. If using either the -e or the -o option and the `-j eo|oe` option, the `-j` option takes precedence and all standard error and output messages go to the chosen output file.
Example	`> msub -j oe cmd.sh` *STDOUT and STDERR will be merged into one file.*

-k	
Name	Keep
Format	[e|o|eo|oe|n]
Default	n (not retained)
Description	Defines which (if either) of output and error streams will be retained on the execution host (overrides path for stream).
Example	`> msub -k oe myjob.sh` *STDOUT and STDERR for the job will be retained on the execution host.*

-K	
Name	Continue Running
Format	N/A
Description	Tells the client to continue running until the submitted job is completed. The client will query the status of the job every 5 seconds. The time interval between queries can be specified or disabled via MSUBQUERYINTERVAL. Use the `-K` option sparingly (if at all) as it slows down the Moab scheduler with frequent queries. Running ten jobs with the `-K` option creates an additional fifty queries per minute for the scheduler.

-K

Example

```
> msub -K newjob.sh
3
Job 3 completed*
```

**Only shows up after job completion.*

-l

Name

Resource List

Format

<STRING>

-l [BANDWIDTH|DDISK|DEADLINE|DEPEND|DMEM|EXCLUDENODES|FEATURE...|]

Additional options can be referenced on the resource manager extensions page.

Description

Defines the resources that are required by the job and establishes a limit to the amount of resource that can be consumed. Resources native to the resource manager, scheduler resource manager extensions, or job flags may be specified. Note that resource lists are dependent on the resource manager in use.

For information on specifying multiple types of resources for allocation, see Multi-Req Support.

Example

```
> msub -l nodes=32:ppn=2,pmem=1800mb,walltime=3600,VAR=testvar:
myvalue cmd.sh
```

The job requires `32` nodes with `2` processors each, `1800` MB per task, a walltime of `3600` seconds, and a variable named `testvar` with a value of `myvalue`.

If JOBNODEMATCHPOLICY is not set, Moab does not reserve the requested number of processors on the requested number of nodes. It reserves the total number of requested processors (nodes x ppn) on any number of nodes. Rather than setting `nodes=<value>:ppn=<value>`, set `procs=<value>`, replacing *<value>* with the total number of processors the job requires. Note that JOBNODEMATCHPOLICY is not set by default.

```
> msub -l nodes=32:ppn=2 -l advres=!<resvid>
```

This entry would tell Moab to only consider resources outside of the specified <reservation id>.

-m	
Name	Mail Options
Format	*<STRING>* (either `n` or one or more of the characters `a`, `b`, and `e`)
Description	Defines the set of conditions (abort,begin,end) when the server will send a mail message about the job to the user.
Example	`> msub -m be cmd.sh` *Mail notifications will be sent when the job begins and ends.*

-M	
Name	Mail List
Format	*<user>*[@*<host>*][,*<user>*[@*<host>*],...]
Default	$JOBOWNER
Description	Specifies the list of users to whom mail is sent by the execution server. Overrides the **EMAILADDRESS** specified on the USERCFG credential.
Example	`> msub -M jon@node01,bill@node01,jill@node02 cmd.sh` *Mail will be sent to the specified users if the job is aborted.*

-N	
Name	Name
Format	*<STRING>*
Default	STDIN or name of job script
Description	Specifies the user-specified job name attribute.

-N	
Example	`> msub -N chemjob3 cmd.sh` *Job will be associated with the name `chemjob3`.*

-o	
Name	Output Path
Format	[<*hostname*>:]<*path*> - %J and %I are acceptable variables. %J is the master array name and %I is the array member index in the array.
Default	`$SUBMISSIONDIR/$JOBNAME.o$JOBID`
Description:	Defines the path to be used for the standard output stream of the batch job. More variables are allowed when they are used in the job script instead of msub -o. In the job script, specify a `#PBS -o` line and input your desired variables. The allowable variables are: • OID • OTYPE • USER • OWNER • JOBID • JOBNAME Submitting a job script that has the line `#PBS -o $(USER)_$(JOBID)_$(JOBNAME).txt` results in a file called <*username*>_<*jobID*>_<*jobName*>.txt. Do not use msub -o when submitting a job script that has a `#PBS -o` line defined.

Example	`> msub -o test12/stdout.txt`
	The STDOUT stream of the job will be placed in the relative (to execution) directory specified.
	`> msub -t 1-2 -o /home/jsmith/simulations/%J-%I.out ~/sim5.sh`
	A job array is submitted and the name of the output files includes the master array index and the array member index.

-p	
Name	Priority
Format	*<INTEGER>* (between -1024 and 0)
Default	0
Description	Defines the priority of the job. To enable priority range from -1024 to +1023, see ENABLEPOSUSERPRIORITY.
Example	`> msub -p 25 cmd.sh` *The job will have a user priority of 25.*

-q	
Name	Destination Queue (Class)
Format	[*<queue>*][@*<server>*]
Default	[*<DEFAULT>*]
Description	Defines the destination of the job.
Example	`> msub -q priority cmd.sh` *The job will be submitted to the priority queue.*

-r	
Name	Rerunable
Format	[y|n]
Default	n
Description:	Declares whether the job is rerunable.

-r	
Example	`> msub -r n cmd.sh` *The job cannot be rerun.*

-S	
Name	Shell
Format	*<path>*[@*<host>*][,*<path>*[@*<host>*],...]
Default	$SHELL
Description	Declares the shell that interprets the job script.
Example	`> msub -S /bin/bash` *The job script will be interpreted by the `/bin/bash` shell.*

-t	
Name	Job Arrays
Format	*<name>*.[*<indexlist>*]%*<limit>*
Description	Starts a job array with the jobs in the index list. The limit variable specifies how many jobs may run at a time. For more information, see Submitting Job Arrays. Moab enforces an internal limit of 100,000 sub-jobs that a single array job submission can specify.
Example	`> msub -t myarray.[1-1000]%4`

-u	
Name	User List

-u	
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>*,...] The -v flag is limited to about 500 characters.
Description	Expands the list the environment variables that are exported to the job (taken from the msub command environment).
Example	`> msub -v DEBUG cmd.sh` *The `DEBUG` environment variable will be defined for the job.*

-V	
Name	All Variables
Description	Declares that all environment variables in the msub environment are exported to the batch job
Example	`> msub -V cmd.sh` *All environment variables will be exported to the job.*

-W	
Name	Additional Attributes
Format	*<string>*
Description	Allows for specification of additional job attributes (See Resource Manager Extension)

Example	`> msub -W x=GRES:matlab:1 cmd.sh`
	The job requires one resource of `matlab`.
	This flag can be used to set a filter for what name spaces will be passed from a job to a trigger using a comma-delimited list. This limits the trigger's action to objects contained in certain workflows. For more information, see **Requesting name space variables** on page 687.
	`> msub -W x="trigns=vc1,vc2"`
	The job passes name spaces `vc1` and `vc2` to triggers.

-x	
Format	*<script>* or *<command>*
Description	When running an interactive job, the `-x` flag makes it so that the corresponding script won't be parsed for PBS directives, but is instead a command that is launched once the interactive job has started. The job terminates at the completion of this command. This option works only when using TORQUE. The `-x` option for msub differs from qsub in that qsub does not require the script name to come directly after the flag. The msub command requires a script or command immediately after the `-x` declaration.
Example	`> msub -I -x ./script.pl` `> msub -I -x /tmp/command`

-z	
Name	Silent Mode
Description	The job's identifier will not be printed to stdout upon submission.

-z	
Example	`> 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:

Language	Notes
PBS/TORQUE Job Submission Language	---
LoadLeveler Job Submission Language	Use the INSTANTSTAGE parameter as only a subset of the command file keywords are interpreted by Moab.
SSS XML Job Object Specification	---
LSF Job Submission Language	enabled in Moab 4.2.4 and higher

/etc/msubrc

Sites that wish to automatically add parameters to every job submission can populate the file `/etc/msubrc` with global parameters that every job submission will inherit.

For example, if a site wished every job to request a particular generic resource they could use the following `/etc/msubrc`:

```
-W x=GRES:matlab:2
```

Usage Notes

msub is designed to be as flexible as possible, allowing users accustomed to PBS, LSF, or LoadLeveler syntax, to continue submitting jobs as they normally would. It is not recommended that different styles be mixed together in the same msub command.

When only one resource manager is configured inside of Moab, all jobs are immediately staged to the only resource manager available. However, when multiple resource managers are configured Moab will determine which resource manager can run the job soonest. Once this has been determined, Moab will stage the job to the resource manager.

It is possible to have Moab take a best effort approach at submission time using the `forward` flag. When this flag is specified, Moab will do a quick check and make an intelligent guess as to which resource manager can run the job soonest and then immediately stage the job.

Moab can be configured to instantly stage a job to the underlying resource manager (like TORQUE/LOADLEVELER) through the parameter INSTANTSTAGE. When set inside `moab.cfg`, Moab will migrate the job instantly to an appropriate resource manager. Once migrated, Moab will destroy all knowledge of the job and refresh itself based on the information given to it from the underlying resource manager.

In most instances Moab can determine what syntax style the job belongs to (PBS or LoadLeveler); if Moab is unable to make a guess, it will default the style to whatever resource manager was configured at compile time. If LoadLeveler and PBS were both compiled then LoadLeveler takes precedence.

Moab can translate a subset of job attributes from one syntax to another. It is therefore possible to submit a PBS style job to a LoadLeveler resource manager, and vice versa, though not all job attributes will be translated.

Examples

Example 4-39:

```
> msub -l nodes=3:ppn=2,walltime=1:00:00,pmem=100kb script2.pbs.cmd
4364.orion
```

Example 4-40:

This example is the XML-formatted version of the above example. See Submitting Jobs via msub in XML for more information.

```
<job>
  <InitialWorkingDirectory>/home/user/test/perlAPI
  </InitialWorkingDirectory>
  <Executable>/home/user/test/perlAPI/script2.pbs.cmd
  </Executable>
  <SubmitLanguage>PBS</SubmitLanguage>
  <Requested>
    <Feature>ppn2</Feature>
    <Processors>3</Processors>
    <WallclockDuration>3600</WallclockDuration>
  </Requested>
</job>
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mjobctl command to view, modify, and cancel jobs
- checkjob command to view detailed information about the job
- mshow command to view all jobs in the queue
- DEFAULTSUBMITLANGUAGE parameter
- MSUBQUERYINTERVAL parameter
- SUBMITFILTER parameter
- Applying the msub Submit Filter for job script sample

Applying the msub Submit Filter

When 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

```
#!/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;
```

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.

Global job submit filter

To configure Moab to automatically apply a filter to all job submissions, use the **SERVERSUBMITFILTER** on page 912 parameter. **SERVERSUBMITFILTER** specifies the path to a global job submit filter script, which Moab will run on the head node and apply to every job submitted.

```
SERVERSUBMITFILTER /opt/moab/scripts/jobFilter.pl
```

Moab runs `jobFilter.pl`, located in the `/opt/moab/scripts` directory, on the head node, applying the filter to all jobs submitted.

Submitting Jobs via msub in XML

The following describes the XML format used with the msub command to submit a job to a Moab server. This information can be used to implement a filter and modify the XML normally generated by the msub

command. The XML format described in what follows is based on a variant of the Scalable Systems Software Job Object Specification.

Overall XML Format

The overall format of an XML request to submit a job can be shown through the following example:

```
<job>
**job attribute children**
</job>
```

An example of a simple job element with all the required children for a job submission is as follows:

```
<job>
    <Owner>user</Owner>
    <UserId>user</UserId>
    <GroupId>group</GroupId>
    <InitialWorkingDirectory>/home/user/directory</InitialWorkingDirectory>
    <UMask>18</UMask>
    <Executable>/full/path/to/script/or/first/line/of/stdin</Executable>
    <SubmitLanguage>Resource Manager Type</SubmitLanguage>
    <SubmitString>\START\23!/usr/bin/ruby\0contents\20of\20script</SubmitString>
  </job>
```

The section that follows entitled Job Element Format describes the possible attributes and their meanings in detail. In actuality, all that is needed to run a job in Moab is something similar to the following:

```
<job>
  <SubmitString>\START\23!/bin/sh\0asleep\201000</SubmitString>
</job>
```

This piece of XML requests Moab to submit a job using the contents of the SubmitString tag as a script, which is in this case a simple sh script to sleep for 1000 seconds. The msub command will create default values for all other needed attributes.

Job Element Format

The job element of the submission request contains a list of children and string values inside the children that represent the attribute/value pairs for the job. The earlier section, Overall XML Format, gives an example of this format. This section explains these attributes in detail.

Arguments — The arguments to be passed to the program are normally specified as arguments after the first argument specifying the script to be executed.

EligibleTime — The minimum time after which the job is eligible. This is the equivalent of the `-a` option in msub. Format: `[[[[CC]YY]MM]DD]hhmm[.SS]`

Environment — The semi-colon list of environment variables that are exported to the job (taken from the msub command environment). The `-V` msub flag, for example, adds all the environment variables present at the time msub is invoked. Environment variables are delimited by the `~rs;` characters. Following is an example of the results of the `msub -v arg1=1,arg2=2` command:

```
<Environment>arg1=1~rs;arg2=2~rs;</Environment>
```

ErrorFile — Defines the path to be used for the standard error stream of the batch job. This is equivalent to the `-e` flag in msub.

Executable — This is normally either the name of the script to be executed, or the first line of the script if it is passed to msub through standard input.

Extension — The resource manager extension string. This can be specified via the command line in a number of ways, including the -W x= directive. Some other requests, such as some extensions used in the `-l` flag, are also converted to an extension string. The element has the following format:

```
<Extension>x=extension</Extension>
```

See Using the Extension Element to Submit Triggers for additional information on the extension element.

GroupId — The string name of the group of the user submitting the job. This will correspond to the user's primary group on the operating system.

Hold — Specifies that a user hold be applied to the job at submission time. This is the equivalent to the msub flag `-h`. It will have the form:

```
<Hold>User</Hold>
```

InitialWorkingDirectory — Specifies in which directory the job should begin executing. This is equivalent to the `-d` flag in the msub command.

```
<InitialWorkingDirectory>/home/user/directory</InitialWorkingDirectory>
```

Interactive — Specifies that the job is to be interactive. This is the equivalent of the `-I` flag in msub.

```
<Interactive>TRUE</Interactive>
```

JobName — Specifies the user-specified job name attribute. This is equivalent to the `-N` flag in msub.

NotificationList — Specifies the job states after which an email should be sent and also specifies the users to be emailed. This is the equivalent of the `-m` and `-M` options in msub.

```
<NotificationList URI=user1:user2>JobFail,JobStart,JobEnd</NotificationList>
```

In this example, the command `msub -m abe -M user1:user2` ran indicating that emails should be sent when a job fails, starts, or ends, and that they should be sent to `user1` and `user2`.

OutputFile — Defines the path to be used for the standard output stream of the batch job. This is the equivalent of the `-o` flag in msub.

Priority — A user-requested priority value. This is the equivalent to the msub `-p` flag.

ProjectId — Defines the account associated with the job. This is equivalent to the `-A` msub flag.

QueueName — The requested class of the job. This is the equivalent of the msub `-q` flag.

Requested — Specifies resources and attributes the job specifically requests and has the following form:

```
<Requested>
     <... requested attributes>
</Requested>
```

See the section dedicated to requestable attributes in this element.

RMFlags — Flags that will get passed directly to the resource manager on job submission. This is equivalent to any arguments listed after the `-l msub` flag.

```
<RMFlags>arg1 arg2 arg3</RMFlags>
```

ShellName — Declares the shell that interprets the job script. This is equivalent to the `msub` flag `-S`.

SubmitLanguage — Resource manager whose language the job is using. Use TORQUE to specify a TORQUE resource manager.

SubmitString — Contains the contents of the script to be run, retrieved either from an actual script or from standard input. This also includes all resource manager specific directives that may have been in the script already or added as a result of other command line arguments.

TaskGroup — Groups a set of requested resources together. It does so by encapsulating a Requested element. For example, the command `msub -l nodes=2+nodes=3:ppn=2` generates the following XML:

```
<TaskGroup>
    <Requested>
      <Processors>2</Processors>
      <TPN>2</TPN>
    </Requested>
  </TaskGroup>
  <TaskGroup>
    <Requested>
      <Processors>2</Processors>
    </Requested>
  </TaskGroup>
```

UserId — The string value of the user ID of the job owner. This will correspond to the user's name on the operating system.

Using the Extension Element to Submit Triggers

Use the Extension element to submit triggers. With the exception of certain characters, the syntax for trigger creation is the same for non-XML trigger submission. See **About object triggers** on page 655 for more information on triggers. The ampersand (`&`) and less than sign (`<`) characters must be replaced for the XML to be valid. The following example shows how the Extension element is used to submit multiple triggers (separated by a semi-colon). Note that ampersand characters are replaced with `&` in the example:

```
<Job>
  <UserId>user1</UserId>
  <GroupId>user1</GroupId>
  <Arguments>60</Arguments>
  <Executable>/bin/sleep</Executable>

<Extension>x=trig:AType=exec&amp;Action="env"&amp;EType=start;trig:AType=exec&amp;Acti
on="trig2.sh"&amp;EType=end</Extension>
  <Processors>3</Processors>
  <Disk>500</Disk>
  <Memory>1024</Memory>
  <Swap>600</Swap>
  <WallclockDuration>300</WallclockDuration>
  <Environment>PERL5LIB=/perl5:</Environment>
</Job>
```

Elements Found in Requested Element

The following describes the tags that can be found in the Requested sub-element of the job element in a job submission request.

Nodes — A list of nodes that the job requests to be run on. This is the equivalent of the `-l hosts=<host-list>` msub directive.

```
<Requested>
  <Nodes>
    <Node>n1:n2</Node>
  </Nodes>
</Requested>
```

In this example, the users requested the hosts `n1` and `n2` with the command `msub -l host=n1:n2`.

Processors — The number of processors requested by the job. The following example was generated with the command `msub -l nodes=5`:

```
<Requested>
    <Processors>5</Processors>
</Requested>
```

TPN — Tasks per node. This is generated using the ppn resource manager extensions. For example, from `msub -l nodes=3:ppn=2`, the following results:

```
  <Requested>
    <Processors>6</Processors>
    <TPN>2</TPN>
  </Requested>
```

WallclockDuration — The requested wallclock duration of the job. This attribute is specified in the Requested element.

```
<Requested>
  <WallclockDuration>3600</WallclockDuration>
</Requested>
```

Related topics

- Applying the msub Submit Filter
- SUBMITFILTER parameter

mvcctl

(Moab Virtual Container Control)

Synopsis

- mvcctl -a *<OType>*:*<OName>*[,*<OType>*:*<OName>*] *<name>*
- mvcctl -c [*<description>*]
- mvcctl -d *<name>*

- mvcctl -m *<ATTR>*=VAL[,*<ATTR>*=*<VAL>*] *<name>*
- mvcctl -q [*<name>*|ALL] [--xml][--blocking][--flags=fullxml]
- mvcctl -r *<OType>*:*<OName>*[,*<OType>*:*<OName>*] *<name>*
- mvcctl -x *<action><name>*

Overview

A virtual container (VC) is a logical grouping of objects with a shared variable space and applied policies. Containers can hold virtual machines, jobs, reservations, and nodes. Containers can also be nested inside other containers.

A VC can be owned by a user, group, or account. Users can only view VCs to which they have access. Level 1 administrators (Admin1) can view and modify all VCs. The owner can also be changed. When modifying the owner, you must also specify the owner type:

```
mvcctl -m OWNER=acct:bob myvc
```

Adding objects to VCs at submission: You associate jobs, VMs, and reservations with a specified VC upon submission. For example,

- mrsvctl -c ... -H *<VC>*
- msub ... -W x="vc=*<VC>*"
- mvmctl -c ...,vc=*<VC>*

The user who submits objects must have access to the VC or the command is rejected.

FullXML flag

The FullXML flag will cause the mvcctl -q command to show VCs in a hierarchical manner. If doing a non-XML (plaintext) query, sub-VCs will be listed inside their parent VCs. Each VC will be indented more than its parent.

```
VC[vc2] (vc2)
     Owner: user:jason
     VCs:
     VC[vc1] (vc1)
         Owner: user:jason
         Jobs:  Moab.1
         Rsvs:  system.1
         VCs:
         VC[vc3] (vc3)
             Owner: user:jason
     VC[vc4] (vc4)
         Owner: user:jason
```

If doing an XML query, the XML for all sub-objects (VCs, but also reservations, jobs, etc.) will also be included in the VC.

```
<Data>
   <vc DESCRIPTION="vc2" NAME="vc2" OWNER="user:jason">
     <vc DESCRIPTION="vc1" NAME="vc1" OWNER="user:jason">
       <job CmdFile="sleep 7200" Flags="GLOBALQUEUE,NORMSTART"
```

```
      Group="jason" JobID="Moab.1" PAL="[base]" RM="internal"
      ReqAWDuration="2:00:00" User="jason">
         <req Index="0"></req>
       </job>
       <rsv ACL="RSV=%=system.1=;" AUser="jason"
       AllocNodeList="n0,n1,n2,n3,n4,n5,n6,n7,n8,n9" HostExp="ALL"
       HostExpIsSpecified="TRUE" Name="system.1" Partition="base"
       ReqNodeList="n0:1,n1:1,n2:1,n3:1,n4:1,n5:1,n6:1,n7:1,n8:1,n9:1"
       Resources="PROCS=[ALL]" StatCIPS="5964.00" SubType="Other"
       Type="User" ctime="1299953557" duration="3600"
       endtime="1299957157"
       flags="ISCLOSED,ISGLOBAL,ISACTIVE,REQFULL"
       starttime="1299953557">
         <ACL aff="neutral" cmp="%=" name="system.1" type="RSV">
         </ACL>
         <CL aff="neutral" cmp="%=" name="system.1" type="RSV"></CL>
         <History>
           <event state="PROCS=40" time="1299953557"></event>
         </History>
       </rsv>
       <vc DESCRIPTION="vc3" NAME="vc3" OWNER="user:jason"></vc>
     </vc>
     <vc DESCRIPTION="vc4" NAME="vc4" OWNER="user:jason"></vc>
   </vc>
 </Data>
```

Note that the XML from the blocking and non-blocking commands may differ.

Virtual Container Flags

The following table indicates available virtual container (VC) flags and associated descriptions. Note that the `Deleting`, `HasStarted`, and `Workflow` flags cannot be set by a user but are helpful indicators of status.

VC Flags	
DestroyObjects	When the VC is destroyed, any reservations, jobs, and VMs in the VC are also destroyed. This is recursive, so any objects in sub-VCs are also destroyed. Nodes are not removed.
DestroyWhenEmpty	When the VC is empty, it is destroyed.
Deleting	Set by the scheduler when the VC has been instructed to be removed. Internal flag. Administrators cannot set or clear this flag.
HasStarted	This flag is set on a VC workflow where at least one job has started. Internal flag. Administrators cannot set or clear this flag.
HoldJobs	This flag will place a hold on any job that is submitted to the VC while this flag is set. It is not applied for already existing jobs that are added into the VC. If a job with a workflow is submitted to the VC, all jobs within the workflow are placed on hold.

VC Flags	
Workflow	Designates this VC as a VC that is for workflows. This flag is set when generated by a job template workflow. Workflow jobs can only be attached to one workflow VC. Internal flag. Administrators cannot set or clear this flag.

Format

-a	
Format	mvcctl -a*<OType>*:*<OName>*[,*<OType>*:*<OName>*] *<name>* Where *<OType>* is one of JOB, RSV, NODE, VC, or VM.
Description	Add the given object(s).
Example	`mvcctl -a JOB:Moab.45 vc13` `>>job 'Moab.45' added to VC 'vc13'`

-c	
Format	mvcctl -c [*<description>*]
Description	Create a virtual container (VC). The VC name is auto-generated. It is recommended that you supply a description; otherwise the description is the same as the auto-generated name.
Example	`mvcctl -c "Linux testing machine"` `>>VC 'vc13' created`

-d	
Format	mvcctl -d*<lab01>*
Description	Destroy the VC.

Example	`mvcctl -d vc13`
	`>>VC 'vc13'`
	`destroyed`

-m	
Format	mvcctl -m*<ATTR>*=VAL[,*<ATTR>*=*<VAL>*] *<name>*
Description	Modify the VC. Attributes are flags, owner, reqstarttime, reqnodeset, variables, and owner; note that only the owner can modify owner. Use reqstarttime when implementing guaranteed start time to specify when jobs should start. The reqnodeset attribute indicates the node set that jobs should run in that are submitted to a virtual container.

Example	`mvcctl -m variables+=HV=node8 vc13`
	`>>VC 'vc13' successfully modified`
	
	`mvcctl -m flags+=DESTROYWHENEMPTY vc1`
	`>>VC 'vc1' successfully modified`

-q	
Format	mvcctl -q [*<name>*|ALL] [--xml][--blocking][--flags=fullxml]
Description	Query VCs

Example	`mvcctl -q ALL`
	`VC[vc13] (Linux testing machine)`
	`Create Time: 1311027343    Creator: jdoe`
	`Owner: user:jdoe`
	`ACL:   USER=%=jdoe+;`
	`Jobs:  Moab.45`
	`Vars:  HV=node88`
	`Flags: DESTROYWHENEMPTY`

-r	
Format	mvcctl -r*<OType>*:*<OName>*[,*<OType>*:*<OName>*] *<name>* Where *<OType>* is one of JOB, RSV, NODE, VC, or VM.
Description	Remove the given object(s) from the VC.
Example	`mvcctl -r JOB:Moab.45 vc13` `>>job 'Moab.45' removed from VC 'vc13'`

-x	
Format	mvcctl -x*<action><name>*
Description	Executes the given action on the virtual container (VC).
Example	`mvcctl -x schedulevc vc1`

mvmctl

Synopsis

mvmctl -d *<vmid>* mvmctl -f *<migrationPolicy>* mvmctl -m [*<options>*] *<vmid>* mvmctl -M dsthost=*<newhost><vmid>* mvmctl -q*<vmid>* [--blocking] [--xml] mvmctl -w state=drained

Overview

mvmctl controls the modification, querying, migration, and destruction of virtual machines (VMs).

Format

-d	
Name	Destroy
Format	*<vmid>*
Description	Destroys the specified VM.
Example	`> mvmctl -d oldVM`

-f	
Name	Force Migrate
Format	mvmctl -f consolidation|overcommit [--flags=eval [--xml]]
Description	Forces the migration policy on the system. The `eval` flag causes Moab to run through migration routines and report the results without actually migrating the VMs.

-f	

Example	`> mvmctl -f consolidation --flags=eval`
	
	Moab returns a report like the following:
	
	`1: VM 'vm1' from 'h0' to 'h3'`
	`2: VM 'vm2' from 'h0' to 'h5'`

-m	
Name	Modify
Format	[*<options>*] *<vmid>* The *<options>* variable is a comma-separated list of *<attr>*=*<value>* pairs.
Description	Modifies the VM.

-m	

Example	`> mvmctl -m gevent=hitemp:'mymessage' myNewVM`
	Gevents can be set using `gevent`.
	`> mvmctl -m gmetric=bob:5.6 myNewVM`
	Gmetrics can be set using `gmetric`.
	`> mvmctl -m os=compute myNewVM`
	Reprovisioning is done by changing `os`.
	`> mvmctl -m powerstate=off myNewVM`
	Power management is done by modifying `powerstate`.
	`> mvmctl -m variable=user:bob+purpose:myVM myNewVM`
	Disallow VM migration by using `cannotmigrate`.
	`> mvmctl -m flags=cannotmigrate myNewVM`
	Allow a VM to migrate by setting the `canmigrate` flag.
	`> mvmctl -m flags=canmigrate myNewVM`

Notes	• The variable option is a set-only operation. Previous variables will be over-written.

-M	
Name	Migrate
Format	dsthost=*<newhost><vmid>*

-M	
Description	Migrate the given VM to the destination host. When you set the *vmid* to ANY, Moab migrates the VM to any available eligible hypervisor. For this to work, the following conditions must be met: • The VM reports a **CPULOAD**, and it is greater than 0. • The VM's **AMEMORY** is less than its **CMEMORY**. This indicates that some memory is currently in use and tells Moab that the RM is reporting memory correctly. • The VM's state is not "Unknown." • All hypervisors report a **CPULOAD**, and it is greater than 0. • All hypervisors report an **AMEMORY**, and it is less than its **CMEMORY**. • All hypervisors report a hypervisor type.

Example	`> mvmctl -M dsthost=node05 myNewVM`
	myNewVM migrates to node05.
	`> mvmctl -M dsthost=ANY vm42`
	Moab migrates vm42 to a node based on policy destination limitations (such as the NoVMMigrations flag).

-q	
Name	Query
Format	<vmid> [--blocking] [--xml]
Description	Queries the specified VM; that is, it returns detailed information about the given VM. May be used with or without the `--xml` flag. ALL may also be used to display information about all VMs. This option gathers information from the Moab cache which prevents it from waiting for the scheduler, but the `--blocking` option can be used to bypass the cache and allow waiting for the scheduler.

Example	`> mvmctl -q myNewVM`
	`> mvmctl -q ALL --blocking`
	`> mvmctl -q ALL --xml`

-w	
Name	Constraint
Format	state=drained
Description	Overrides the HIDEDRAINED DISPLAYFLAGS attribute allowing display of VMs in a DRAINED state.
Example	`> mvmctl -q -w state=drained`

showbf

Synopsis

showbf [-A] [-a *account*] [-c *class*] [-d *duration*] [-D] [-f *features*] [-g *group*] [-L] [-m [==|>|>=|<|<=] *memory*] [-n *nodecount*] [-p *partition*] [-q *qos*] [-u *user*] [-v] [--blocking]

Overview

Shows what resources are available for immediate use.

The results Moab returns do not include resources that may be freed due to preemption.

This command can be used by any user to find out how many processors are available for immediate use on the system. It is anticipated that users will use this information to submit jobs that meet these criteria and thus obtain quick job turnaround times. This command incorporates down time, reservations, and node state information in determining the available backfill window.

If specific information is not specified, showbf will return information for the user and group running but with global access for other credentials. For example, if `-q qos` is not specified, Moab will return resource availability information for a job as if it were entitled to access all QoS based resources (i.e., resources covered by reservations with a QoS based ACL), if `-c class` is not specified, the command will return information for resources accessible by any class.

The showbf command incorporates node configuration, node utilization, node state, and node reservation information into the results it reports. This command does not incorporate constraints imposed by credential based fairness policies on the results it reports.

Access

By default, this command can be used by any user or administrator.

Parameters

Parameter	Description
ACCOUNT	Account name.
CLASS	Class/queue required.
DURATION	Time duration specified as the number of seconds or in [DD:]HH:MM:SS notation.
FEATURELIST	Colon separated list of node features required.
GROUP	Specify particular group.
MEMCMP	Memory comparison used with the `-m` flag. Valid signs are >, >=, ==, <=, and <.
MEMORY	Specifies the amount of required real memory configured on the node, (in MB), used with the `-m` flag.
NODECOUNT	Specify number of nodes for inquiry with `-n` flag.
PARTITION	Specify partition to check with `-p` flag.
QOS	Specify QoS to check with `-q` flag.
USER	Specify particular user to check with `-u` flag.

Flags

Flag	Description
-A	Show resource availability information for all users, groups, and accounts. By default, showbf uses the default user, group, and account ID of the user issuing the command.
-a	Show resource availability information only for specified account.
--blocking	Do not use cache information in the output. The `--blocking` flag retrieves results exclusively from the scheduler.
-d	Show resource availability information for specified duration.
-D	Display current and future resource availability notation.

Flag	Description
-g	Show resource availability information only for specified group.
-h	Help for this command.
-L	Enforce Hard limits when showing available resources.
-m	Allows user to specify the memory requirements for the backfill nodes of interest. It is important to note that if the optional MEMCMP and MEMORY parameters are used, they must be enclosed in single ticks (') to avoid interpretation by the shell. For example, enter `showbf -m '==256'` to request nodes with 256 MB memory.
-n	Show resource availability information for a specified number of nodes. That is, this flag can be used to force showbf to display only blocks of resources with at least this many nodes available.
-p	Show resource availability information for the specified partition.
-q	Show information for the specified QoS.
-r	Show resource availability for the specified processor count.
-u	Show resource availability information only for specified user.

Examples

Example 4-41:

In this example, a job requiring up to 2 processors could be submitted for immediate execution in partition `ClusterA` for any duration. Additionally, a job requiring 1 processor could be submitted for immediate execution in partition `ClusterB`. Note that by default, each task is tracked and reported as a request for a single processor.

```
> showbf
Partition     Tasks  Nodes   StartOffset      Duration       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               3      3      00:00:00      INFINITY  11:32:38_08/19
ReqID=0
ClusterA          1      1      00:00:00      INFINITY  11:32:38_08/19
ReqID=0
ClusterB          2      2      00:00:00      INFINITY  11:32:38_08/19
ReqID=0
```

StartOffset is the amount of time remaining before resources will be available.

Example 4-42:

In this example, the output verifies that a backfill window exists for jobs requiring a 3 hour runtime and at least 16 processors. Specifying job duration is of value when time based access is assigned to reservations (i.e., using the **SRCFG TIMELIMIT** ACL)

```
> showbf -r 16 -d 3:00:00
Partition    Tasks  Nodes   Duration    StartOffset    StartDate
---------    -----  -----   --------    -----------    ---------
ALL          20     20      INFINITY    00:00:00       09:22:25_07/19
```

Example 4-43:

In this example, a resource availability window is requested for processors located only on nodes with at least 512 MB of memory.

```
> showbf -m ' =512'
Partition    Tasks  Nodes   Duration    StartOffset    StartDate
---------    -----  -----   --------    -----------    ---------
ALL          20     20      INFINITY    00:00:00       09:23:23_07/19
ClusterA     10     10      INFINITY    00:00:00       09:23:23_07/19
ClusterB     10     10      INFINITY    00:00:00       09:23:23_07/19
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- showq
- mdiag -t

showq

Synopsis

showq [-b] [-g] [-l] [-c|-i|-r] [-n] [-o] [-p *partition*] [-R *rsvid*] [-u] [-v] [-w <CONSTRAINT>] [--blocking] [--noblock]

Overview

Displays information about active, eligible, blocked, and/or recently completed jobs. Since the resource manager is not actually scheduling jobs, the job ordering it displays is not valid. The showq command displays the actual job ordering under the Moab Workload Manager. When used without flags, this command displays all jobs in active, idle, and non-queued states.

Access

By default, this command can be run by any user. However, the `-c`, `-i`, and `-r` flags can only be used by level 1, 2, or 3 Moab administrators.

Flags

Flag	Description
-b	Display blocked jobs only
-c	Display details about recently completed jobs (see example, JOBCPURGETIME).
-g	Display system ids for all jobs.
-i	Display extended details about idle jobs. (see example)
-l	Display local/remote view.
-n	Displays normal showq output, but lists job names under JOBID
-o	Displays jobs in the active queue in the order specified (uses format `showq -o <specifiedOrder>`). Valid options include REMAINING, REVERSEREMAINING, JOB, USER, STATE, and STARTTIME. The default is REMAINING.
-p	Display only jobs assigned to the specified partition.
-r	Display extended details about active (running) jobs. (see example)
-R	Display only jobs which overlap the specified reservation.
-u	Display all running jobs for a particular user.
-v	Display local and full resource manager job IDs as well as partitions. If specified with the `-i` option, will display job reservation time. The `-v` option displays all array subjobs. All showq commands without the `-v` option show just the master jobs in an array.
-w	Display only jobs associated with the specified constraint. Valid constraints include user, group, acct, class, and qos (see showq -w example.).
--blocking	Do not use cache information in the output. The `--blocking` flag retrieves results exclusively from the scheduler.
--noblock	Use cache information for a faster response.

Details

Beyond job information, the 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

- **Default Report** on page 237
  - **Detailed Active/Running Job Report** on page 239
  - **Eligible Jobs** on page 238
  - **Detailed Completed Job Report** on page 242
- **Filtered Job Report** on page 244

Example 4-44: Default Report

The output of this command is divided into three parts, Active Jobs, Eligible Jobs, and Blocked Jobs.

```
> showq

active jobs------------------------
JOBIDUSERNAMESTATEPROCSREMAINING            STARTTIME

12941               sartois    Running    25     2:44:11  Thu Sep  1 15:02:50
12954                tgates    Running     4     2:57:33  Thu Sep  1 15:02:52
12944                 eval1    Running    16     6:37:31  Thu Sep  1 15:02:50
12946                tgates    Running     2  1:05:57:31  Thu Sep  1 15:02:50

4 active jobs             47 of 48 processors active (97.92%)
                          32 of 32 nodes active      (100.00%)

eligible jobs----------------------
JOBID               USERNAME      STATE  PROCS     WCLIMIT            QUEUETIME

12956               cfosdyke       Idle     32     6:40:00  Thu Sep  1 15:02:50
12969               cfosdyke       Idle      4     6:40:00  Thu Sep  1 15:03:23
12939                  eval1       Idle     16     3:00:00  Thu Sep  1 15:02:50
12940                mwillis       Idle      2     3:00:00  Thu Sep  1 15:02:50
12947                mwillis       Idle      2     3:00:00  Thu Sep  1 15:02:50
12949                  eval1       Idle      2     3:00:00  Thu Sep  1 15:02:50
12953                 tgates       Idle     10     4:26:40  Thu Sep  1 15:02:50
12955                  eval1       Idle      2     4:26:40  Thu Sep  1 15:02:50
12957                 tgates       Idle     16     3:00:00  Thu Sep  1 15:02:50
12963                  eval1       Idle     16  1:06:00:00  Thu Sep  1 15:02:52
12964                 tgates       Idle     16  1:00:00:00  Thu Sep  1 15:02:52
12937                allendr       Idle      9  1:00:00:00  Thu Sep  1 15:02:50
12962                 aacker       Idle      6    00:26:40  Thu Sep  1 15:02:50
12968                tamaker       Idle      1     4:26:40  Thu Sep  1 15:02:52

14 eligible jobs

blocked jobs-----------------------
JOBID               USERNAME      STATE  PROCS     WCLIMIT            QUEUETIME


0 blocked jobs

Total jobs:  18
```

The fields are as follows:

Column	Description
JOBID	Job identifier.
USERNAME	User owning job.
STATE	Job State. Current batch state of the job.
PROCS	Number of processors being used by the job.
REMAINING/WCLIMIT	For active jobs, the time the job has until it has reached its wallclock limit or for idle/blocked jobs, the amount of time requested by the job. Time specified in [DD:]HH:MM:SS notation.
STARTTIME	Time job started running.

Active Jobs

Active jobs are those that are Running or Starting and consuming resources. Displayed are the job id*, the job's owner, and the job state. Also displayed are the number of processors allocated to the job, the amount of time remaining until the job completes (given in HH:MM:SS notation), and the time the job started. All active jobs are sorted in "Earliest Completion Time First" order.

*Job ids may be marked with a single character to specify the following conditions:

Character	Description
_ (underbar)	job violates usage limit
*** (asterisk)**	job is backfilled AND is preemptible
+ (plus)	job is backfilled AND is NOT preemptible
- (hyphen)	job is NOT backfilled AND is preemptible

Detailed active job information can be obtained using the `-r` flag.

Eligible Jobs

Eligible Jobs are those that are queued and eligible to be scheduled. They are all in the Idle job state and do not violate any fairness policies or have any job holds in place. The jobs in the Idle section display the same information as the Active Jobs section except that the wallclock CPULIMIT is specified rather than job time REMAINING, and job QUEUETIME is displayed rather than job STARTTIME. The jobs in this

section are ordered by job priority. Jobs in this queue are considered eligible for both scheduling and backfilling.

Detailed eligible job information can be obtained using the `-i` flag.

Blocked Jobs

Blocked jobs are those that are ineligible to be run or queued. Jobs listed here could be in a number of states for the following reasons:

State	Description
Idle	Job violates a fairness policy. Use `diagnose -q` for more information.
UserHold	A user hold is in place.
SystemHold	An administrative or system hold is in place.
BatchHold	A scheduler batch hold is in place (used when the job cannot be run because the requested resources are not available in the system or because the resource manager has repeatedly failed in attempts to start the job).
Deferred	A scheduler defer hold is in place (a temporary hold used when a job has been unable to start after a specified number of attempts. This hold is automatically removed after a short period of time).
NotQueued	Job is in the resource manager state NQ (indicating the job's controlling scheduling daemon in unavailable).

A summary of the job queue's status is provided at the end of the output.

Example 4-45: Detailed Active/Running Job Report

```
> showq -r

active jobs------------------------
JOBID              S  PAR  EFFIC  XFACTOR  Q      USER    GROUP        MHOST PROCS
REMAINING           STARTTIME

12941              R    3 100.00      1.0  -   sartois   Arches       G5-014 25
2:43:31  Thu Sep  1 15:02:50
12954              R    3 100.00      1.0 Hi    tgates   Arches       G5-016  4
2:56:54  Thu Sep  1 15:02:52
12944              R    2 100.00      1.0 De     eval1  RedRock     P690-016 16
6:36:51  Thu Sep  1 15:02:50
12946              R    3 100.00      1.0  -    tgates   Arches       G5-001  2
1:05:56:51  Thu Sep  1 15:02:50

4 active jobs             47 of 48 processors active (97.92%)
                          32 of 32 nodes active      (100.00%)
```

```
Total jobs:  4
```

The fields are as follows:

Column	Description
JOBID	Name of active job.
S	Job State. Either R for Running or S for Starting.
PAR	Partition in which job is running.
EFFIC	CPU efficiency of job.
XFACTOR	Current expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit
Q	Quality Of Service specified for job.
USERNAME	User owning job.
GROUP	Primary group of job owner.
MHOST	Master Host running primary task of job.
PROCS	Number of processors being used by the job.
REMAINING	Time the job has until it has reached its wallclock limit. Time specified in HH:MM:SS notation.
STARTTIME	Time job started running.

After displaying the running jobs, a summary is provided indicating the number of jobs, the number of allocated processors, and the system utilization.

Column	Description
JobName	Name of active job.
S	Job State. Either R for Running or S for Starting.
CCode	Completion Code. The return/completion code given when a job completes. (Only applicable to completed jobs.)

Column	Description
Par	Partition in which job is running.
Effic	CPU efficiency of job.
XFactor	Current expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit
Q	Quality Of Service specified for job.
User	User owning job.
Group	Primary group of job owner.
Nodes	Number of processors being used by the job.
Remaining	Time the job has until it has reached its wallclock limit. Time specified in HH:MM:SS notation.
StartTime	Time job started running.

```
> showq -i

eligible jobs----------------------
JOBID                 PRIORITY  XFACTOR  Q      USER    GROUP  PROCS     WCLIMIT
CLASS      SYSTEMQUEUETIME

12956*                      20      1.0  -  cfosdyke  RedRock     32     6:40:00
batch  Thu Sep  1 15:02:50
12969*                      19      1.0  -  cfosdyke  RedRock      4     6:40:00
batch  Thu Sep  1 15:03:23
12939                       16      1.0  -     eval1  RedRock     16     3:00:00
batch  Thu Sep  1 15:02:50
12940                       16      1.0  -   mwillis   Arches      2     3:00:00
batch  Thu Sep  1 15:02:50
12947                       16      1.0  -   mwillis   Arches      2     3:00:00
batch  Thu Sep  1 15:02:50
12949                       16      1.0  -     eval1  RedRock      2     3:00:00
batch  Thu Sep  1 15:02:50
12953                       16      1.0  -    tgates   Arches     10     4:26:40
batch  Thu Sep  1 15:02:50
12955                       16      1.0  -     eval1  RedRock      2     4:26:40
batch  Thu Sep  1 15:02:50
12957                       16      1.0  -    tgates   Arches     16     3:00:00
batch  Thu Sep  1 15:02:50
12963                       16      1.0  -     eval1  RedRock     16  1:06:00:00
batch  Thu Sep  1 15:02:52
12964                       16      1.0  -    tgates   Arches     16  1:00:00:00
batch  Thu Sep  1 15:02:52
12937                        1      1.0  -   allendr  RedRock      9  1:00:00:00
batch  Thu Sep  1 15:02:50
```

```
12962                        1      1.2  -    aacker  RedRock      6    00:26:40
batch  Thu Sep  1 15:02:50
12968                        1      1.0  -   tamaker  RedRock      1     4:26:40
batch  Thu Sep  1 15:02:52

14 eligible jobs

Total jobs:  14
```

The fields are as follows:

Column	Description
JOBID	Name of job.
PRIORITY	Calculated job priority.
XFACTOR	Current expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit
Q	Quality Of Service specified for job.
USER	User owning job.
GROUP	Primary group of job owner.
PROCS	Minimum number of processors required to run job.
WCLIMIT	Wallclock limit specified for job. Time specified in HH:MM:SS notation.
CLASS	Class requested by job.
SYSTEMQUEUETIME	Time job was admitted into the system queue.

An asterisk at the end of a job (job `12956*` in this example) indicates that the job has a job reservation created for it. The details of this reservation can be displayed using the checkjob command.

Example 4-46: Detailed Completed Job Report

```
> showq -c
completed jobs------------------------
JOBID              SCCODE  PAR  EFFIC  XFACTOR  Q  USERNAME    GROUP        MHOST
PROC    WALLTIME           STARTTIME
13098              C     0  bas  93.17      1.0  -   sartois   Arches       G5-014
25    2:43:31  Thu Sep  1 15:02:50
```

```
13102                C     0  bas  99.55      2.2 Hi    tgates   Arches       G5-016
 4    2:56:54  Thu Sep  1 15:02:52
13103                C     2  tes  99.30      2.9 De     eval1  RedRock     P690-016
16    6:36:51  Thu Sep  1 15:02:50
13115                C     0  tes  97.04      1.0  -    tgates   Arches       G5-001
 2 1:05:56:51  Thu Sep  1 15:02:50
3 completed jobs
```

The fields are as follows:

Column	Description
JOBID	job id for completed job.
S	Job State. Either C for Completed or V for Vacated.
CCODE	Completion code reported by the job.
PAR	Partition in which job ran.
EFFIC	CPU efficiency of job.
XFACTOR	Expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit
Q	Quality of Service specified for job.
USERNAME	User owning job.
GROUP	Primary group of job owner.
MHOST	Master Host which ran the primary task of job.
PROCS	Number of processors being used by the job.
WALLTIME	Wallclock time used by the job. Time specified in [DD:]HH:MM:SS notation.
STARTTIME	Time job started running.

After displaying the active jobs, a summary is provided indicating the number of jobs, the number of allocated processors, and the system utilization.

Example 4-47: Filtered Job Report

Show only jobs associated with user `john` and class `benchmark`.

```
> showq -w class=benchmark -w user=john
...
```

Job Array

Job arrays show the name of the job array and then in parenthesis, the number of sub-jobs in the job array that are in the specified state.

```
> showq

active jobs------------------------
JOBID              USERNAME      STATE PROCS   REMAINING            STARTTIME

Moab.1(14)          aesplin    Running    14    00:59:41  Fri May 27 14:58:57

14 active jobs            14 of 14 processors in use by local jobs (100.00%)
2 of 2 nodes active      (100.00%)

eligible jobs----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME

Moab.1(4)           aesplin       Idle     4     1:00:00  Fri May 27 14:58:52

4 eligible jobs

blocked jobs-----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME

Moab.1(2)           aesplin    Blocked     2     1:00:00  Fri May 27 14:58:52

2 blocked jobs

Total jobs:  20
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- showbf - command to display resource availability.
- mdiag -j - command to display detailed job diagnostics.
- checkjob - command to check the status of a particular job.
- JOBCPURGETIME - parameter to adjust the duration of time Moab preserves information about completed jobs
- DISPLAYFLAGS - parameter to control what job information is displayed

showhist.moab.pl

Synopsis

`showhist.moab.pl` [-a *accountname*]
[-c *classname*] [-e *enddate*]
[-g *groupname*] [-j *jobid*] [-n *days*]

[-q *qosname*] [-s *startdate*]
[-u *username*]

Overview

The `showhist.moab.pl` script displays historical job information. Its purpose is similar to the checkjob command's, but `showhist.moab.pl` displays information about jobs that have already completed.

Access

By default, this script's use is limited to administrators on the head node; however, end users can also be given power to run the script. To grant access to the script to end users, move `showhist.moab.pl` from the `tools` directory to the `bin` directory.

Arguments

-a (Account)	
Format	*<ACCOUNTNAME>*
Description	Displays job records matching the specified account.
Example	`> showhist.moab.pl -a myAccount` *Information about jobs related to the account `myAccount` is displayed.*

-c (Class)	
Format	*<CLASSNAME>*
Description	Displays job records matching the specified class (queue).
Example	`> showhist.moab.pl -c newClass` *Information about jobs related to the class `newClass` is displayed.*

-e (End Date)	
Format	YYYY-MM-DD
Description	Displays the records of jobs recorded before or on the specified date.

-e (End Date)

Example

```
> showhist.moab.pl -e 2001-01-03
```

Information about all jobs recorded on or before January 3, 2001 is displayed.

```
> showhist.moab.pl -s 2011-01-01 -e 2011-01-31
```

Information is displayed about all jobs recorded in January 2011.

-g (Group)

Format	*<GROUPNAME>*
Description	Displays job records matching the specified group.
Example	`> showhist.moab.pl -g admins` *Information about jobs related to the group `admins` is displayed.*

-j (Job ID)

Format	*<JOBID>*
Description	Displays job records matching the specified job id.
Example	`> showhist.moab.pl -j moab01` *Information about job `moab01` is displayed.*

-n (Number of Days)

Format	*<INTEGER>*

-n (Number of Days)

Description	Restricts the number of past jobs to search by a specified number of days relative to today.
Example	`> showhist.moab.pl -n 90 -j moab924` *Displays job information for job `moab924`. The search is restricted to the last 90 days.*

-q (QoS)

Format	*<QOSNAME>*
Description	Displays job records matching the specified quality of service.
Example	`> showhist.moab.pl -q myQos` *Information about jobs related to the QoS `myQos` is displayed.*

-s (Start Date)

Format	YYYY-MM-DD
Description	Displays the records of jobs that recorded on the specified date and later.

Example	`> showhist.moab.pl -s 1776-07-04`
	Information about all jobs recorded on July 4, 1776 and later is displayed.
	`> showhist.moab.pl -s 2001-07-05 -e 2002-07-05`
	Information is displayed about all jobs recorded between July 5, 2001 and July 5, 2002.

-u (User)	
Format	*<USERNAME>*
Description	Displays job records matching the specified user.
Example	`> showhist.moab.pl -u bob` *Information about user `bob`'s jobs is displayed.*

Sample Output

```
> showhist.moab.pl
Job Id             : Moab.4
User Name          : user1
Group Name         : company
Queue Name         : NONE
Processor Count    : 4
Wallclock Duration: 00:00:00
Submit Time        : Mon Nov 21 10:48:32 2011
Start Time         : Mon Nov 21 10:49:37 2011
End Time           : Mon Nov 21 10:49:37 2011
Exit Code          : 0
Allocated Nodelist: 10.10.10.3

Job Id             : Moab.1
Executable         : 4
User Name          : user1
Group Name         : company
Account Name       : 1321897709
Queue Name         : NONE
Quality Of Service: 0M
Processor Count    : -0
Wallclock Duration: 00:01:05
Submit Time        : Mon Nov 21 10:48:29 2011
Start Time         : Mon Nov 21 10:48:32 2011
End Time           : Mon Nov 21 10:49:37 2011
Exit Code          : 0
Allocated Nodelist: 512M
```

Information is displayed for all completed jobs.

When a job's Start Time and End Time are the same, the job is infinite and still running.

Related topics

- checkjob - explains how to query for a status report for a specified job.
- mdiag -j command - display additional detailed information regarding jobs
- showq command - showq high-level job summaries

showres

Synopsis

showres [-f] [-n [-g]] [-o] [-r] [*reservationid*]

Overview

This command displays all reservations currently in place within Moab. The default behavior is to display reservations on a reservation-by-reservation basis.

Access

By default, this command can be run by any Moab administrator.

Flag	Description
-f	Show free (unreserved) resources rather than reserved resources. The -f flag cannot be used in conjunction with the any other flag
-g	When used with the -n flag, shows grep-able output with nodename on every line
-n	Display information regarding all nodes reserved by *<RSVID>*
-o	Display all reservations which overlap *<RSVID>* (in time and space) Not supported with -n flag
-r	Display reservation timeframes in relative time mode
-v	Show verbose output. If used with the -n flag, the command will display all reservations found on nodes contained in *<RSVID>*. Otherwise, it will show long reservation start dates including the reservation year.

Parameter	Description
RSVID	ID of reservation of interest — optional

Examples

Example 4-48:

```
> showres

ReservationID       Type S       Start         End    Duration    N/P    StartTime

12941                Job R   -00:05:01     2:41:39     2:46:40   13/25   Thu Sep  1
15:02:50
```

```
12944                Job R   -00:05:01     6:34:59     6:40:00   16/16   Thu Sep  1
15:02:50
12946                Job R   -00:05:01  1:05:54:59  1:06:00:00    1/2    Thu Sep  1
15:02:50
12954                Job R   -00:04:59     2:55:01     3:00:00    2/4    Thu Sep  1
15:02:52
12956                Job I  1:05:54:59  1:12:34:59     6:40:00   16/32   Fri Sep  2
21:02:50
12969                Job I     6:34:59    13:14:59     6:40:00    4/4    Thu Sep  1
21:42:50

6 reservations located
```

The above example shows all reservations on the system.

The fields are as follows:

Column	Description
Type	Reservation Type. This will be one of the following: Job or User.
ReservationID	This is the name of the reservation. Job reservation names are identical to the job name. User, Group, or Account reservations are the user, group, or account name followed by a number. System reservations are given the name SYSTEM followed by a number.
S	State. This field is valid only for job reservations. It indicates whether the job is (S)tarting, (R)unning, or (I)dle.
Start	Relative start time of the reservation. Time is displayed in HH:MM:SS notation and is relative to the present time.
End	Relative end time of the reservation. Time is displayed in HH:MM:SS notation and is relative to the present time. Reservations that will not complete in 1,000 hours are marked with the keyword INFINITY.
Duration	Duration of the reservation in HH:MM:SS notation. Reservations lasting more than 1,000 hours are marked with the keyword INFINITY.
Nodes	Number of nodes involved in reservation.
StartTime	Time Reservation became active.

Example 4-49:

```
> showres -n
reservations on Thu Sep  1 16:49:59

NodeName        Type      ReservationID   JobState Task       Start    Duration
StartTime
```

```
G5-001           Job              12946    Running    2    -1:47:09  1:06:00:00  Thu
Sep  1 15:02:50
G5-001           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-002           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-002           Job              12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-003           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-003           Job              12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-004           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-004           Job              12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-005           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-005           Job              12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-006           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-006           Job              12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-007           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-007           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-008           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-008           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-009           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-009           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-010           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-010           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-011           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-011           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-012           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-012           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-013           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-013           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-014           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-014           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-015           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-015           Job              12949    Running    2   -00:08:57     3:00:00  Thu
Sep  1 16:41:02
G5-016           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
```

```
Sep  2 21:02:50
G5-016           Job              12947    Running    2   -00:08:57     3:00:00  Thu
Sep  1 16:41:02
P690-001         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-002         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-003         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-004         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-005         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-006         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-007         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-008         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-009         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-010         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-011         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-012         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-013         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-013         Job              12969       Idle    1     4:52:51     6:40:00  Thu
Sep  1 21:42:50
P690-014         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-014         Job              12969       Idle    1     4:52:51     6:40:00  Thu
Sep  1 21:42:50
P690-015         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-015         Job              12969       Idle    1     4:52:51     6:40:00  Thu
Sep  1 21:42:50
P690-016         Job              12944    Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-016         Job              12969       Idle    1     4:52:51     6:40:00  Thu
Sep  1 21:42:50

52 nodes reserved
```

This example shows reservations for nodes.

The fields are as follows:

Column	Description
NodeName	Node on which reservation is placed.
Type	Reservation Type. This will be one of the following: Job or User.

Column	Description
ReservationID	This is the name of the reservation. Job reservation names are identical to the job name. User, Group, or Account reservations are the user, group, or account name followed by a number. System reservations are given the name SYSTEM followed by a number.
JobState	This field is valid only for job reservations. It indicates the state of the job associated with the reservation.
Start	Relative start time of the reservation. Time is displayed in HH:MM:SS notation and is relative to the present time.
Duration	Duration of the reservation in HH:MM:SS notation. Reservations lasting more than 1000 hours are marked with the keyword INFINITY.
StartTime	Time Reservation became active.

Example 4-50:

```
> showres 12956

ReservationID       Type S       Start         End    Duration    N/P    StartTime

12956                Job I  1:04:09:32  1:10:49:32     6:40:00   16/32   Fri Sep  2
21:02:50

1 reservation located
```

In this example, information for a specific reservation (job) is displayed.

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mrsvctl -c - create new reservations.
- mrsvctl -r - release existing reservations.
- mdiag -r - diagnose/view the state of existing reservations.
- Reservation Overview - description of reservations and their use.

showstart

Synopsis

showstart {*jobid*|*proccount*[@*duration*]|*s3jobspec*} [-e {all|hist|prio|rsv}] [-f] [-g [*peer*]] [-l qos=<QOS>] [--blocking] [--format=xml]

Overview

This command displays the estimated start time of a job based a number of analysis types. This analysis may include information based on historical usage, earliest available reservable resources, and priority based backlog analysis. Each type of analysis will provide somewhat different estimates based on current cluster environmental conditions. By default, only reservation based analysis is performed.

The start time estimate Moab returns does not account for resources that will become available due to preemption.

Historical analysis utilizes historical queue times for jobs which match a similar processor count and job duration profile. This information is updated on a sliding window which is configurable within `moab.cfg`

Reservation based start time estimation incorporates information regarding current administrative, user, and job reservations to determine the earliest time the specified job could allocate the needed resources and start running. In essence, this estimate will indicate the earliest time the job would start assuming this job was the highest priority job in the queue.

Priority based job start analysis determines when the queried job would fit in the queue and determines the estimated amount of time required to complete the jobs which are currently running or scheduled to run before this job can start.

In all cases, if the job is running, this command will return the time the job started. If the job already has a reservation, this command will return the start time of the reservation.

Access

By default, this command can be run by any user.

Parameters

Parameter	Description
--blocking	Do not use cache information in the output. The `--blocking` flag retrieves results exclusively from the scheduler.
DURATION	Duration of pseudo-job to be checked in format [[[DD:]HH:]MM:]SS (default duration is 1 second)
-e	Estimate method. By default, Moab will use the reservation based estimation method.
-f	Use feedback. If specified, Moab will apply historical accuracy information to Improve the quality of the estimate. See ENABLESTARTESTIMATESTATS for more information.
-l qos-s=<QOS>	Specifies what QoS the job must start under, using the same syntax as the **msub** command. Currently, no other resource manager extensions are supported. This flag only applies to hypothetical jobs by using the `proccount[@duration]` syntax.
JOBID	Job to be checked

Parameter	Description
PROCCOUNT	Number of processors in pseudo-job to be checked
S3JOBSPEC	XML describing the job according to the Dept. of Energy Scalable Systems Software/S3 job specification.

Examples

Example 4-51:

```
> 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 4-52:

```
> showstart 12@3600
job 12@3600 requires 12 procs for 1:00:00
Earliest start in         00:01:39 on Wed Aug 31 16:30:45
Earliest completion in     1:01:39 on Wed Aug 31 17:30:45
Best Partition: 32Bit
```

You cannot specify job flags when running showstart, and since a job by default can only run on one partition, showstart fails when querying for a job requiring more nodes than the largest partition available.

Additional Information

For reservation based estimates, the information provided by this command is more highly accurate if the job is highest priority, if the job has a reservation, or if the majority of the jobs which are of higher priority have reservations. Consequently, sites wishing to make decisions based on this information may want to consider using the RESERVATIONDEPTH parameter to increase the number of priority based reservations. This can be set so that most, or even all, idle jobs receive priority reservations and make the results of this command generally useful. The only caution of this approach is that increasing the RESERVATIONDEPTH parameter more tightly constrains the decisions of the scheduler and may resulting in slightly lower system utilization (typically less than 8% reduction).

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- checkjob
- showres

- showstats -f eststarttime
- showstats -f avgqtime
- Job Start Estimates

showstate

Synopsis

showstate

Overview

This command provides a summary of the state of the system. It displays a list of all active jobs and a text-based map of the status of all nodes and the jobs they are servicing. Basic diagnostic tests are also performed and any problems found are reported.

Access

By default, this command can be run by any Moab Administrator.

Examples

Example 4-53:

```
> showstate
cluster state summary for Wed Nov 23 12:00:21
    JobID              S      User    Group Procs   Remaining            StartTime
    ------------------ - --------- -------- ----- -----------  -------------------
(A)       fr17n11.942.0 R     johns    staff    16    13:21:15      Nov 22 12:00:21
(B)       fr17n12.942.0 S     johns    staff    32    13:07:11      Nov 22 12:00:21
(C)       fr17n13.942.0 R     johns    staff     8    11:22:25      Nov 22 12:00:21
(D)       fr17n14.942.0 S     johns    staff     8    10:43:43      Nov 22 12:01:21
(E)       fr17n15.942.0 S     johns    staff     8     9:19:25      Nov 22 12:01:21
(F)       fr17n16.942.0 R     johns    staff     8     9:01:16      Nov 22 12:01:21
(G)       fr17n17.942.0 R     johns    staff     1     7:28:25      Nov 22 12:03:22
(H)       fr17n18.942.0 R     johns    staff     1     3:05:17      Nov 22 12:04:22
(I)       fr17n19.942.0 S     johns    staff    24     0:54:38      Nov 22 12:00:22
Usage Summary:  9 Active Jobs  106 Active Nodes
              [0][0][0][0][0][0][0][0][0][1][1][1][1][1][1][1]
              [1][2][3][4][5][6][7][8][9][0][1][2][3][4][5][6]
Frame      2: XXXXXXXXXXXXXXXXXXXXXXX[ ][A][C][ ][A][C][C][A]
Frame      3: [ ][ ][ ][ ][ ][ ][A][ ][I][ ][I][ ][ ][ ][ ][ ]
Frame      4: [ ][I][ ][ ][ ][ ][A][ ][I][ ][ ][ ][E][ ][I][ ][E]
Frame      5: [F][ ][E][ ][ ][ ][ ][F][F][F][I][ ][ ][E][ ][E][E]
Frame      6: [ ][I][I][E][I][ ][ ][I][I][ ][ ][I][F][I][I][I][I][F]
Frame      7: [ ]XXX[ ]XXX[ ]XXX[ ]XXX[b]XXX[ ]XXX[ ]XXX[#]XXX
Frame      9: [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][E][ ]
Frame     11: [ ][ ][ ][ ][ ][ ][I][F][@][ ][A][I][ ][F][ ][A]
Frame     12: [A][ ][ ][A][ ][ ][C][A][ ][C][A][A][ ][ ][ ][ ]
Frame     13: [D]XXX[I]XXX[ ]XXX[ ]XXX[ ]XXX[ ]XXX[I]XXX[I]XXX
Frame     14: [D]XXX[I]XXX[I]XXX[D]XXX[ ]XXX[H]XXX[I]XXX[ ]XXX
Frame     15: [b]XXX[b]XXX[b]XXX[b]XXX[D]XXX[b]XXX[b]XXX[b]XXX
Frame     16: [b]XXX[ ]XXX[b]XXX[ ]XXX[b]XXX[b]XXX[ ]XXX[b]XXX
Frame     17: [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
Frame     21: [ ]XXX[b]XXX[b]XXX[ ]XXX[b]XXX[b]XXX[b]XXX[b]XXX
Frame     22: [b]XXX[b]XXX[b]XXX[ ]XXX[b]XXX[ ]XXX[b]XXX[b]XXX
Frame     27: [b]XXX[b]XXX[ ]XXX[b]XXX[b]XXX[b]XXX[b]XXX[b]XXX
```

```
Frame     28: [G]XXX[ ]XXX[D]XXX[ ]XXX[D]XXX[D]XXX[D]XXX[ ]XXX
Frame     29: [A][C][A][A][C][ ][A][C]XXXXXXXXXXXXXXXXXXXXXXXX
Key:  XXX:Unknown [*]:Down w/Job [#]:Down ['']:Idle w/Job [ ]:Idle [@]:Busy w/No Job
[!]:Drained
Key:  [a]:(Any lower case letter indicates an idle node that is assigned to a job)

Check Memory on Node fr3n07
Check Memory on Node fr4n06
Check Memory on Node fr4n09
```

In this example, nine active jobs are running on the system. Each job listed in the top of the output is associated with a letter. For example, job `fr17n11.942.0` is associated with the letter A. This letter can now be used to determine where the job is currently running. By looking at the system map, it can be found that job `fr17n11.942.0` (job A) is running on nodes `fr2n10`, `fr2n13`, `fr2n16`, `fr3n07`…

The key at the bottom of the system map can be used to determine unusual node states. For example, `fr7n15` is currently in the state down.

After the key, a series of warning messages may be displayed indicating possible system problems. In this case, warning message indicate that there are memory problems on three nodes, `fr3n07`, `fr4n06`, and `fr4n09`. Also, warning messages indicate that job `fr15n09.1097.0` is having difficulty starting. Node `fr11n08` is in state BUSY but has no job assigned to it (it possibly has a runaway job running on it).

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- Specifying Node Rack/Slot Location

showstats

Synopsis

showstats

showstats -a [*accountid*] [-v] [-t <TIMESPEC>]

showstats -c [*classid*] [-v] [-t <TIMESPEC>]

showstats -f <statistictype>

showstats -g [*groupid*] [-v] [-t <TIMESPEC>]

showstats -j [*jobtemplate*] [-t <TIMESPEC>]

showstats -n [*nodeid*] [-t <TIMESPEC>]

showstats -q [*qosid*] [-v] [-t <TIMESPEC>]

showstats -s

showstats -T [*leafid* | *tree-level*]

showstats -u [*userid*] [-v] [-t <TIMESPEC>]

Overview

This command shows various accounting and resource usage statistics for the system. Historical statistics cover the timeframe from the most recent execution of the mschedctl -f command.

Access

By default, this command can be run by any Moab level 1, 2, or 3 Administrator.

Parameters

Flag	Description
-a[<ACCOUNTID>]	Display account statistics. See **Account statistics** on page 259 for an example.
-c[<CLASSID>]	Display class statistics
-f <statistictype>	Display full matrix statistics (see showstats -f for full details)
-g[<GROUPID>]	Display group statistics. See **Group statistics** on page 260 for an example.
-j [<JOBTEMPLATE>]	Display template statistics
-n[<NODEID>]	Display node statistics (ENABLEPROFILING must be set). See **Node statistics** on page 262 for an example.
-q [<QOSID>]	Display QoS statistics
-s	Display general scheduler statistics

-t	Display statistical information from the specified timeframe:
	`<START_TIME>[,<END_TIME>]`
	`(ABSTIME: [HH[:MM[:SS]]][_MO[/DD[/YY]]] ie 14:30_06/20)`
	`(RELTIME: -[[[DD:]HH:]MM:]SS)`
	See **Statistics from an absolute time frame** on page 267 and **Statistics from a relative time frame** on page 267 for examples.
	Profiling must be enabled for the credential type you want statistics for. See Credential Statistics for information on how to enable profiling. Also, **-t** is not a stand-alone option. It must be used in conjunction with the **-a**, **-c**, **-g**, **-n**, **-q**, or **-u** flag.

-T	Display fairshare tree statistics. See **Fairshare tree statistics** on page 266 for an example.
-u[<USERID>]	Display user statistics. See **User statistics** on page 264 for an example.
-v	Display verbose information. See **Verbose statistics** on page 263 for an example.

Examples

Example 4-54: Account statistics

```
> showstats -a
Account Statistics Initialized Tue Aug 26 14:32:39
              |----- Running ------|---------------------------------- Completed ------
----------------------------|
  Account     Jobs Procs ProcHours Jobs    %   PHReq    %    PHDed    %   FSTgt  AvgXF
 MaxXF  AvgQH  Effic  WCAcc
   137651       16    92   1394.52  229  39.15 18486  45.26 7003.5  41.54 40.00   0.77
  8.15   5.21  90.70  34.69
   462212       11    63    855.27   43   7.35  6028  14.76 3448.4  20.45  6.25   0.71
  5.40   3.14  98.64  40.83
   462213        6    72    728.12   90  15.38  5974  14.63 3170.7  18.81  6.25   0.37
  4.88   0.52  82.01  24.14
   005810        3    24    220.72   77  13.16  2537   6.21 1526.6   9.06 -----   1.53
 14.81   0.42  98.73  28.40
   175436        0     0      0.00   12   2.05  6013  14.72  958.6   5.69  2.50   1.78
  8.61   5.60  83.64  17.04
   000102        0     0      0.00    1   0.17    64   0.16    5.1   0.03 -----  10.85
 10.85  10.77  27.90   7.40
   000023        0     0      0.00    1   0.17    12   0.03    0.2   0.00 -----   0.04
  0.04   0.19  21.21   1.20
```

This example shows a statistical listing of all active accounts. The top line (Account Statistics Initialized...) of the output indicates the beginning of the timeframe covered by the displayed statistics.

The statistical output is divided into two categories, Running and Completed. Running statistics include information about jobs that are currently running. Completed statistics are compiled using historical information from both running and completed jobs.

The fields are as follows:

Column	Description
Account	Account Number
Jobs	Number of running jobs
Procs	Number of processors allocated to running jobs
ProcHours	Number of proc-hours required to complete running jobs
Jobs*	Number of jobs completed
%	Percentage of total jobs that were completed by account
PHReq*	Total proc-hours requested by completed jobs
%	Percentage of total proc-hours requested by completed jobs that were requested by account

Column	Description
PHDed	Total proc-hours dedicated to active and completed jobs. The proc-hours dedicated to a job are calculated by multiplying the number of allocated procs by the length of time the procs were allocated, regardless of the job's CPU usage.
%	Percentage of total proc-hours dedicated that were dedicated by account
FSTgt	Fairshare target. An account's fairshare target is specified in the `fs.cfg` file. This value should be compared to the account's node-hour dedicated percentage to determine if the target is being met.
AvgXF*	Average expansion factor for jobs completed. A job's XFactor (expansion factor) is calculated by the following formula: (QueuedTime + RunTime) / WallClockLimit.
MaxXF*	Highest expansion factor received by jobs completed
AvgQH*	Average queue time (in hours) of jobs
Effic	Average job efficiency. Job efficiency is calculated by dividing the actual node-hours of CPU time used by the job by the node-hours allocated to the job.
WCAcc*	Average wallclock accuracy for jobs completed. Wallclock accuracy is calculated by dividing a job's actual run time by its specified wallclock limit. A job's wallclock accuracy is capped at 100% so even if a job exceeds its requested walltime it will report an accuracy of 100%.

* These fields are empty until an account has completed at least one job.

Example 4-55: Group statistics

```
> showstats -g
Group Statistics Initialized Tue Aug 26 14:32:39
              |----- Running ------|---------------------------------- Completed ------
---------------------------|
GroupName  GID Jobs Procs ProcHours Jobs    %   PHReq    %    PHDed    %   FSTgt
AvgXF  MaxXF  AvgQH  Effic  WCAcc
     univ  214   16    92   1394.52  229  39.15 18486  45.26 7003.5  41.54 40.00
0.77   8.15   5.21  90.70  34.69
      daf  204   11    63    855.27   43   7.35  6028  14.76 3448.4  20.45  6.25
0.71   5.40   3.14  98.64  40.83
    dnavy  207    6    72    728.12   90  15.38  5974  14.63 3170.7  18.81  6.25
0.37   4.88   0.52  82.01  24.14
     govt  232    3    24    220.72   77  13.16  2537   6.21 1526.6   9.06 -----
1.53  14.81   0.42  98.73  28.40
      asp  227    0     0      0.00   12   2.05  6013  14.72  958.6   5.69  2.50
1.78   8.61   5.60  83.64  17.04
    derim  229    0     0      0.00   74  12.65   669   1.64  352.5   2.09 -----
0.50   1.93   0.51  96.03  32.60
   dchall  274    0     0      0.00    3   0.51   447   1.10  169.2   1.00 25.00
```

```
0.52   0.88   2.49  95.82  33.67
      nih  239    0     0       0.00   17   2.91   170   0.42  148.1   0.88 -----
0.95   1.83   0.14  97.59  84.31
    darmy  205    0     0       0.00   31   5.30   366   0.90   53.9   0.32  6.25
0.14   0.59   0.07  81.33  12.73
  systems   80    0     0       0.00    6   1.03    67   0.16   22.4   0.13 -----
4.07   8.49   1.23  28.68  37.34
      pdc  252    0     0       0.00    1   0.17    64   0.16    5.1   0.03 -----
10.85  10.85  10.77  27.90   7.40
    staff    1    0     0       0.00    1   0.17    12   0.03    0.2   0.00 -----
0.04   0.04   0.19  21.21   1.20
```

This example shows a statistical listing of all active groups. The top line (Group Statistics Initialized...) of the output indicates the beginning of the timeframe covered by the displayed statistics.

The statistical output is divided into two categories, Running and Completed. Running statistics include information about jobs that are currently running. Completed statistics are compiled using historical information from both running and completed jobs.

The fields are as follows:

Column	Description
GroupName	Name of group.
GID	Group ID of group.
Jobs	Number of running jobs.
Procs	Number of procs allocated to running jobs.
ProcHours	Number of proc-hours required to complete running jobs.
Jobs*	Number of jobs completed.
%	Percentage of total jobs that were completed by group.
PHReq*	Total proc-hours requested by completed jobs.
%	Percentage of total proc-hours requested by completed jobs that were requested by group.
PHDed	Total proc-hours dedicated to active and completed jobs. The proc-hours dedicated to a job are calculated by multiplying the number of allocated procs by the length of time the procs were allocated, regardless of the job's CPU usage.
%	Percentage of total proc-hours dedicated that were dedicated by group.

Column	Description
FSTgt	Fairshare target. A group's fairshare target is specified in the `fs.cfg` file. This value should be compared to the group's node-hour dedicated percentage to determine if the target is being met.
AvgXF*	Average expansion factor for jobs completed. A job's XFactor (expansion factor) is calculated by the following formula: (QueuedTime + RunTime) / WallClockLimit.
MaxXF*	Highest expansion factor received by jobs completed.
AvgQH*	Average queue time (in hours) of jobs.
Effic	Average job efficiency. Job efficiency is calculated by dividing the actual node-hours of CPU time used by the job by the node-hours allocated to the job.
WCAcc*	Average wallclock accuracy for jobs completed. Wallclock accuracy is calculated by dividing a job's actual run time by its specified wallclock limit. A job's wallclock accuracy is capped at 100% so even if a job exceeds its requested walltime it will report an accuracy of 100%.

* These fields are empty until a group has completed at least one job.

Example 4-56: Node statistics

```
> showstats -n
node stats from Mon Jul 10 00:00:00 to Mon Jul 10 16:30:00
node       CfgMem MinMem MaxMem AvgMem | CfgProcs MinLoad MaxLoad AvgLoad
node01      58368      0  21122   5841        32    0.00   32.76   27.62
node02     122880      0  19466    220        30    0.00   33.98   29.54
node03      18432      0   9533   2135        24    0.00   25.10   18.64
node04      60440      0  17531   4468        32    0.00   30.55   24.61
node05      13312      0   2597   1189         8    0.00    9.85    8.45
node06      13312      0   3800   1112         8    0.00    8.66    5.27
node07      13312      0   2179   1210         8    0.00    9.62    8.27
node08      13312      0   3243   1995         8    0.00   11.71    8.02
node09      13312      0   2287   1943         8    0.00   10.26    7.58
node10      13312      0   2183   1505         8    0.00   13.12    9.28
node11      13312      0   3269   2448         8    0.00    8.93    6.71
node12      13312      0  10114   6900         8    0.00   13.13    8.44
node13      13312      0   2616   2501         8    0.00    9.24    8.21
node14      13312      0   3888    869         8    0.00    8.10    3.85
node15      13312      0   3788    308         8    0.00    8.40    4.67
node16      13312      0   4386   2191         7    0.00   18.37    8.36
node17      13312      0   3158   1870         8    0.00    8.95    5.91
node18      13312      0   5022   2397         8    0.00   19.25    8.19
node19      13312      0   2437   1371         8    0.00    8.98    7.09
node20      13312      0   4474   2486         8    0.00    8.51    7.11
node21      13312      0   4111   2056         8    0.00    8.93    6.68
node22      13312      0   5136   2313         8    0.00    8.61    5.75
node23      13312      0   1850   1752         8    0.00    8.39    5.71
node24      13312      0   3850   2539         8    0.00    8.94    7.80
```

```
node25     13312      0   3789   3702        8    0.00   21.22   12.83
node26     13312      0   3809   1653        8    0.00    9.34    4.91
node27     13312      0   5637     70        4    0.00   17.97    2.46
node28     13312      0   3076   2864        8    0.00   22.91   10.33
```

Example 4-57: Verbose statistics

```
> showstats -v
current scheduler time: Sat Aug 18 18:23:02 2007
moab active for      00:00:01   started on Wed Dec 31 17:00:00
statistics for iteration     0  initialized on Sat Aug 11 23:55:25
Eligible/Idle Jobs:                6/8      (75.000%)
Active Jobs:                      13
Successful/Completed Jobs:       167/167    (100.000%)
Preempt Jobs:                      0
Avg/Max QTime (Hours):           0.34/2.07
Avg/Max XFactor:                 1.165/3.26
Avg/Max Bypass:                  0.40/8.00
Dedicated/Total ProcHours:      4.46K/4.47K  (99.789%)
Preempt/Dedicated ProcHours:     0.00/4.46K  (0.000%)
Current Active/Total Procs:        32/32     (100.0%)
Current Active/Total Nodes:        16/16      (100.0%)
Avg WallClock Accuracy:          64.919%
Avg Job Proc Efficiency:         99.683%
Min System Utilization:          87.323% (on iteration 46)
Est/Avg Backlog:                02:14:06/03:02:567
```

This example shows a concise summary of the system scheduling state. Note that showstats and showstats -s are equivalent.

The first line of output indicates the number of scheduling iterations performed by the current scheduling process, followed by the time the scheduler started. The second line indicates the amount of time the Moab Scheduler has been scheduling in HH:MM:SS notation followed by the statistics initialization time.

The fields are as follows:

Column	Description
Active Jobs	Number of jobs currently active (Running or Starting).
Eligible Jobs	Number of jobs in the system queue (jobs that are considered when scheduling).
Idle Jobs	Number of jobs both in and out of the system queue that are in the LoadLeveler Idle state.
Completed Jobs	Number of jobs completed since statistics were initialized.
Successful Jobs	Jobs that completed successfully without abnormal termination.
XFactor	Average expansion factor of all completed jobs.
Max XFactor	Maximum expansion factor of completed jobs.

Column	Description
Max Bypass	Maximum bypass of completed jobs.
Available ProcHours	Total proc-hours available to the scheduler.
Dedicated ProcHours	Total proc-hours made available to jobs.
Effic	Scheduling efficiency (DedicatedProcHours / Available ProcHours).
Min Efficiency	Minimum scheduling efficiency obtained since scheduler was started.
Iteration	Iteration on which the minimum scheduling efficiency occurred.
Available Procs	Number of procs currently available.
Busy Procs	Number of procs currently busy.
Effic	Current system efficiency (BusyProcs/AvailableProcs).
WallClock Accuracy	Average wallclock accuracy of completed jobs (job-weighted average).
Job Efficiency	Average job efficiency (UtilizedTime / DedicatedTime).
Est Backlog	Estimated backlog of queued work in hours.
Avg Backlog	Average backlog of queued work in hours.

Example 4-58: User statistics

```
> showstats -u
User Statistics Initialized Tue Aug 26 14:32:39
              |----- Running ------|-------------------------------- Completed ------
--------------------------|
 UserName  UID Jobs Procs ProcHours Jobs    %   PHReq    %    PHDed    %   FSTgt
AvgXF  MaxXF  AvgQH  Effic  WCAcc
  moorejt 2617    1    16     58.80    2   0.34   221   0.54 1896.6  11.25 -----
1.02   1.04   0.14  99.52 100.00
    zhong 1767    3    24    220.72   20   3.42  2306   5.65 1511.3   8.96 -----
0.71   0.96   0.49  99.37  67.48
      lui 2467    0     0      0.00   16   2.74  1970   4.82 1505.1   8.93 -----
1.02   6.33   0.25  98.96  57.72
    evans 3092    0     0      0.00   62  10.60  4960  12.14 1464.3   8.69   5.0
0.62   1.64   5.04  87.64  30.62
   wengel 2430    2    64    824.90    1   0.17   767   1.88  630.3   3.74 -----
0.18   0.18   4.26  99.63   0.40
```

```
    mukho 2961    2    16     71.06    6   1.03   776   1.90  563.5   3.34 -----
0.31   0.82   0.20  93.15  30.28
  jimenez 1449    1    16    302.29    2   0.34   768   1.88  458.3   2.72 -----
0.80   0.98   2.31  97.99  70.30
     neff 3194    0     0      0.00   74  12.65   669   1.64  352.5   2.09  10.0
0.50   1.93   0.51  96.03  32.60
   cholik 1303    0     0      0.00    2   0.34   552   1.35  281.9   1.67 -----
1.72   3.07  25.35  99.69  66.70
 jshoemak 2508    1    24    572.22    1   0.17   576   1.41  229.1   1.36 -----
0.55   0.55   3.74  99.20  39.20
     kudo 2324    1     8    163.35    6   1.03  1152   2.82  211.1   1.25 -----
0.12   0.34   1.54  96.77   5.67
   xztang 1835    1     8     18.99 ---- ------ ----- ------  176.3   1.05  10.0 -----
- ------ ------  99.62 ------
   feller 1880    0     0      0.00   17   2.91   170   0.42  148.1   0.88 -----
0.95   1.83   0.14  97.59  84.31
    maxia 2936    0     0      0.00    1   0.17   191   0.47  129.1   0.77   7.5
0.88   0.88   4.49  99.84  69.10
 ktgnov71 2838    0     0      0.00    1   0.17   192   0.47   95.5   0.57 -----
0.53   0.53   0.34  90.07  51.20
```

This example shows a statistical listing of all active users. The top line (User Statistics Initialized...) of the output indicates the timeframe covered by the displayed statistics.

The statistical output is divided into two statistics categories, Running and Completed. Running statistics include information about jobs that are currently running. Completed statistics are compiled using historical information from both running and completed jobs.

The fields are as follows:

Column	Description
UserName	Name of user.
UID	User ID of user.
Jobs	Number of running jobs.
Procs	Number of procs allocated to running jobs.
ProcHours	Number of proc-hours required to complete running jobs.
Jobs*	Number of jobs completed.
%	Percentage of total jobs that were completed by user.
PHReq*	Total proc-hours requested by completed jobs.
%	Percentage of total proc-hours requested by completed jobs that were requested by user.

Column	Description
PHDed	Total proc-hours dedicated to active and completed jobs. The proc-hours dedicated to a job are calculated by multiplying the number of allocated procs by the length of time the procs were allocated, regardless of the job's CPU usage.
%	Percentage of total prochours dedicated that were dedicated by user.
FSTgt	Fairshare target. A user's fairshare target is specified in the `fs.cfg` file. This value should be compared to the user's node-hour dedicated percentage to determine if the target is being met.
AvgXF*	Average expansion factor for jobs completed. A job's XFactor (expansion factor) is calculated by the following formula: (QueuedTime + RunTime) / WallClockLimit.
MaxXF*	Highest expansion factor received by jobs completed.
AvgQH*	Average queue time (in hours) of jobs.
Effic	Average job efficiency. Job efficiency is calculated by dividing the actual node-hours of CPU time used by the job by the node-hours allocated to the job.
WCAcc*	Average wallclock accuracy for jobs completed. Wallclock accuracy is calculated by dividing a job's actual run time by its specified wallclock limit. A job's wallclock accuracy is capped at 100% so even if a job exceeds its requested walltime it will report an accuracy of 100%.

* These fields are empty until a user has completed at least one job.

Example 4-59: Fairshare tree statistics

```
> showstats -T
statistics initialized Mon Jul 10 15:29:41
              |-------- Active ---------|---------------------------------- Completed
-----------------------------------|
user           Jobs Procs ProcHours  Mem Jobs    %    PHReq    %    PHDed    %   FSTgt
  AvgXF  MaxXF  AvgQH  Effic  WCAcc
root              0     0      0.00    0   56 100.00  2.47K 100.00  1.58K  48.87 -----
   1.22   0.00   0.24 100.00  58.84
 l1.1             0     0      0.00    0   25  44.64 845.77  34.31 730.25  22.54 -----
   1.97   0.00   0.20 100.00  65.50
  Administrati    0     0      0.00    0   10  17.86 433.57  17.59 197.17   6.09 -----
   3.67   0.00   0.25 100.00  62.74
  Engineering     0     0      0.00    0   15  26.79 412.20  16.72 533.08  16.45 -----
   0.83   0.00   0.17 100.00  67.35
 l1.2             0     0      0.00    0   31  55.36  1.62K  65.69 853.00  26.33 -----
   0.62   0.00   0.27 100.00  53.46
  Shared          0     0      0.00    0    3   5.36  97.17   3.94  44.92   1.39 -----
   0.58   0.00   0.56 100.00  31.73
  Test            0     0      0.00    0    3   5.36  14.44   0.59  14.58   0.45 -----
```

```
   0.43   0.00   0.17 100.00  30.57
  Research        0     0      0.00    0   25  44.64  1.51K  61.16 793.50  24.49 -----
   0.65   0.00   0.24 100.00  58.82

> showstats -T 2
statistics initialized Mon Jul 10 15:29:41
              |-------- Active ---------|--------------------------------- Completed
----------------------------------|
user           Jobs Procs ProcHours  Mem Jobs    %    PHReq    %    PHDed    %   FSTgt
  AvgXF  MaxXF  AvgQH  Effic  WCAcc
  Test            0     0      0.00    0   22   4.99 271.27   0.55 167.42   0.19 -----
   3.86   0.00   2.89 100.00  60.76
  Shared          0     0      0.00    0   59  13.38 12.30K  24.75  4.46K   5.16 -----
   6.24   0.00  10.73 100.00  49.87
  Research        0     0      0.00    0  140  31.75  9.54K  19.19  5.40K   6.25 -----
   2.84   0.00   5.52 100.00  57.86
  Administrati    0     0      0.00    0   84  19.05  7.94K  15.96  4.24K   4.91 -----
   4.77   0.00   0.34 100.00  62.31
  Engineering     0     0      0.00    0  136  30.84 19.67K  39.56 28.77K  33.27 -----
   3.01   0.00   3.66 100.00  63.70

> showstats -T l1.1
statistics initialized Mon Jul 10 15:29:41
              |-------- Active ---------|--------------------------------- Completed
----------------------------------|
user           Jobs Procs ProcHours  Mem Jobs    %    PHReq    %    PHDed    %   FSTgt
  AvgXF  MaxXF  AvgQH  Effic  WCAcc
 l1.1             0     0      0.00    0  220  49.89 27.60K  55.52 33.01K  38.17 -----
   3.68   0.00   2.39 100.00  63.17
  Administrati    0     0      0.00    0   84  19.05  7.94K  15.96  4.24K   4.91 -----
   4.77   0.00   0.34 100.00  62.31
  Engineering     0     0      0.00    0  136  30.84 19.67K  39.56 28.77K  33.27 -----
   3.01   0.00   3.66 100.00  63.70
```

Example 4-60: Statistics from an absolute time frame

```
> showstats -c batch -v -t 00:00:01_01/01/13,23:59:59_12/31/13
statistics initialized Wed Jan 1 00:00:00

-------- Active ---------       ------------------------------------ Completed ------------
-----------------------

class Jobs Procs ProcHours Mem Jobs     %   PHReq     %   PHDed     %    FSTgt AvgXF
MaxXF  AvgQH  Effic  WCAcc
batch    0     0      0.00   0   23  100.00    15  100.00     1  100.00  -----  0.40
5.01   0.00  88.94  39.87
```

Moab returns information about the class `batch` from January 1, 2013 to December 31, 2013. For more information about specifying absolute dates, see "Absolute Time Format" in **TIMESPEC** on page 270.

Example 4-61: Statistics from a relative time frame

```
> showstats -u bob -v -t -30:00:00:00
statistics initialized Mon Nov 11 15:30:00

-------- Active ---------       ------------------------------------ Completed ------------
-----------------------

user Jobs Procs ProcHours Mem Jobs     %   PHReq     %   PHDed     %    FSTgt AvgXF
MaxXF  AvgQH  Effic  WCAcc
bob      0     0      0.00   0   23  100.00    15  100.00     1  100.00  -----  0.40
5.01   0.00  88.94  39.87
```

Moab returns information about user bob from the past 30 days. For more information about specifying relative dates, see "Relative Time Format" in **TIMESPEC** on page 270.

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mschedctl -f command - re-initialize statistics
- showstats -f command - display full matrix statistics

showstats -f

Synopsis

showstats -f *<statistictype>*

Overview

Shows table of various scheduler statistics.

This command displays a table of the selected Moab Scheduler statistics, such as expansion factor, bypass count, jobs, proc-hours, wallclock accuracy, and backfill information.

Statistics are aggregated over time. This means statistical information is not available for time frames and the `-t` option is not supported with showstats -f.

Access

This command can be run by any Moab Scheduler Administrator.

Parameters

Parameter	Description
AVGBYPASS	Average bypass count. Includes summary of job-weighted expansion bypass and total samples.
AVGQTIME	Average queue time. Includes summary of job-weighted queue time and total samples.
AVGXFACTOR	Average expansion factor. Includes summary of job-weighted expansion factor, processor-weighted expansion factor, processor-hour-weighted expansion factor, and total number of samples.
BFCOUNT	Number of jobs backfilled. Includes summary of job-weighted backfill job percent and total samples.
BFPHRUN	Number of proc-hours backfilled. Includes summary of job-weighted backfill proc-hour percentage and total samples.

Parameter	Description
ESTSTARTTIME	Job start time estimate for jobs meeting specified processor/duration criteria. This estimate is based on the reservation start time analysis algorithm.
JOBCOUNT	Number of jobs. Includes summary of total jobs and total samples.
MAXBYPASS	Maximum bypass count. Includes summary of overall maximum bypass and total samples.
MAXXFACTOR	Maximum expansion factor. Includes summary of overall maximum expansion factor and total samples.
PHREQUEST	proc-hours requested. Includes summary of total proc-hours requested and total samples.
PHRUN	proc-hours run. Includes summary of total proc-hours run and total samples.
QOSDELIVERED	Quality of service delivered. Includes summary of job-weighted quality of service success rate and total samples.
WCACCURACY	Wallclock accuracy. Includes summary of overall wall clock accuracy and total samples.

Examples

Example 4-62:

```
> showstats -f AVGXFACTOR
Average XFactor Grid
[ NODES ][ 00:02:00 ][ 00:04:00 ][ 00:08:00 ][ 00:16:00 ][ 00:32:00 ][ 01:04:00 ][
02:08:00 ][ 04:16:00 ][ 08:32:00 ][ 17:04:00 ][ 34:08:00 ][   TOTAL  ]
[    1  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[    2  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[    4  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][
1.00   1][ -------- ][  1.12   2][ -------- ][ -------- ][  1.10   3]
[    8  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][
1.00   2][  1.24   2][ -------- ][ -------- ][ -------- ][  1.15   4]
[   16  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][  1.01   2][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][  1.01   2]
[   32  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[   64  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[  128  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[  256  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[ T TOT ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][  1.01   2][
1.00   3][  1.24   2][  1.12   2][ -------- ][ -------- ]
Job Weighted X Factor:     1.0888
Node Weighted X Factor:    1.1147
```

```
NS Weighted X Factor:      1.1900
Total Samples:                9
```

The showstats -f command returns a table with data for the specified STATISTICTYPE parameter. The left-most column shows the maximum number of processors required by the jobs shown in the other columns. The column headers indicate the maximum wallclock time (in HH:MM:SS notation) requested by the jobs shown in the columns. The data returned in the table varies by the STATISTICTYPE requested. For table entries with one number, it is of the data requested. For table entries with two numbers, the left number is the data requested and the right number is the number of jobs used to calculate the average. Table entries that contain only dashes (-------) indicate no job has completed that matches the profile associated for this inquiry. The bottom row shows the totals for each column. Following each table is a summary, which varies by the STATISTICTYPE requested.

The column and row break down can be adjusted using the STATPROC and STATTIME* parameters respectively.*

This particular example shows the average expansion factor grid. Each table entry indicates two pieces of information — the average expansion factor for all jobs that meet this slot's profile and the number of jobs that were used to calculate this average. For example, the XFactors of two jobs were averaged to obtain an average XFactor of 1.24 for jobs requiring over 2 hours 8 minutes, but not more than 4 hours 16 minutes and between 5 and 8 processors. Totals along the bottom provide overall XFactor averages weighted by job, processors, and processor-hours.

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes
- mschedctl -f command
- showstats command
- STATPROCMIN parameter
- STATPROCSTEPCOUNT parameter
- STATPROCSTEPSIZE parameter
- STATTIMEMIN parameter
- STATTIMESTEPCOUNT parameter
- STATTIMESTEPSIZE parameter

TIMESPEC

Relative Time Format

The relative time format specifies a time by using the current time as a reference and specifying a time offset.

Format

+[[[DD:]HH:]MM:]SS

Examples

2 days, 3 hours and 57 seconds in the future:

```
+02:03:0:57
```

21 days (3 weeks) in the future:

```
+21:0:0:0
```

30 seconds in the future:

```
+30
```

Absolute Time Format

The absolute time format specifies a specific time in the future.

Format

[HH[:MM[:SS]]][_MO[/DD[/YY]]] ie 14:30_06/20)

Examples

1 PM, March 1 (this year)

```
13:00_03/01
```

4.6.1 Deprecated commands

canceljob

This command is deprecated. Use mjobctl -c instead.

Synopsis

canceljob *jobid* [*jobid*]...

Overview

The canceljob command is used to selectively cancel the specified job(s) (active, idle, or non-queued) from the queue.

Access

This command can be run by any Moab Administrator and by the owner of the job (see ADMINCFG).

Flag	Name	Format	Default	Description	Example
-h	HELP		N/A	Display usage information	`> canceljob -h`
	JOB ID	<STRING>	---	a jobid, a job expression, or the keyword ALL	`> canceljob 13001 13003`

Examples

Example 4-63: Cancel job 6397

```
> canceljob 6397
```

changeparam

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 effect at the beginning of the next scheduling iteration. They are not persistent, only lasting until Moab is shut down.

changeparam is a compact command of mschedctl -m.

Access

This command can be run by a level 1 Moab administrator.

diagnose

This command is deprecated. Use mdiag instead.

Synopsis

diagnose -a [*accountid*]

diagnose -b [-l *policylevel*] [-t *partition*]

diagnose -c [*classid*]

diagnose -C [*configfile*]

diagnose -f [-o user|group|account|qos|class]

diagnose -g [*groupid*]

diagnose -j [*jobid*]

diagnose -L

diagnose -m [*rackid*]

diagnose -n [-t *partition*] [*nodeid*]

diagnose -p [-t *partition*]

diagnose -q [*qosid*]

diagnose -r [*reservationid*]

diagnose -R [*resourcemanagername*]

diagnose -s [*standingreservationid*]

diagnose -S diagnose -u [*userid*]

diagnose -v

diagnose -x

Overview

The diagnose command is used to display information about various aspects of scheduling and the results of internal diagnostic tests.

releasehold

This command is deprecated. Use mjobctl -u instead.

Synopsis

releasehold [-a|-b] jobexp

Overview

Release hold on specified job(s).

This command allows you to release batch holds or all holds (system, user, and batch) on specified jobs. Any number of jobs may be released with this command.

Access

By default, this command can be run by any Moab Scheduler Administrator.

Parameters

JOBEXP	Job expression of job(s) to release.

Flags

-a	Release all types of holds (user, system, batch) for specified job(s).
-b	Release batch hold from specified job(s).
-h	Help for this command.

Examples

Example 4-64: releasehold -b

```
> releasehold -b 6443
batch hold released for job 6443
```

In this example, a batch hold was released from this one job.

Example 4-65: releasehold -a

```
> releasehold -a "81[1-6]"
holds modified for job 811
holds modified for job 812
holds modified for job 813
holds modified for job 814
holds modified for job 815
holds modified for job 816
```

In this example, all holds were released from the specified jobs.

Related topics

- sethold
- mjobctl

releaseres

This command is deprecated. Use mrsvctl -r instead.

Synopsis

releaseres [*arguments*] *reservationid* [*reservationid*...]

Overview

Release existing reservation.

This command allows Moab Scheduler Administrators to release any user, group, account, job, or system reservation. Users are allowed to release reservations on jobs they own. Note that releasing a reservation on an active job has no effect since the reservation will be automatically recreated.

Access

Users can use this command to release any reservation they own. Level 1 and level 2 Moab administrators may use this command to release any reservation.

Parameters

RESERVATION ID	Name of reservation to release.

Examples

Example 4-66: Release two existing reservations

```
> releaseres system.1 bob.2
released User reservation 'system.1'
released User reservation 'bob.2'
```

resetstats

This command is deprecated. Use mschedctl -f instead.

Synopsis

resetstats

Overview

This command resets all internally-stored Moab Scheduler statistics to the initial start-up state as of the time the command was executed.

Access

By default, this command can be run by level 1 scheduler administrators.

Examples

Example 4-67:

```
> resetstats Statistics Reset at time Wed Feb 25 23:24:55 2011
```

Related topics

- Moab Client Installation - explains how to distribute this command to client nodes

runjob

This command is deprecated. Use mjobctl -x instead.

Synopsis

runjob [-c|-f|-n *nodelist*|-p *partition*|-s|-x] *jobid*

Overview

This command will attempt to immediately start the specified job.

runjob is a deprecated command, replaced by mjobctl.

Access

By default, this command can be run by any Moab administrator.

Parameters

JOBID	Name of the job to run.

Args	Description
-c	*Clear* job parameters from previous runs (used to clear PBS neednodes attribute after PBS job launch failure)
-f	Attempt to *force* the job to run, ignoring throttling policies
-n <NODELIST>	Attempt to start the job using the specified *nodelist* where node names are comma or colon delimited
-p <PARTITION>	Attempt to start the job in the specified *partition*
-s	Attempt to *suspend* the job
-x	Attempt to force the job to run, ignoring throttling policies, QoS constraints, and reservations

Examples

Example 4-68: Run job `cluster.231`

```
> runjob cluster.231
job cluster.231 successfully started
```

See Also

- mjobctl
- canceljob - cancel a job.
- checkjob - show detailed status of a job.
- showq - list queued jobs.

sethold

This command is deprecated. Use mjobctl -h instead.

Synopsis

sethold [-b] *jobid* [*jobid*...]

Overview

Set hold on specified job(s).

Permissions

This command can be run by any Moab Scheduler Administrator.

Parameters

JOB	Job number of job to hold.

Flags

-b	Set a batch hold. Typically, only the scheduler places batch holds. This flag allows an administrator to manually set a batch hold.
-h	Help for this command.

Examples

Example 4-69:

```
> sethold -b fr17n02.1072.0 fr15n03.1017.0
Batch Hold Placed on All Specified Jobs
```

In this example, a batch hold is placed on job `fr17n02.1072.0` and job `fr15n03.1017.0`.

setqos

This command is deprecated. Use mjobctl -m instead.

Synopsis

setqos *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.

Examples

Example 4-70:

```
> setqos high_priority moab.3

Job QOS Adjusted
```

This example sets the Quality Of Service to a value of `high_priority` for job `moab.3`.

setres

This command is deprecated. Use mrsvctl -c instead.

Synopsis

setres [*arguments*] *resourceexpression*
[-a <ACCOUNT_LIST>]
[-b <SUBTYPE>]
[-c <CHARGE_SPEC>]
[-d <DURATION>]
[-e <ENDTIME>]
[-E] // EXCLUSIVE
[-f <FEATURE_LIST>]
[-g <GROUP_LIST>]
[-n <NAME>]
[-o <OWNER>]
[-p <PARTITION>]
[-q <QUEUE_LIST>] // (ie CLASS_LIST)
[-Q <QOSLIST>]
[-r <RESOURCE_DESCRIPTION>]
[-R <RESERVATION_PROFILE>]
[-s <STARTTIME>]
[-T <TRIGGER>]
[-u <USER_LIST>]
[-x <FLAGS>]

Overview

Reserve resources for use by jobs with particular credentials or attributes.

Access

This command can be run by level 1 and level 2 Moab administrators.

Parameters

Name	Format	Default	Description
ACCOUNT_LIST	*<STRING>* [:*<STRING>*]...	---	List of accounts that will be allowed access to the reserved resources
SUBTYPE	*<STRING>*	---	Specify the subtype for a reservation
CHARGE_SPEC	*<ACCOUNT>* [,*<GROUP>* [,*<USER>*]]	---	Specifies which credentials will be accountable for unused resources dedicated to the reservation
CLASS_LIST	*<STRING>* [:*<STRING>*]...	---	List of classes that will be allowed access to the reserved resource
DURATION	[[[DD:]HH:]MM:]SS	INFINITY	Duration of the reservation (not needed if ENDTIME is specified)
ENDTIME	[HH[:MM[:SS]]][_ MO[/DD[/YY]]] or +[[[DD:]HH:]MM:] SS	INFINITY	Absolute or relative time reservation will end (not required if Duration specified)
EXCLUSIVE	N/A	N/A	Requests exclusive access to resources
FEATURE_LIST	*<STRING>* [:*<STRING>*]...	---	List of node features which must be possessed by the reserved resources
FLAGS	*<STRING>* [:*<STRING>*]...	---	List of reservation flags (See Managing Reservations for details)
GROUP_LIST	*<STRING>* [:*<STRING>*]...	---	List of groups that will be allowed access to the reserved resources
NAME	*<STRING>*	Name set to first name listed in ACL or `SYSTEM` if no ACL specified	Name for new reservation

Name	Format	Default	Description
OWNER	*<CREDTYPE>* :*<CREDID>* where *CREDTYPE* is one of user, group, acct, class, or qos	N/A	Specifies which credential is granted reservation ownership privileges
PARTITION	*<STRING>*	[ANY]	Partition in which resources must be located
QOS_LIST	*<STRING>* [:*<STRING>*]...	---	List of QoSes that will be allowed access to the reserved resource
RESERVATION_ PROFILE	Existing reservation profile ID	N/A	Requests that default reservation attributes be loaded from the specified reservation profile (see RSVPROFILE)
RESOURCE_ DESCRIPTION	Colon delimited list of zero or more of the following *<ATTR>*=*<VALUE>* pairs PROCS=*<INTEGER>* MEM=*<INTEGER>* DISK=*<INTEGER>* SWAP=*<INTEGER>* GRES=*<STRING>*	PROCS=-1	Specifies the resources to be reserved per task. (-1 indicates all resources on node)
RESOURCE_ EXPRESSION	ALL or TASKS{==|>=} *<TASKCOUNT>* or *<HOST_REGEX>*	Required Field. No Default	Specifies the tasks to reserve. ALL indicates all resources available should be reserved. If ALL or a host expression is specified, Moab will apply the reservation regardless of existing reservations and exclusive issues. If TASKS is used, Moab will only allocate accessible resources.
STARTTIME	[HH[:MM[:SS]]][_ MO[/DD[/YY]]] or +[[[DD:]HH:]MM:] SS	NOW	Absolute or relative time reservation will start

Name	Format	Default	Description
TRIGGER	<STRING>	N/A	Comma delimited reservation trigger list following format described in the trigger format section of the reservation configuration overview.
USER_LIST	<STRING> [:<STRING>]...	---	List of users that will be allowed access to the reserved resources

Description

The setres command allows an arbitrary block of resources to be reserved for use by jobs which meet the specified access constraints. The timeframe covered by the reservation can be specified on either an absolute or relative basis. Only jobs with credentials listed in the reservation ACL (i.e., **USERLIST**, **GROUPLIST**,...) can utilize the reserved resources. However, these jobs still have the freedom to utilize resources outside of the reservation. The reservation will be assigned a name derived from the ACL specified. If no reservation ACL is specified, the reservation is created as a system reservation and no jobs will be allowed access to the resources during the specified timeframe (valuable for system maintenance, etc). See the Reservation Overview for more information.

Reservations can be viewed using the showres command and can be released using the releaseres command.

Examples

Example 4-71:

```
> setres -u john:mary -s +24:00:00 -d 8:00:00 TASKS==2
reservation 'john.1' created on 2 nodes (2 tasks)
node001:1
node005:1
```

Reserve two nodes for use by users `john` and `mary` for a period of 8 hours starting in 24 hours

Example 4-72:

```
> setres -s 8:00:00_06/20 -e 17:00:00_06/22 ALL
reservation 'system.1' created on 8 nodes (8 tasks)
node001:1
node002:1
node003:1
node004:1
node005:1
node006:1
node007:1
node008:1
```

Schedule a system wide reservation to allow system maintenance on Jun 20, 8:00 AM until Jun 22, 5:00 PM.

Example 4-73:

```
> setres -r PROCS=1:MEM=512 -g staff -l interactive 'node00[3-6]'
reservation 'staff.1' created on 4 nodes (4 tasks)
```

```
node003:1
node004:1
node005:1
node006:1
```

Reserve one processor and 512 MB of memory on nodes node003 through node node006 for members of the group staff and jobs in the interactive class.

setspri

This command is deprecated. Use mjobctl -p instead.

Synopsis

setspri [-r] *priorityjobid*

Overview

(This command is deprecated by the mjobctl command)

Set or remove absolute or relative system priorities for a specified job.

This command allows you to set or remove a system priority level for a specified job. Any job with a system priority level set is guaranteed a higher priority than jobs without a system priority. Jobs with higher system priority settings have priority over jobs with lower system priority settings.

Access

This command can be run by any Moab Scheduler Administrator.

Parameters

JOB	Name of job.
PRIORITY	System priority level. By default, this priority is an absolute priority overriding the policy generated priority value. Range is 0 to clear, 1 for lowest, 1000 for highest. The given value is added onto the system priority (see 32-bit and 64-bit values below), except for a given value of zero. If the '-r' flag is specified, the system priority is relative, adding or subtracting the specified value from the policy generated priority. If a relative priority is specified, any value in the range +/- 1,000,000,000 is acceptable.

Flags

-r	Set relative system priority on job.

Examples

Example 4-74:

```
> setspri 10 orion.4752
```

```
job system priority adjusted
```

In this example, a system priority of `10` is set for job `orion.4752`.

Example 4-75:

```
> setspri 0 clusterB.1102
job system priority adjusted
```

In this example, system priority is cleared for job `clusterB.1102`.

Example 4-76:

```
> setspri -r 100000 job.00001
job system priority adjusted
```

In this example, the job's priority will be increased by `100000` over the value determine by configured priority policy.

This command is deprecated. Use mjobctl instead.

showconfig

This command is deprecated. Use mschedctl -l instead.

Synopsis

showconfig [-v]

Overview

View the current configurable parameters of the Moab Scheduler.

The showconfig command shows the current scheduler version and the settings of all "in memory" parameters. These parameters are set via internal defaults, command line arguments, environment variable settings, parameters in the `moab.cfg` file, and via the mschedctl -m command. Because of the many sources of configuration settings, the output may differ from the contents of the `moab.cfg` file. The output is such that it can be saved and used as the contents of the `moab.cfg` file if desired.

Access

This command can be run by a level 1, 2, or 3 Moab administrator.

Flags

-h	Help for this command.
-v	This optional flag turns on verbose mode, which shows all possible Moab Scheduler parameters and their current settings. If this flag is not used, this command operates in context-sensitive terse mode, which shows only relevant parameter settings.

Examples

Example 4-77: showconfig

```
> showconfig
# moab scheduler version 4.2.4 (PID: 11080)
BACKFILLPOLICY                  FIRSTFIT
BACKFILLMETRIC                  NODES
ALLOCATIONPOLICY                MINRESOURCE
RESERVATIONPOLICY               CURRENTHIGHEST
...
```

The showconfig command without the -v flag does not show the settings of all parameters. It does show all major parameters and all parameters which are in effect and have been set to non-default values. However, it hides other rarely used parameters and those which currently have no effect or are set to default values. To show the settings of all parameters, use the -v (verbose) flag. This will provide an extended output. This output is often best used in conjunction with the grep command as the output can be voluminous.

Related topics

- Use the mschedctl -m command to change the various Moab Scheduler parameters.
- See the Parameters document for details about configurable parameters.

5.0 Prioritizing Jobs and Allocating Resources

- **Job Prioritization** on page 285
- **Node Allocation Policies** on page 301
- **Node Access Policies** on page 309
- **Node Availability Policies** on page 311

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.

- Priority Overview
- Job Priority Factors
- **Fairshare Job Priority Example** on page 296
- Common Priority Usage
- Prioritization Strategies
- Manual Priority Management

Related topics

- 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:

<COMPONENT WEIGHT> * <SUBCOMPONENT WEIGHT> * <PRIORITY SUBCOMPONENT VALUE>

Each priority component contains one or more subcomponents as described in the section titled **Job Priority Factors** on page 286. For example, the Resource component consists of Node, Processor, Memory, Swap, Disk, Walltime, and PE subcomponents. While there are numerous priority components and many more subcomponents, a site need only focus on and configure the subset of components related to their particular priority needs. In actual usage, few sites use more than a small fraction (usually 5 or fewer) of the available priority subcomponents. This results in fairly straightforward priority configurations and tuning. By mixing and matching priority weights, sites may generally obtain the desired job-start behavior. At any time, you can issue the mdiag -p command to determine the impact of the current priority-weight settings on idle jobs. Likewise, the command showstats -f can assist the administrator in evaluating priority effectiveness on historical system usage metrics such as queue time or expansion factor.

As mentioned above, a job's priority is the weighted sum of its activated subcomponents. By default, the value of all component and subcomponent weights is set to 1 and 0 respectively. The one exception is the "QUEUETIME" subcomponent weight that is set to 1. This results in a total job priority equal to the period of time the job has been queued, causing Moab to act as a simple FIFO. Once the summed component weight is determined, this value is then bounded resulting in a priority ranging between 0 and MAX_PRIO_VAL which is currently defined as 1000000000 (one billion). In no case will a job obtain a priority in excess of MAX_PRIO_VAL through its priority subcomponent values.

Negative priority jobs may be allowed if desired; see ENABLENEGJOBPRIORITY and REJECTNEGPRIOJOBS for more information.

Using the mjobctl -p command, site administrators may adjust the base calculated job priority by either assigning a relative priority adjustment or an absolute system priority. A relative priority adjustment causes the base priority to be increased or decreased by a specified value. Setting an absolute system priority, SPRIO, causes the job to receive a priority equal to MAX_PRIO_VAL + SPRIO, and thus guaranteed to be of higher value than any naturally occurring job priority.

Related topics

- REJECTNEGPRIOJOBS parameter

5.1.2 Job Priority Factors

- Credential (CRED) Component
- Fairshare (FS) Component
- Resource (RES) Component
- Service (SERVICE) Component
- Target Service (TARG) Component
- Usage (USAGE) Component
- Job Attribute (ATTR) Component

Moab allows jobs to be prioritized based on a range of job related factors. These factors are broken down into a two-tier hierarchy of priority factors and subfactors, each of which can be independently

assigned a weight. This approach provides the administrator with detailed yet straightforward control of the job selection process.

Each factor and subfactor can be configured with independent priority weight and priority cap values (described later). In addition, per credential and per QoS priority weight adjustments may be specified for a subset of the priority factors. For example, QoS credentials can adjust the queuetime subfactor weight and group credentials can adjust fairshare subfactor weight.

The following table highlights the factors and subfactors that make up a job's total priority.

Factor	SubFactor	Metric
CRED (job credentials)	USER	user-specific priority (See USERCFG)
	GROUP	group-specific priority (See GROUPCFG)
	ACCOUNT	account-specific priority (SEE ACCOUNTCFG)
	QOS	QoS-specific priority (See QOSCFG)
	CLASS	class/queue-specific priority (See CLASSCFG)

Factor	SubFactor	Metric
FS (fairshare usage)	FSUSER	user-based historical usage (See Fairshare Overview)
	FSGROUP	group-based historical usage (See Fairshare Overview)
	FSACCOUNT	account-based historical usage (See Fairshare Overview)
	FSQOS	QoS-based historical usage (See Fairshare Overview)
	FSCLASS	class/queue-based historical usage (See Fairshare Overview)
	FSGUSER	imported global user-based historical usage (See ID Manager and Fairshare Overview)
	FSGGROUP	imported global group-based historical usage (See ID Manager and Fairshare Overview)
	FSGACCOUNT	imported global account-based historical usage (See ID Manager and Fairshare Overview)
	FSJPU	current active jobs associated with job user
	FSPPU	current number of processors allocated to active jobs associated with job user
	FSPSPU	current number of processor-seconds allocated to active jobs associated with job user
	WCACCURACY	user's current historical job wallclock accuracy calculated as total processor-seconds dedicated / total processor-seconds requested Factor values are in the range of 0.0 to 1.0.

Factor	SubFactor	Metric
RES (requested job resources)	NODE	number of nodes requested
	PROC	number of processors requested
	MEM	total real memory requested (in MB)
	SWAP	total virtual memory requested (in MB)
	DISK	total local disk requested (in MB)
	PS	total processor-seconds requested
	PE	total processor-equivalent requested
	WALLTIME	total walltime requested (in seconds)
SERV (current service levels)	QUEUETIME	time job has been queued (in minutes)
	XFACTOR	minimum job expansion factor
	BYPASS	number of times job has been bypassed by backfill
	STARTCOUNT	number of times job has been restarted
	DEADLINE	proximity to job deadline
	SPVIOLATION	Boolean indicating whether the active job violates a soft usage limit
	USERPRIO	user-specified job priority
TARGET (target service levels)	TARGETQUEUETIME	time until queuetime target is reached (exponential)
	TARGETXFACTOR	distance to target expansion factor (exponential)

Factor	SubFactor	Metric
USAGE **(consumed resources -- active jobs only)**	CONSUMED	processor-seconds dedicated to date
	REMAINING	processor-seconds outstanding
	PERCENT	percent of required walltime consumed
	EXECUTIONTIME	seconds since job started
ATTR **(job attribute-based prioritization)**	ATTRATTR	Attribute priority if specified job attribute is set (attributes may be user-defined or one of **preemptor**, or **preemptee**). Default is 0.
	ATTRSTATE	Attribute priority if job is in specified state (see Job States). Default is 0.
	ATTRGRES	Attribute priority if a generic resource is requested. Default is 0.

*CAP parameters (FSCAP, for example) are available to limit the maximum absolute value of each priority component and subcomponent. If set to a positive value, a priority cap will bound priority component values in both the positive and negative directions.

All *CAP and *WEIGHT parameters are specified as positive or negative integers. Non-integer values are not supported.

Credential (CRED) Component

The credential component allows a site to prioritize jobs based on political issues such as the relative importance of certain groups or accounts. This allows direct political priorities to be applied to jobs.

The priority calculation for the credential component is as follows:

Priority += CREDWEIGHT * (
 USERWEIGHT * Job.User.Priority +
 GROUPWEIGHT * Job.Group.Priority +
 ACCOUNTWEIGHT * Job.Account.Priority +
 QOSWEIGHT * Job.Qos.Priority +
  CLASSWEIGHT * Job.Class.Priority)

All user, group, account, QoS, and class weights are specified by setting the **PRIORITY** attribute of using the respective *CFG parameter (namely, USERCFG, GROUPCFG, ACCOUNTCFG, QOSCFG, and CLASSCFG).

For example, to set user and group priorities, you might use the following:

```
CREDWEIGHT      1
USERWEIGHT      1
GROUPWEIGHT     1
USERCFG[john]   PRIORITY=2000
USERCFG[paul]   PRIORITY=-1000
GROUPCFG[staff] PRIORITY=10000
```

Class (or queue) priority may also be specified via the resource manager where supported (as in PBS queue priorities). However, if Moab class priority values are also specified, the resource manager priority values will be overwritten.

All priorities may be positive or negative.

Fairshare (FS) Component

Fairshare components allow a site to favor jobs based on short-term historical usage. The Fairshare Overview describes the configuration and use of fairshare in detail.

The fairshare factor is used to adjust a job's priority based on current and historical percentage system utilization of the job's user, group, account, class, or QoS. This allows sites to steer workload toward a particular usage mix across user, group, account, class, and QoS dimensions.

The fairshare priority factor calculation is as follows:

Priority += FSWEIGHT * MIN(FSCAP, (
  FSUSERWEIGHT * DeltaUserFSUsage +
  FSGROUPWEIGHT * DeltaGroupFSUsage +
  FSACCOUNTWEIGHT * DeltaAccountFSUsage +
  FSQOSWEIGHT * DeltaQOSFSUsage +
  FSCLASSWEIGHT * DeltaClassFSUsage +
  FSJPUWEIGHT * ActiveUserJobs +
  FSPPUWEIGHT * ActiceUserProcs +
  FSPSPUWEIGHT * ActiveUserPS +
  WCACCURACYWEIGHT * UserWCAccuracy))

All *WEIGHT parameters just listed are specified on a per partition basis in the `moab.cfg` file. The `Delta*Usage` components represent the difference in actual fairshare usage from the corresponding fairshare usage target. Actual fairshare usage is determined based on historical usage over the time frame specified in the fairshare configuration. The target usage can be a target, floor, or ceiling value as specified in the fairshare configuration file. See the Fairshare Overview for further information on configuring and tuning fairshare. Additional insight may be available in the fairshare usage example. The `ActiveUser*` components represent current usage by the job's user credential.

How violated ceilings and floors affect fairshare-based priority

Moab determines `FSUsageWeight` in the previous section. In order to account for violated ceilings and floors, Moab multiplies that number by the `FSUsagePriority` as demonstrated in the following formula:

```
FSPriority = FSUsagePriority * FSUsageWeight
```

When a ceiling or floor is violated, `FSUsagePriority = 0`, so `FSPriority = 0`. This means the job will gain no priority because of fairshare. If fairshare is the only component of priority, then violation

takes the priority to 0. For more information, see **Priority-Based Fairshare** on page 347 and **Fairshare Targets** on page 344.

Resource (RES) Component

Weighting jobs by the amount of resources requested allows a site to favor particular types of jobs. Such prioritization may allow a site to better meet site mission objectives, improve fairness, or even improve overall system utilization.

Resource based prioritization is valuable when you want to favor jobs based on the resources requested. This is good in three main scenarios: (1) when you need to favor large resource jobs because it's part of your site's mission statement, (2) when you want to level the response time distribution across large and small jobs (small jobs are more easily backfilled and thus generally have better turnaround time), and (3) when you want to improve system utilization. While this may be surprising, system utilization actually increases as large resource jobs are pushed to the front of the queue. This keeps the smaller jobs in the back where they can be selected for backfill and thus increase overall system utilization. The situation is like the story about filling a cup with golf balls and sand. If you put the sand in first, it gets in the way and you are unable to put in as many golf balls. However, if you put in the golf balls first, the sand can easily be poured in around them completely filling the cup.

The calculation for determining the total resource priority factor is as follows:

```
Priority += RESWEIGHT* MIN(RESCAP, (
  NODEWEIGHT * TotalNodesRequested +
  PROCWEIGHT * TotalProcessorsRequested +
  MEMWEIGHT * TotalMemoryRequested +
  SWAPWEIGHT * TotalSwapRequested +
  DISKWEIGHT * TotalDiskRequested +
  WALLTIMEWEIGHT* TotalWalltimeRequested +
  PEWEIGHT * TotalPERequested))
```

The sum of all weighted resources components is then multiplied by the **RESWEIGHT** parameter and capped by the **RESCAP** parameter. Memory, Swap, and Disk are all measured in megabytes (MB). The final resource component, PE, represents Processor Equivalents. This component can be viewed as a processor-weighted maximum *percentage of total resources* factor.

For example, if a job requested 25% of the processors and 50% of the total memory on a 128-processor system, it would have a PE value of MAX(25,50) * 128, or 64. The concept of PEs is a highly effective metric in shared resource systems.

Ideal values for requested job processor count and walltime can be specified using PRIORITYTARGETPROCCOUNT and PRIORITYTARGETDURATION.

Service (SERVICE) Component

The Service component specifies which service metrics are of greatest value to the site. Favoring one service subcomponent over another generally improves that service metric.

The priority calculation for the service priority factor is as follows:

```
Priority += SERVICEWEIGHT * (
  QUEUETIMEWEIGHT * <QUEUETIME> +
  XFACTORWEIGHT * <XFACTOR> +
```

BYPASSWEIGHT * <BYPASSCOUNT> +
STARTCOUNTWEIGHT * <STARTCOUNT> +
DEADLINEWEIGHT * <DEADLINE> +
SPVIOLATIONWEIGHT * <SPBOOLEAN> +
USERPRIOWEIGHT * <USERPRIO>)

QueueTime (QUEUETIME) Subcomponent

In the priority calculation, a job's queue time is a duration measured in minutes. Using this subcomponent tends to prioritize jobs in a FIFO order. Favoring queue time improves queue time based fairness metrics and is probably the most widely used single job priority metric. In fact, under the initial default configuration, this is the only priority subcomponent enabled within Moab. It is important to note that within Moab, a job's queue time is not necessarily the amount of time since the job was submitted. The parameter JOBPRIOACCRUALPOLICY allows a site to select how a job will accrue queue time based on meeting various throttling policies. Regardless of the policy used to determine a job's queue time, this effective queue time is used in the calculation of the QUEUETIME, XFACTOR, TARGETQUEUETIME, and TARGETXFACTOR priority subcomponent values.

The need for a distinct effective queue time is necessitated by the fact that many sites have users who like to work the system, whatever system it happens to be. A common practice at some long existent sites is for some users to submit a large number of jobs and then place them on hold. These jobs remain with a hold in place for an extended period of time and when the user is ready to run a job, the needed executable and data files are linked into place and the hold released on one of these pre-submitted jobs. The extended hold time guarantees that this job is now the highest priority job and will be the next to run. The use of the **JOBPRIOACCRUALPOLICY** parameter can prevent this practice and prevent "queue stuffers" from doing similar things on a shorter time scale. These "queue stuffer" users submit hundreds of jobs at once to swamp the machine and consume use of the available compute resources. This parameter prevents the user from gaining any advantage from stuffing the queue by not allowing these jobs to accumulate any queue time based priority until they meet certain idle and active Moab fairness policies (such as max job per user and max idle job per user).

As a final note, you can adjust the QUEUETIMEWEIGHT parameter on a per QoS basis using the QOSCFG parameter and the **QTWEIGHT** attribute. For example, the line **QOSCFG***[special]* **QTWEIGHT**=*5000* causes jobs using the QoS *special* to have their queue time subcomponent weight increased by 5000.

Expansion Factor (XFACTOR) Subcomponent

The expansion factor subcomponent has an effect similar to the queue time factor but favors shorter jobs based on their requested wallclock run time. In its traditional form, the expansion factor (XFactor) metric is calculated as follows:

XFACTOR = 1 + <QUEUETIME> / <EXECUTIONTIME>

However, a couple of aspects of this calculation make its use more difficult. First, the length of time the job will actually run—<EXECUTIONTIME>—is not actually known until the job completes. All that is known is how much time the job requests. Secondly, as described in the Queue Time Subcomponent section, Moab does not necessarily use the raw time since job submission to determine <QUEUETIME> to prevent various scheduler abuses. Consequently, Moab uses the following modified equation:

XFACTOR = 1 + <EFFQUEUETIME> / <WALLCLOCKLIMIT>

In the equation Moab uses, <EFFQUEUETIME> is the effective queue time subject to the JOBPRIOACCRUALPOLICY parameter and <WALLCLOCKLIMIT> is the user—or system—specified job wallclock limit.

Using this equation, it can be seen that short running jobs will have an XFactor that will grow much faster over time than the xfactor associated with long running jobs. The following table demonstrates this favoring of short running jobs:

Job Queue Time	1 hour	2 hours	4 hours	8 hours	16 hours
XFactor for 1 hour job	1 + (1 / 1) = 2.00	1 + (2 / 1) = 3.00	1 + (4 / 1) = 5.00	1 + (8 / 1) = 9.00	1 + (16 / 1) = 17.0
XFactor for 4 hour job	1 + (1 / 4) = 1.25	1 + (2 / 4) = 1.50	1 + (4 / 4) = 2.00	1 + (8 / 4) = 3.00	1 + (16 / 4) = 5.0

Since XFactor is calculated as a ratio of two values, it is possible for this subcomponent to be almost arbitrarily large, potentially swamping the value of other priority subcomponents. This can be addressed either by using the subcomponent cap XFACTORCAP, or by using the XFMINWCLIMIT parameter. If the latter is used, the calculation for the XFactor subcomponent value becomes:

XFACTOR = 1 + <EFFQUEUETIME> / MAX(<XFMINWCLIMIT>,<WALLCLOCKLIMIT>)

Using the **XFMINWCLIMIT** parameter allows a site to prevent very short jobs from causing the XFactor subcomponent to grow inordinately.

Some sites consider XFactor to be a more fair scheduling performance metric than queue time. At these sites, job XFactor is given far more weight than job queue time when calculating job priority and job XFactor distribution consequently tends to be fairly level across a wide range of job durations. (That is, a flat XFactor distribution of 1.0 would result in a one-minute job being queued on average one minute, while a 24-hour job would be queued an average of 24 hours.)

Like queue time, the effective XFactor subcomponent weight is the sum of two weights, the **XFACTORWEIGHT** parameter and the QoS-specific XFWEIGHT setting. For example, the line **QOSCFG** *[special]* **XFWEIGHT**=*5000* causes jobs using the QoS *special* to increase their expansion factor subcomponent weight by 5000.

Bypass (BYPASS) Subcomponent

The bypass factor is based on the bypass count of a job where the bypass count is increased by one every time the job is bypassed by a lower priority job via backfill. Backfill starvation has never been reported, but if encountered, use the BYPASS subcomponent.

StartCount (STARTCOUNT) Subcomponent

Apply the startcount factor to sites with trouble starting or completing due to policies or failures. The primary causes of an idle job having a startcount greater than zero are resource manager level job start failure, administrator based requeue, or requeue based preemption.

Deadline (DEADLINE) Subcomponent

The deadline factor allows sites to take into consideration the proximity of a job to its DEADLINE. As a jobs moves closer to its deadline its priority increases linearly. This is an alternative to the strict deadline discussed in QOS SERVICE.

Soft Policy Violation (SPVIOLATION) Subcomponent

The soft policy violation factor allows sites to favor jobs which do not violate their associated soft resource limit policies.

User Priority (USERPRIO) Subcomponent

The user priority subcomponent allows sites to consider end-user specified job priority in making the overall job priority calculation. Under Moab, end-user specified priorities may only be negative and are bounded in the range 0 to -1024. See Manual Priority Usage and Enabling End-user Priorities for more information.

User priorities can be positive, ranging from -1024 to 1023, if ENABLEPOSUSERPRIORITY TRUE is specified in moab.cfg.

Target Service (TARG) Component

The target factor component of priority takes into account job scheduling performance targets. Currently, this is limited to target expansion factor and target queue time. Unlike the expansion factor and queue time factors described earlier which increase gradually over time, the target factor component is designed to grow exponentially as the target metric is approached. This behavior causes the scheduler to do essentially all in its power to make certain the scheduling targets are met.

The priority calculation for the target factor is as follows:

```
Priority += TARGETWEIGHT* (
  TARGETQUEUETIMEWEIGHT  * QueueTimeComponent +
  TARGETXFACTORWEIGHT    * XFactorComponent)
```

The queue time and expansion factor target are specified on a per QoS basis using the **XFTARGET** and **QTTARGET** attributes with the QOSCFG parameter. The QueueTime and XFactor component calculations are designed to produce small values until the target value begins to approach, at which point these components grow very rapidly. If the target is missed, this component remains high and continues to grow, but it does not grow exponentially.

Usage (USAGE) Component

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

```
Priority += USAGEWEIGHT * (
  USAGECONSUMEDWEIGHT       * ProcSecondsConsumed +
  USAGEHUNGERWEIGHT       * ProcNeededToBalanceDynamicJob +
  USAGEREMAININGWEIGHT      * ProcSecRemaining +
  USAGEEXECUTIONTIMEWEIGHT  * SecondsSinceStart +
  USAGEPERCENTWEIGHT        * WalltimePercent )
```

Job Attribute (ATTR) Component

The Attribute component allows the incorporation of job attributes into a job's priority. The most common usage for this capability is to do one of the following:

- adjust priority based on a job's state (favor suspended jobs)
- adjust priority based on a job's requested node features (favor jobs that request attribute **pvfs**)
- adjust priority based on internal job attributes (disfavor **backfill** or **preemptee** jobs)
- adjust priority based on a job's requested licenses, network consumption, or generic resource requirements

To use job attribute based prioritization, the JOBPRIOF parameter must be specified to set corresponding attribute priorities. To favor jobs based on node feature requirements, the parameter NODETOJOBATTRMAP must be set to map node feature requests to job attributes.

The priority calculation for the attribute priority factor is as follows:

Priority += ATTRWEIGHT * (
  ATTRATTRWEIGHT * <ATTRPRIORITY> +
  ATTRSTATEWEIGHT * <STATEPRIORITY> +
  ATTRGRESWEIGHT * <GRESPRIORITY>
  JOBIDWEIGHT * <JOBID> +
  JOBNAMEWEIGHT * <JOBNAME_INTEGER>)

Example 5-1:

```
ATTRWEIGHT      100
ATTRATTRWEIGHT    1
ATTRSTATEWEIGHT   1
ATTRGRESWEIGHT    5
# favor suspended jobs
# disfavor preemptible jobs
# favor jobs requesting 'matlab'

JOBPRIOF STATE[Running]=100  STATE[Suspended]=1000  ATTR[PREEMPTEE]=-200  ATTR[gpfs]
=30  GRES[matlab]=400
# map node features to job features

NODETOJOBATTRMAP  gpfs,pvfs
...
```

Related topics

- Node Allocation Priority
- Per Credential Priority Weight Offsets
- Managing Consumable Generic Resources

5.1.3 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.

5.1.4 Common Priority Usage

- Credential Priority Factors
- Service Level Priority Factors
- Priority Factor Caps
- User Selectable Prioritization

Site administrators vary widely in their preferred manner of prioritizing jobs. Moab's scheduling hierarchy allows sites to meet job control needs without requiring adjustments to dozens of parameters. Some choose to use numerous subcomponents, others a few, and still others are content with the default FIFO behavior. Any subcomponent that is not of interest may be safely ignored.

Credential Priority Factors

To help clarify the use of priority weights, a brief example may help. Suppose a site wished to maintain the FIFO behavior but also incorporate some credential based prioritization to favor a special user. Particularly, the site would like the user *john* to receive a higher initial priority than all other users. Configuring this behavior requires two steps. First, the user credential subcomponent must be enabled and second, *john* must have his relative priority specified. Take a look at the sample `moab.cfg` file:

```
USERWEIGHT       1
USERCFG[john]    PRIORITY=300
```

The "USER" priority subcomponent was enabled by setting the USERWEIGHT parameter. In fact, the parameters used to specify the weights of all components and subcomponents follow this same "*WEIGHT" naming convention (as in RESWEIGHT and TARGETQUEUETIMEWEIGHT.

The second part of the example involves specifying the actual user priority for the user *john*. This is accomplished using the USERCFG parameter. Why was the priority 300 selected and not some other value? Is this value arbitrary? As in any priority system, actual priority values are meaningless, only relative values are important. In this case, we are required to balance user priorities with the default queue time based priorities. Since queuetime priority is measured in minutes queued, the user priority of 300 places a job by user *john* on par with a job submitted 5 minutes earlier by another user.

Is this what the site wants? Maybe, maybe not. At the onset, most sites are uncertain what they want in prioritization. Often, an estimate initiates prioritization and adjustments occur over time. Cluster resources evolve, the workload evolves, and even site policies evolve, resulting in changing priority needs over time. Anecdotal evidence indicates that most sites establish a relatively stable priority policy within a few iterations and make only occasional adjustments to priority weights from that point.

Service Level Priority Factors

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

- favor jobs in the low, medium, and high QoSes so they will run in QoS order
- balance job expansion factor
- use job queue time to prevent jobs from starving

Under such conditions, the sample `moab.cfg` file might appear as follows:

```
QOSWEIGHT                1
XFACTORWEIGHT            1
QUEUETIMEWEIGHT         10
TARGETQUEUETIMEWEIGHT    1
QOSCFG[low]         PRIORITY=1000
QOSCFG[medium]      PRIORITY=10000
QOSCFG[high]        PRIORITY=100000
QOSCFG[DEFAULT]     QTTARGET=4:00:00
```

This example is a bit more complicated but is more typical of the needs of many sites. The desired QoS weightings are established by enabling the QoS subfactor using the QOSWEIGHT parameter while the various QoS priorities are specified using QOSCFG. XFACTORWEIGHT is then set as this subcomponent tends to establish a balanced distribution of expansion factors across all jobs. Next, the queuetime component is used to gradually raise the priority of all jobs based on the length of time they have been queued. Note that in this case, QUEUETIMEWEIGHT was explicitly set to 10, overriding its default value of 1. Finally, the TARGETQUEUETIMEWEIGHT parameter is used in conjunction with the USERCFG line to specify a queue time target of 4 hours.

Priority Factor Caps

Assume now that the site administrator is content with this priority mix but has a problem with users submitting large numbers of very short jobs. Very short jobs would tend to have rapidly growing XFactor values and would consequently quickly jump to the head of the queue. In this case, a factor cap would be appropriate. Such caps allow a site to limit the contribution of a job's priority factor to be within a defined range. This prevents certain priority factors from swamping others. Caps can be applied to either priority components or subcomponents and are specified using the *<COMPONENTNAME>*CAP parameter (such as **QUEUETIMECAP**, **RESCAP**, and SERVCAP). Note that both component and subcomponent caps apply to the pre-weighted value, as in the following equation:

```
Priority =
      C1WEIGHT * MIN(C1CAP,SUM(
        S11WEIGHT * MIN(S11CAP,S11S) +
        S12WEIGHT * MIN(S12CAP,S12S) +
        ...)) +
      C2WEIGHT * MIN(C2CAP,SUM(
        S21WEIGHT * MIN(S21CAP,S21S) +
        S22WEIGHT * MIN(S22CAP,S22S) +
        ...)) +
      ...
```

Example 5-2: Priority cap

```
QOSWEIGHT              1
QOSCAP                 10000
XFACTORWEIGHT          1
XFACTORCAP             1000
QUEUETIMEWEIGHT        10
QUEUETIMECAP           1000
```

User Selectable Prioritization

Moab allows users to specify a job priority to jobs they own or manage. This priority may be set at job submission time or it may be dynamically modified (using setspri or mjobctl) after submitting the job.

For fairness reasons, users may only apply a negative priority to their job and thus slide it further back in the queue. This enables users to allow their more important jobs to run before their less important ones without gaining unfair advantage over other users.

User priorities can be positive if ENABLEPOSUSERPRIORITY TRUE is specified in moab.cfg.

In order to set **ENABLEPOSUSERPRIORITY**, you must change the USERPRIOWEIGHT from its default value of 0. For example:

```
USERPRIOWEIGHT   100
```

```
> setspri -r 100 332411
successfully modified job priority
```

Specifying a user priority at job submission time is resource manager specific. See the associated resource manager documentation for more information.

User Selectable Priority w/QoS

Using the QoS facility, organizations can set up an environment in which users can more freely select the desired priority of a given job. Organizations may enable access to a number of QoSes each with its own charging rate, priority, and target service levels. Users can then assign job importance by selecting the appropriate QoS. If desired, this can allow a user to jump ahead of other users in the queue if they are willing to pay the associated costs.

Related topics

- User Selectable Priority

5.1.5 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.

5.1.6 Manual Job Priority Adjustment

Batch administrator's regularly find a need to adjust the calculated priority of a job to meet current needs. Current needs often are broken into two categories:

1. The need to run an administrator test job as soon as possible.
2. The need to pacify a disserviced user.

You can use the setspri command to handle these issues in one of two ways; this command allows the specification of either a relative priority adjustment or the specification of an absolute priority. Using absolute priority specification, administrators can set a job priority guaranteed to be higher than any calculated value. Where Moab-calculated job priorities are in the range of 0 to 1 billion, system administrator assigned absolute priorities start at 1 billion and go up. Issuing the `setspri <PRIO> <JOBID>` command, for example, assigns a priority of 1 billion + <PRIO> to the job. Thus, `setspri 5 job.1294` sets the priority of "job.1294" to 1000000005.

For more information, see Common Priority Usage - End-user Adjustment.

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 NODEALLOCATIONPOLICY. There are multiple node allocation policies to choose from allowing selection based on reservation constraints, node configuration, resource usage, prefed other factors. You can specify these policies with a system-wide default value, on a per-partition basis, or on a per-job basis. Please note that **LASTAVAILABLE** is the default policy.

Available algorithms are described in detail in the following sections and include FIRSTAVAILABLE, LASTAVAILABLE, PRIORITY, CPULOAD, MINRESOURCE, CONTIGUOUS, MAXBALANCE, and PLUGIN.

- Node Allocation Overview
  - Heterogeneous Resources
  - Shared Nodes
  - Reservations or Service Guarantees
  - Non-flat Network
- **Node selection factors** on page 305

- Resource-Based Algorithms
  - CPULOAD
  - FIRSTAVAILABLE
  - LASTAVAILABLE
  - PRIORITY
  - MINRESOURCE
  - CONTIGUOUS
  - MAXBALANCE
- User-Defined Algorithms
  - PLUGIN
- Specifying Per Job Resource Preferences
  - Specifying Resource Preferences
  - Selecting Preferred Resources

Node Allocation Overview

Node allocation is the process of selecting the best resources to allocate to a job from a list of available resources. Making this decision intelligently is important in an environment that possesses one or more of the following attributes:

- heterogeneous resources (resources which vary from node to node in terms of quantity or quality)
- shared nodes (nodes may be utilized by more than one job)
- reservations or service guarantees
- non-flat network (a network in which a perceptible performance degradation may potentially exist depending on workload placement)

Heterogeneous Resources

Moab analyzes job processing requirements and assigns resources to maximize hardware utility.

For example, suppose two nodes are available in a system, A and B. Node A has 768 MB of RAM and node B has 512 MB. The next two jobs in the queue are X and Y. Job X requests 256 MB and job Y requests 640 MB. Job X is next in the queue and can fit on either node, but Moab recognizes that job Y (640 MB) can only fit on node A (768 MB). Instead of putting job X on node A and blocking job Y, Moab can put job X on node B and job Y on node A.

Shared Nodes

Symmetric Multiprocessing (SMP)

When sharing SMP-based compute resources amongst tasks from more than one job, resource contention and fragmentation issues arise. In SMP environments, the general goal is to deliver maximum system

utilization for a combination of compute-intensive and memory-intensive jobs while preventing overcommitment of resources.

By default, most current systems do not do a good job of logically partitioning the resources (such as CPU, memory, and network bandwidth) available on a given node. Consequently contention often arises between tasks of independent jobs on the node. This can result in a slowdown for all jobs involved, which can have significant ramifications if large-way parallel jobs are involved. Virtualization, CPU sets, and other techniques are maturing quickly as methods to provide logical partitioning within shared resources.

On large-way SMP systems (> 32 processors/node), job packing can result in intra-node fragmentation. For example, take two nodes, A and B, each with 64 processors. Assume they are currently loaded with various jobs and A has 24 and B has 12 processors free. Two jobs are submitted; job X requests 10 processors and job Y requests 20 processors. Job X can start on either node but starting it on node A prevents job Y from running. An algorithm to handle intra-node fragmentation is straightforward for a single resource case, but the algorithm becomes more involved when jobs request a combination of processors, memory, and local disk. These workload factors should be considered when selecting a site's node allocation policy as well as identifying appropriate policies for handling resource utilization limit violations.

Interactive Nodes

In many cases, sites are interested in allowing multiple users to simultaneously use one or more nodes for interactive purposes. Workload is commonly not compute intensive consisting of intermittent tasks including coding, compiling, and testing. Because these jobs are highly varied in terms of resource usage over time, sites are able to pack a larger number of these jobs onto the same node. Consequently, a common practice is to restrict job scheduling based on utilized, rather than dedicated resources.

Interactive Node Example

The example configuration files that follow show one method by which node sharing can be accomplished within a TORQUE + Moab environment. This example is based on a hypothetical cluster composed of 4 nodes each with 4 cores. For the compute nodes, job tasks are limited to actual cores preventing overcommitment of resources. For the interactive nodes, up to 32 job tasks are allowed, but the node also stops allowing additional tasks if either memory is fully utilized or if the CPU load exceeds 4.0. Thus, Moab continues packing the interactive nodes with jobs until carrying capacity is reached.

Example 5-3: /opt/moab/etc/moab.cfg

```
# constrain interactive jobs to interactive nodes
# constrain interactive jobs to 900 proc-seconds
CLASSCFG[interactive]  HOSTLIST=interactive01,interactive02
CLASSCFG[interactive]  MAX.CPUTIME=900
RESOURCELIMITPOLICY    CPUTIME:ALWAYS:CANCEL
# base interactive node allocation on load and jobs
NODEALLOCATIONPOLICY  PRIORITY
NODECFG[interactive01] PRIORITYF='-20*LOAD - JOBCOUNT'
NODECFG[interactive02] PRIORITYF='-20*LOAD - JOBCOUNT'
```

Example 5-4: /var/spool/torque/server_priv/nodes

```
interactive01 np=32
interactive02 np=32
```

```
compute01     np=4
compute02     np=4
```

Example 5-5: /var/spool/torque/mom_priv/config on "interactive01"

```
# interactive01
$max_load 4.0
```

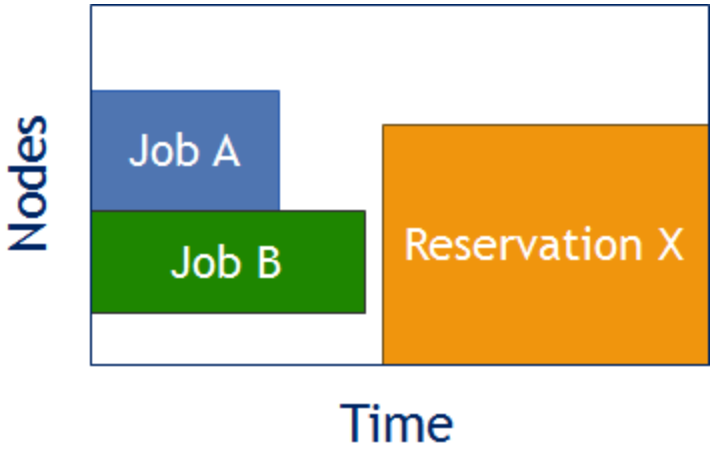
Example 5-6: /var/spool/torque/mom_priv/config on "interactive02"

```
# interactive02
$max_load 4.0
```

Reservations or Service Guarantees

A reservation-based system adds the time dimension into the node allocation decision. With reservations, node resources must be viewed in a type of two dimension node-time space. Allocating nodes to jobs fragments this node-time space and makes it more difficult to schedule jobs in the remaining, more constrained node-time slots. Allocation decisions should be made in such a way as to minimize this fragmentation and maximize the scheduler's ability to continue to start jobs in existing slots. The following figure shows that job A and job B are running. A reservation, X, is created some time in the future. Assume that job A is 2 hours long and job B is 3 hours long. Again, two new single-processor jobs are submitted, C and D; job C requires 3 hours of compute time while job D requires 5 hours. Either job will just fit in the free space located above job A or in the free space located below job B. If job C is placed above job A, job D, requiring 5 hours of time will be prevented from running by the presence of reservation X. However, if job C is placed below job B, job D can still start immediately above job A.

Image 5-1: Job A, Job B, and Reservation X scheduled on nodes



The preceding example demonstrates the importance of time based reservation information in making node allocation decisions, both at the time of starting jobs and at the time of creating reservations. The impact of time based issues grows significantly with the number of reservations in place on a given system. The **LASTAVAILABLE** algorithm works on this premise, locating resources that have the smallest space between the end of a job under consideration and the start of a future reservation.

Non-flat Network

On systems where network connections do not resemble a flat all-to-all topology, task placement may impact performance of communication intensive parallel jobs. If latencies and network bandwidth between any two nodes vary significantly, the node allocation algorithm should attempt to pack tasks of a given job as close to each other as possible to minimize impact of bandwidth and latency differences.

Node selection factors

While the node allocation policy determines which nodes a job will use, other factors narrow the options before the policy makes the final decision. The following process demonstrates how Moab executes its node allocation process and how other policies affect the decision:

1. Moab eliminates nodes that do not meet the hard resource requirements set by the job.
2. Moab gathers affinity information, first from workload proximity rules and then from reservation affinity rules (See **Affinity** on page 410 for more information.). Reservation affinity rules trump workload proximity rules.
3. Moab allocates nodes using the allocation policy.
   - If more than enough nodes with Required affinity exist, only they are passed down for the final sort by the node allocation policy.
   - If the number of nodes with Required affinity matches the number of nodes requested exactly, then the node allocation policy is skipped entirely and all of those nodes are assigned to the job.
   - If too few nodes have Required affinity, all of them are assigned to the job, then the node allocation policy is applied to the remaining eligible nodes (after Required, Moab will use Positive, then Neutral, then Negative.).

Resource-Based Algorithms

Moab contains a number of allocation algorithms that address some of the needs described earlier. You can also create allocation algorithms and interface them with the Moab scheduling system. Each of these policies has a name and descriptive alias. They can be configured using either one, but Moab will only report their names.

If **ENABLEHIGHTHROUGHPUT** on page 824 is *TRUE*, you must set **NODEALLOCATIONPOLICY** on page 872 to *FIRSTAVAILABLE*.

The current suite of algorithms is described in what follows:

Allocation algorithm name	Alias	Description
CPULOAD	ProcessorLoad	Nodes are selected that have the maximum amount of available, unused CPU power (<#of CPU's> - <CPU load>). CPULOAD is a good algorithm for timesharing node systems and applies to jobs starting immediately. For the purpose of future reservations, the MINRESOURCE algorithm is used.
FIRSTAVAILABLE	InReportedOrder	Simple first come, first served algorithm where nodes are allocated in the order they are presented by the resource manager. This is a very simple, and very fast algorithm.
LASTAVAILABLE	InReserveReportedOrder	Nodes are allocated in descending order that they are presented by the resource manager, or the reverse of FIRSTAVAILABLE.

Allocation algorithm name	Alias	Description
PRIORITY	CustomPriority	Allows a site to specify the priority of various static and dynamic aspects of compute nodes and allocate them with preference for higher priority nodes. It is highly flexible allowing node attribute and usage information to be combined with reservation affinity. Using node allocation priority, you can specify the following priority components: • *ADISK* - Local disk currently available to batch jobs in MB. • *AMEM* - Real memory currently available to batch jobs in MB. • *APROCS* - Processors currently available to batch jobs on node (configured procs - dedicated procs). • *ARCH[<ARCH>]* - Processor architecture. • *ASWAP* - Virtual memory currently available to batch jobs in MB. • *CDISK* - Total local disk allocated for use by batch jobs in MB. • *CMEM* - Total real memory on node in MB. • *COST* - Based on node **CHARGERATE**. • *CPROCS* - Total processors on node. • *CSWAP* - Total virtual memory configured on node in MB. • *FEATURE[<FNAME>]* - Boolean; specified feature is present on node. • *FREETIME* - FREETIME is calculated as the time during which there is no reservation on the machine. It uses either the job wallclock limit (if there is a job), or 2 months. The more free time a node has within either the job wallclock limit or 2 months, the higher this value will be. • *GMETRIC[<GMNAME>]* - Current value of specified generic metric on node. • *JOBCOUNT* - Number of jobs currently running on node. • *JOBFREETIME* - The number of seconds that the node is idle between now and when the job is scheduled to start. • *LOAD* - Current 1 minute load average. • *MTBF* - Mean time between failures (in seconds). • *NODEINDEX* - Node's nodeindex as specified by the resource manager. • *OS* - True if job compute requirements match node operating system. • *PARAPROCS* - Processors currently available to batch jobs within partition (configured procs - dedicated procs). • *POWER* - TRUE if node is ON. • *PREF* - Boolean; node meets job specific resource preferences. • *PRIORITY* - Administrator specified node priority. • *RANDOM* - Per iteration random value between 0 and 1. (Allows introduction of random allocation factor.) Regardless of coefficient, the contribution of this

Allocation algorithm name	Alias	Description

		Example 5: Pack tasks onto nodes with the most processors available and the lowest CPU temperature.
		`RMCFG[torque] TYPE=pbs`
		`RMCFG[temp] TYPE=NATIVE`
		`CLUSTERQUERYURL=exec://$TOOLSDIR/hwmon.pl`
		`NODEALLOCATIONPOLICY PRIORITY`
		`NODECFG[DEFAULT] PRIORITYF='100*APROCS - GMETRIC[temp]`
		`'`
		`...`

MINRESOURCE	MinimumConfiguredResources	Prioritizes nodes according to the configured resources on each node. Those nodes with the fewest configured resources that still meet the job's resource constraints are selected.
CONTIGUOUS	Contiguous	Allocates nodes in contiguous (linear) blocks as required by the Compaq RMS system.
MAXBALANCE	ProcessorSpeedBalance	Attempts to allocate the most balanced set of nodes possible to a job. In most cases, but not all, the metric for balance of the nodes is node procspeed. Thus, if possible, nodes with identical procspeeds are allocated to the job. If identical procspeed nodes cannot be found, the algorithm allocates the set of nodes with the minimum node procspeed span or range.

User-Defined Algorithms

User-defined algorithms allow administrators to define their own algorithms based on factors such as their system's network topology. When node allocation is based on topology, jobs finish faster, administrators see better cluster productivity and users pay less for resources.

PLUGIN

This algorithm allows administrators to define their own node allocation policy and create a plug-in that allocates nodes based on factors such as a cluster's network topology. This has the following advantages:

- plug-ins keep the source code of the cluster's interconnect network for node allocation separate from Moab's source code (customers can implement plug-ins independent of Moab's release schedule)
- plug-ins can be independently created and tailored to specific hardware and network topology
- plug-ins can be modified without assistance from Adaptive Computing, Inc.

Specifying *Per Job* Resource Preferences

While the resource based node allocation algorithms can make a good guess at what compute resources would best satisfy a job, sites often possess a subset of jobs that benefit from more explicit resource allocation specification. For example one job may perform best on a particular subset of nodes due to direct access to a tape drive, another may be very memory intensive. Resource preferences are distinct from node requirements. While the former describes what a job needs to run at all, the latter describes what the job needs to run well. In general, a scheduler must satisfy a job's node requirement specification and then satisfy the job's resource preferences as well as possible.

Specifying Resource Preferences

A number of resource managers natively support the concept of resource preferences (such as Loadleveler). When using these systems, the language specific preferences keywords may be used. For systems that do not support resource preferences natively, Moab provides a resource manager extension keyword, "PREF," which you can use to specify desired resources. This extension allows specification of node features, memory, swap, and disk space conditions that define whether the node is considered preferred.

Moab 5.2 (and earlier) only supports feature-based preferences.

Selecting Preferred Resources

Enforcing resource preferences is not completely straightforward. A site may have a number of potentially conflicting requirements that the scheduler is asked to simultaneously satisfy. For example, a scheduler may be asked to maximize the proximity of the allocated nodes at the same time it is supposed to satisfy resource preferences and minimize node overcommitment. To allow site specific weighting of these varying requirements, Moab allows resource preferences to be enabled through the **PRIORITY** node allocation algorithm. For example, to use resource preferences together with node load, the following configuration might be used:

```
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT]     PRIORITYF='5 * PREF - LOAD'
...
```

To request specific resource preferences, a user could then submit a job indicating those preferences. In the case of a PBS job, the following can be used:

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

Related topics

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

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:

Policy	Description
SHARED	Tasks from any combination of jobs may use available resources.
SHAREDONLY	Only jobs requesting shared node access may use available resources.
SINGLEACCOUNT	Tasks from any jobs owned by the same account may use available resources.
SINGLEGROUP	Tasks from any jobs owned by the same group may use available resources.
SINGLEJOB	Only tasks from a single job may use available resources.
SINGLETASK	Only a single task from a single job may run on the node.
SINGLEUSER	Tasks from any jobs owned by the same user may use available resources.
UNIQUEUSER	Any number of tasks from a single job may allocate resources from a node but only if the user has no other jobs running on that node. **UNIQUEUSER** limits the number of jobs a single user can run on a node, allowing other users to run jobs with the remaining resources. This policy is useful in environments where job epilog/prologs scripts are used to clean up processes based on userid.

Configuring Node Access Policies

The global node access polices may be specified via the parameter NODEACCESSPOLICY. This global default may be overridden on a per node basis with the ACCESS attribute of the NODECFG parameter or on a per job basis using the resource manager extension NACCESSPOLICY. Finally, a per queue node access policy may also be specified by setting either the NODEACCESSPOLICY or FORCENODEACCESSPOLICY attributes of the CLASSCFG parameter. **FORCENODEACCESSPOLICY** overrides any per job specification in all cases, whereas **NODEACCESSPOLICY** is overridden by per job specification.

By default, nodes are accessible using the setting of the system wide **NODEACCESSPOLICY** parameter unless a specific ACCESS policy is specified on a per node basis using the **NODECFG** parameter. Jobs may override this policy and subsequent jobs are bound to conform to the access policies of all jobs currently running on a given node. For example, if the **NODEACCESSPOLICY** parameter is set to **SHARED**, a new job may be launched on an idle node with a job specific access policy of **SINGLEUSER**. While this job runs, the effective node access policy changes to **SINGLEUSER** and subsequent job tasks may only be launched on this node provided they are submitted by the same user. When all single user jobs have completed on that node, the effective node access policy reverts back to **SHARED** and the node can again be used in **SHARED** mode.

For example, to set a global policy of **SINGLETASK** on all nodes except nodes 13 and 14, use the following:

```
# by default, enforce dedicated node access on all nodes
NODEACCESSPOLICY  SINGLETASK
# allow nodes 13 and 14 to be shared
NODECFG[node13]   ACCESS=SHARED
NODECFG[node14]   ACCESS=SHARED
```

Related topics

- Per job naccesspolicy specification via Resource Manager Extensions
- JOBNODEMATCHPOLICY parameter
- NODEAVAILABILITY parameter

5.4 Node Availability Policies

- Node Resource Availability Policies
- Node Categorization
- Node Failure/Performance Based Notification
- Node Failure/Performance Based Triggers
- Handling Transient Node Failures
- **Allocated Resource Failure Policy for Jobs** on page 315

Moab enables several features relating to node availability. These include policies that determine how per node resource availability should be reported, how node failures are detected, and what should be done in the event of a node failure.

Node Resource Availability Policies

Moab allows a job to be launched on a given compute node as long as the node is not full or busy. The NODEAVAILABILITYPOLICY parameter allows a site to determine what criteria constitute a node being busy. The legal settings are listed in the following table:

Availability Policy	Description
DEDICATED	The node is considered busy if dedicated resources equal or exceed configured resources.
UTILIZED	The node is considered busy if utilized resources equal or exceed configured resources.
COMBINED	The node is considered busy if either dedicated or utilized resources equal or exceed configured resources.

The default setting for all nodes is **COMBINED**, indicating that a node can accept workload so long as the jobs that the node was allocated to do not request or use more resources than the node has available. In

a load balancing environment, this may not be the desired behavior. Setting the NODEAVAILABILITYPOLICY parameter to **UTILIZED** allows jobs to be packed onto a node even if the aggregate resources requested exceed the resources configured. For example, assume a scenario with a 4-processor compute node and 8 jobs requesting 1 processor each. If the resource availability policy was set to **COMBINED**, this node would only allow 4 jobs to start on this node even if the jobs induced a load of less than 1.0 each. With the resource availability policy set to **UTILIZED**, the scheduler continues allowing jobs to start on the node until the node's load average exceeds a per processor load value of 1.0 (in this case, a total load of 4.0). To prevent a node from being over populated within a single scheduling iteration, Moab artificially raises the node's load for one scheduling iteration when starting a new job. On subsequent iterations, the actual measured node load information is used.

Per Resource Availability Policies

By default, the NODEAVAILABILITYPOLICY sets a global per node resource availability policy. This policy applies to all resource types on each node such as processors, memory, swap, and local disk. However, the syntax of this parameter is as follows:

```
<POLICY>[:<RESOURCETYPE>] ...
```

This syntax allows per resource availability specification. For example, consider the following:

```
NODEAVAILABILITYPOLICY  DEDICATED:PROC COMBINED:MEM COMBINED:DISK
...
```

This configuration causes Moab to only consider the quantity of processing resources actually dedicated to active jobs running on each node and ignore utilized processor information (such as CPU load). For memory and disk, both utilized resource information and dedicated resource information should be combined to determine what resources are actually available for new jobs.

Node Categorization

Moab allows organizations to detect and use far richer information regarding node status than the standard batch "idle," "busy," "down states" commonly found. Using node categorization, organizations can record, track, and report on per node and cluster level status including the following categories:

Category	Description
Active	Node is healthy and currently executing batch workload.
BatchFailure	Node is unavailable due to a failure in the underlying batch system (such as a resource manager server or resource manager node daemon).
Benchmark	Node is reserved for benchmarking.
EmergencyMaintenance	Node is reserved for unscheduled system maintenance.

Category	Description
HardwareFailure	Node is unavailable due to a failure in one or more aspects of its hardware configuration (such as a power failure, excessive temperature, memory, processor, or swap failure).
HardwareMaintenance	Node is reserved for scheduled system maintenance.
Idle	Node is healthy and is currently not executing batch workload.
JobReservation	Node is reserved for job use.
NetworkFailure	Node is unavailable due to a failure in its network adapter or in the switch.
Other	Node is in an uncategorized state.
OtherFailure	Node is unavailable due to a general failure.
PersonalReservation	Node is reserved for dedicated use by a personal reservation.
Site[1-8]	Site specified usage categorization.
SoftwareFailure	Node is unavailable due to a failure in a local software service (such as automounter, security or information service such as NIS, local databases, or other required software services).
SoftwareMaintenance	Node is reserved for software maintenance.
StandingReservation	Node is reserved by a standing reservation.
StorageFailure	Node is unavailable due to a failure in the cluster storage system or local storage infrastructure (such as failures in Lustre, GPFS, PVFS, or SAN).
UserReservation	Node is reserved for dedicated use by a particular user or group and may or may not be actively executing jobs.

Node categories can be explicitly assigned by cluster administrators using the mrsvctl -c command to create a reservation and associate a category with that node for a specified timeframe. Further, outside of this explicit specification, Moab automatically mines all configured interfaces to learn about its environment and the health of the resources it is managing. Consequently, Moab can identify many hardware failures, software failures, and batch failures without any additional configuration. However, it is often desirable to make additional information available to Moab to allow it to integrate this information into reports; automatically notify managers, users, and administrators; adjust internal

policies to steer workload around failures; and launch various custom triggers to rectify or mitigate the problem.

You can specify the FORCERSVSUBTYPE parameter to require all administrative reservations be associated with a node category at reservation creation time. For example:

```
NODECFG[DEFAULT] ENABLEPROFILING=TRUE
FORCERSVSUBTYPE  TRUE
```

Node health and performance information from external systems can be imported into Moab using the native resource manager interface. This is commonly done using generic metrics or consumable generic resources for performance and node categories or node variables for status information. Combined with arbitrary node messaging information, Moab can combine detailed information from remote services and report this to other external services.

Use the NODECATCREDLIST parameter to generate extended node category based statistics.

Node Failure/Performance Based Notification

Moab can be configured to cause node failures and node performance levels that cross specified thresholds to trigger notification events. This is accomplished using the GEVENTCFG parameter as described in the Generic Event Overview section. For example, the following configuration can be used to trigger an email to administrators each time a node is marked down.

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

Node Failure/Performance Based Triggers

Moab supports per node triggers that can be configured to fire when specific events are fired or specific thresholds are met. These triggers can be used to modify internal policies or take external actions. A few examples follow:

- decrease node allocation priority if node throughput drops below threshold X
- launch local diagnostic/recovery script if parallel file system mounts become stale
- reset high performance network adapters if high speed network connectivity fails
- create general system reservation on node if processor or memory failure occurs

As mentioned, Moab triggers can be used to initiate almost any action, from sending mail to updating a database, to publishing data for an SNMP trap, to driving a web service.

Handling Transient Node Failures

Since Moab actively schedules both current and future actions of the cluster, it is often important for it to have a reasonable estimate of when failed nodes will be again available for use. This knowledge is particularly useful for proper scheduling of new jobs and management of resources in regard to backfill. With backfill, Moab determines which resources are available for priority jobs and when the highest

priority idle jobs can run. If a node experiences a failure, Moab should have a concept of when this node will be restored.

When Moab analyzes down nodes for allocation, one of two issues may occur with the highest priority jobs. If Moab believes that down nodes will not be recovered for an extended period of time, a transient node failure within a reservation for a priority job may cause the reservation to slide far into the future allowing other lower priority jobs to allocate and launch on nodes previously reserved for it. Moments later, when the transient node failures are resolved, Moab may be unable to restore the early reservation start time as other jobs may already have been launched on previously available nodes.

In the reverse scenario, if Moab recognizes a likelihood that down nodes will be restored too quickly, it may make reservations for top priority jobs that allocate those nodes. Over time, Moab slides those reservations further into the future as it determines that the reserved nodes are not being recovered. While this does not delay the start of the top priority jobs, these unfulfilled reservations can end up blocking other jobs that should have properly been backfilled and executed.

Creating Automatic Reservations

If a node experiences occasional transient failures (often not associated with a node state of down), Moab can automatically create a temporary reservation over the node to allow the transient failure time to clear and prevent Moab from attempting to re-use the node while the failure is active. This reservation behavior is controlled using the NODEFAILURERESERVETIME parameter as in the following example:

```
# reserve nodes for 1 minute if transient failures are detected
NODEFAILURERESERVETIME  00:01:00
```

Blocking Out Down Nodes

If one or more resource managers identify failures and mark nodes as down, Moab can be configured to associate a default *unavailability* time with this failure and the node state *down*. This is accomplished using the NODEDOWNSTATEDELAYTIME parameter. This delay time floats and is measured as a fixed time into the future from the time " NOW"; it is not associated with the time the node was originally marked down. For example, if the delay time was set to 10 minutes, and a node was marked down 20 minutes ago, Moab would still consider the node unavailable until 10 minutes into the future.

While it is difficult to select a good default value that works for all clusters, the following is a general rule of thumb:

- Increase **NODEDOWNSTATEDELAYTIME** if jobs are getting blocked due to priority reservations sliding as down nodes are not recovered.
- Decrease **NODEDOWNSTATEDELAYTIME** if high priority job reservations are getting regularly delayed due to transient node failures.

```
# assume down nodes will not be recovered for one hour
NODEDOWNSTATEDELAYTIME  01:00:00
```

Allocated Resource Failure Policy for Jobs

If a failure occurs within a collection of nodes allocated to a job, Moab can automatically re-allocate replacement resources. For jobs, this can be configured with JOBACTIONONNODEFAILURE.

How an active job behaves when one or more of its allocated resources fail depends on the allocated resource failure policy. Depending on the type of job, type of resources, and type of middleware

infrastructure, a site may choose to have different responses based on the job, the resource, and the type of failure.

Failure Responses

By default, Moab cancels a job when an allocated resource failure is detected. However, you can specify the following actions:

Option	Policy action
CANCEL	Cancels the job
FAIL	Terminates the job as a failed job
HOLD	Places a hold on the job. This option is only applicable if you are using checkpointing
IGNORE	Ignores the failed node, allowing the job to proceed
NOTIFY	Notifies the administrator and user of failure but takes no further action
REQUEUE	Requeues job and allows it to run when alternate resources become available

Policy Precedence

For a given job, the applied policy can be set at various levels with policy precedence applied in the job, class/queue, partition, and then system level. The following table indicates the available methods for setting this policy:

Object	Parameter	Example
Job	RESFAILPOLICY resource manager extension	`> qsub -l resfailpolicy=requeue`
Class/Queue	RESFAILPOLICY attribute of CLASSCFG parameter	`CLASSCFG[batch] RESFAILPOLICY=CANCEL`
Partition	JOBACTIONONNODE FAILURE attribute of PARCFG parameter	`PARCFG[web3] JOBACTIONONNODEFAILURE=NOTIFY`
System	NODEALLOCRESFAILURE POLICY parameter	`NODEALLOCRESFAILUREPOLICY=MIGRATE`

Failure Definition

Any allocated node going down constitutes a failure. However, for certain types of workload, responses to failures may be different depending on whether it is the master task (task 0) or a slave task that fails. To indicate that the associated policy should only take effect if the master task fails, the allocated resource failure policy should be specified with a trailing asterisk (*), as in the following example:

```
CLASSCFG[virtual_services] RESFAILPOLICY=requeue*
```

TORQUE Failure Details

When a node fails becoming unresponsive, the resource manager central daemon identifies this failure within a configurable time frame (default: 60 seconds). Detection of this failure triggers an event that causes Moab to immediately respond. Based on the specified policy, Moab notifies administrators, holds the job, requeues the job, allocates replacement resources to the job, or cancels the job. If the job is canceled or requeued, Moab sends the request to TORQUE, which immediately frees all non-failed resources making them available for use by other jobs. Once the failed node is recovered, it contacts the resource manager central daemon, determines that the associated job has been canceled/requeued, cleans up, and makes itself available for new workload.

Related topics

- Node State Overview
- JOBACTIONONNODEFAILURE parameter
- NODEFAILURERESERVETIME parameter
- NODEDOWNSTATEDELAYTIME parameter (down nodes will be marked unavailable for the specified duration)
- NODEDRAINSTATEDELAYTIME parameter (offline nodes will be marked unavailable for the specified duration)
- NODEBUSYSTATEDELAYTIME parameter (nodes with unexpected background load will be marked unavailable for the specified duration)
- NODEALLOCRESFAILUREPOLICY parameter (action to take if executing jobs have one or more allocated nodes fail)

6.0 Managing Fairness - Throttling Policies, Fairshare, and Allocation Management

- **Fairness Overview** on page 319
- **Usage Limits/Throttling Policies** on page 322
- **Fairshare** on page 340
- **Charging and Allocation Management** on page 354
- **Charging a Workflow** on page 369
- **NAMI Queuing** on page 372

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:

Job Prioritization	
Description:	Specifies what is most important to the scheduler. Using service based priority factors allows a site to balance job turnaround time, expansion factor, or other scheduling performance metrics.

Example:	`SERVICEWEIGHT 1`
	`QUEUETIMEWEIGHT 10`
	Causes jobs to increase in priority by 10 points for every minute they remain in the queue.

Usage Limits (Throttling Policies)

Description: Specifies limits on exactly what resources can be used at any given instant.

Example:

```
USERCFG[john]     MAXJOB=3
GROUPCFG[DEFAULT] MAXPROC=64
GROUPCFG[staff]   MAXPROC=128
```

Allows john to only run 3 jobs at a time. Allows the group staff to use up to 128 total processors and all other groups to use up to 64 processors.

Fairshare

Description: Specifies usage targets to limit resource access or adjust priority based on historical cluster resource usage.

Example:

```
USERCFG[steve] FSTARGET=25.0+
FSWEIGHT       1
FSUSERWEIGHT   10
```

Enables priority based fairshare and specifies a fairshare target for user steve such that his jobs are favored in an attempt to keep his jobs using at least 25.0% of delivered compute cycles.

Allocation Management

Description: Specifies long term, credential-based resource usage limits.

Example:

```
AMCFG[mam] TYPE=MAM HOST=server.sys.net
```

Enables the Moab Accounting Manager allocation management interface. Within the allocation manager, project or account based allocations may be configured. These allocations may, for example, do such things as allow project X to use up to 100,000 processor-hours per quarter, provide various QoS sensitive charge rates, and share allocation access.

Quality of Service

Description: Specifies additional resource and service access for particular users, groups, and accounts. QoS facilities can provide special priorities, policy exemptions, reservation access, and other benefits (as well as special charge rates).

Quality of Service

Example:

```
QOSCFG[orion] PRIORITY=1000 XFTARGET=1.2
QOSCFG[orion] QFLAGS=PREEMPTOR,IGNSYSTEM,RESERVEALWAYS
```

Enables jobs requesting the orion QoS a priority increase, an expansion factor target to improve response time, the ability to preempt other jobs, an exemption from system level job size policies, and the ability to always reserve needed resources if it cannot start immediately.

Standing Reservations

Description: Reserves blocks of resources within the cluster for specific, periodic time frames under the constraints of a flexible access control list.

Example:

```
SRCFG[jupiter] HOSTLIST=node01[1-4]
SRCFG[jupiter] STARTTIME=9:00:00 ENDTIME=17:00:00
SRCFG[jupiter] USERLIST=john,steve ACCOUNTLIST=jupiter
```

Reserve nodes node011 through node014 from 9:00 AM until 5:00 PM for use by jobs from user john or steve or from the project jupiter.

Class/Queue Constraints

Description: Associates users, resources, priorities, and limits with cluster classes or cluster queues that can be assigned to or selected by end-users.

Example:

```
CLASSCFG[long] MIN.WCLIMIT=24:00:00
SRCFG[jupiter] PRIORITY=10000
SRCFG[jupiter] HOSTLIST=acn[1-4][0-9]
```

Assigns long jobs a high priority but only allow them to run on certain nodes.

Selecting the Correct Policy Approach

Moab supports a rich set of policy controls in some cases allowing a particular policy to be enforced in more than one way. For example, cycle distribution can be controlled using usage limits, fairshare, or even queue definitions. Selecting the most correct policy depends on site objectives and needs; consider the following when making such a decision:

- Minimal end-user training
    - Does the solution use an approach familiar to or easily learned by existing users?

- End-user transparency
    - Can the configuration be enabled or disabled without impacting user behavior or job submission?
- Impact on system utilization and system responsiveness
- Solution complexity
    - Is the impact of the configuration readily intuitive, and is it easy to identify possible side effects?
- Solution extensibility and flexibility
    - Will the proposed approach allow the solution to be easily tuned and extended as cluster needs evolve?

Related topics

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

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.

- Fairness via Throttling Policies
    - Basic Fairness Policies
    - Multi-Dimension Fairness Policies
- Override Limits
- Idle Job Limits
- Hard and Soft Limits
- Per-partition Limits
- **Usage-based Limits** on page 337
    - Configuring Actions
    - Specifying Hard and Soft Policy Violations

- Constraining Walltime Usage

Fairness via Throttling Policies

Moab allows sign ificant flexibility with usage limits, or throttling policies. At a high level, Moab allows resource usage limits to be specified in three primary workload categories: (1) active, (2) idle, and (3) system job limits.

Basic Fairness Policies

Workload category	Description
Active job limits	Constrain the total cumulative resources available to active jobs at a given time.
Idle job limits	Constrain the total cumulative resources available to idle jobs at a given time.
System job limits	Constrain the maximum resource requirements of any single job.

These limits can be applied to any job credential (user, group, account, QoS, and class), or on a system-wide basis. Using the keyword *DEFAULT*, a site may also specify the default setting for the desired user, group, account, QoS, and class. Additionally, you may configure QoS to allow limit overrides to any particular policy.

To run, a job must meet all policy limits. Limits are applied using the *CFG set of parameters, particularly USERCFG, GROUPCFG, ACCOUNTCFG, QOSCFG, CLASSCFG, and SYSCFG. Limits are specified by associating the desired limit to the individual or default object. The usage limits currently supported are listed in the following table.

MAXARRAYJOB	
Units	Number of simultaneous active array job sub-jobs.
Description	Limits the number of simultaneously active (starting or running) array sub-jobs a credential can have.
Example	`USERCFG[gertrude] MAXARRAYJOB=10` *Gertrude can have a maximum of 10 active job array sub-jobs.*

MAXGRES	
Units	# of concurrent uses of a generic resource

MAXGRES	
Description	Limits the concurrent usage of a generic resource to a specific quantity or quantity range.
Example	`USERCFG[joe] MAXGRES[matlab]=2` `USERCFG[jim] MAXGRES[matlab]=2,4`

MAXJOB	
Units	# of jobs
Description	Limits the number of jobs a credential may have active (starting or running) at any given time. Moab places a hold on all new jobs submitted by that credential once it has reached its maximum number of allowable jobs. `MAXJOB=0` is not supported. You can, however, achieve similar results by using the HOLD attribute of the USERCFG parameter: `USERCFG[john] HOLD=yes`
Example	`USERCFG[DEFAULT] MAXJOB=8` `GROUPCFG[staff]  MAXJOB=2,4`

MAXMEM	
Units	total memory in MB
Description	Limits the total amount of dedicated memory (in MB) that can be allocated by a credential's active jobs at any given time.
Example	`ACCOUNTCFG[jasper] MAXMEM=2048`

MAXNODE	
Units	# of nodes

MAXNODE	
Description	Limits the total number of compute nodes that can be in use by active jobs at any given time. On some systems (including TORQUE/PBS), nodes have been softly defined rather than strictly defined; that is, a job may request 2 nodes but TORQUE will translate this request into 1 node with 2 processors. This can prevent Moab from enforcing a **MAXNODE** policy correctly for a single job. Correct behavior can be achieved using **MAXPROC**.
Example	`CLASSCFG[batch] MAXNODE=64`

MAXPE	
Units	# of processor equivalents
Description	Limits the total number of dedicated processor-equivalents that can be allocated by active jobs at any given time.
Example	`QOSCFG[base] MAXPE=128`

MAXPROC	
Units	# of processors
Description	Limits the total number of dedicated processors that can be allocated by active jobs at any given time per credential. To set **MAXPROC** per job, use msub -W.
Example	`CLASSCFG[debug] MAXPROC=32`

MAXPS	
Units	<# of processors> * <walltime>
Description	Limits the number of outstanding processor-seconds a credential may have allocated at any given time. For example, if a user has a 4-processor job that will complete in 1 hour and a 2-processor job that will complete in 6 hours, they have 4 * 1 * 3600 + 2 * 6 * 3600 = 16 * 3600 outstanding processor-seconds. The outstanding processor-second usage of each credential is updated each scheduling iteration, decreasing as jobs approach their completion time.

MAXPS	
Example	`USERCFG[DEFAULT] MAXPS=720000`

MAXSUBMITJOBS	
Units	# of jobs
Description	Limits the number of jobs a credential may submit and have in the system at once. Moab will reject any job submitted beyond this limit. If you use a TORQUE resource manager, you should also set `max_user_queuable` in case the user submits jobs via qsub instead of msub. See "Queue attributes" in the *TORQUE Administrator Guide* for more information.
Example	`USERCFG[DEFAULT] MAXSUBMITJOBS=5`

MAXWC	
Units	job duration [[[DD:]HH:]MM:]SS
Description	Limits the cumulative remaining walltime a credential may have associated with active jobs. It behaves identically to the **MAXPS** on page 325 limit (listed earlier) only lacking the processor weighting. Like MAXPS, the cumulative remaining walltime of each credential is also updated each scheduling iteration. MAXWC does not limit the maximum wallclock limit per job. For this capability, use **MAX.WCLIMIT** on page 65.
Example	`USERCFG[ops] MAXWC=72:00:00`

The following example demonstrates a simple limit specification:

```
USERCFG[DEFAULT]  MAXJOB=4
USERCFG[john]     MAXJOB=8
```

This example allows user john to run up to 8 jobs while all other users may only run up to 4.

Simultaneous limits of different types may be applied per credential and multiple types of credentials may have limits specified. The next example demonstrates this mixing of limits and is a bit more complicated.

```
USERCFG[steve]    MAXJOB=2 MAXNODE=30
GROUPCFG[staff]   MAXJOB=5
```

```
CLASSCFG[DEFAULT] MAXNODE=16
CLASSCFG[batch]   MAXNODE=32
```

This configuration may potentially apply multiple limits to a single job. As discussed previously, a job may only run if it satisfies all applicable limits. Thus, in this example, the scheduler will be constrained to allow at most *2* simultaneous user *steve* jobs with an aggregate node consumption of no more than *30* nodes. However, if the job is submitted to a class other than *batch*, it may be limited further. Here, only *16* total nodes may be used simultaneously by jobs running in any given class with the exception of the class *batch*. If *steve* submitted a job to run in the class *interactive*, for example, and there were jobs already running in this class using a total of 14 nodes, his job would be blocked unless it requested 2 or fewer nodes by the default limit of *16* nodes per class.

Multi-Dimension Fairness Policies and Per Credential Overrides

Multi-dimensional fairness policies allow a site to specify policies based on combinations of job credentials. A common example might be setting a maximum number of jobs allowed per queue per user or a total number of processors per group per QoS. As with basic fairness policies, multi-dimension policies are specified using the *CFG parameters or through the identity manager interface. Moab supports the most commonly used multi-dimensional fairness policies (listed in the table below) using the following format:

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

*CFG is one of **USERCFG**, **GROUPCFG**, **ACCOUNTCFG**, **QOSCFG**, or **CLASSCFG**, the **<LIMITTYPE>** policy is one of the policies listed in the table in section 6.2.1.1, and **<CRED>** is of the format *<CREDTYPE>[:<VALUE>]* with *CREDTYPE* being one of *USER*, *GROUP*, *ACCT*, *QoS*, or *CLASS*. The optional *<VALUE>* setting can be used to specify that the policy only applies to a specific credential value. For example, the following configuration sets limits on the class *fast*, controlling the maximum number of jobs any group can have active at any given time and the number of processors in use at any given time for user *steve*.

```
CLASSCFG[fast] MAXJOB[GROUP]=12
CLASSCFG[fast] MAXPROC[USER:steve]=50
CLASSCFG[fast] MAXIJOB[USER]=10
```

The following example configuration may clarify further:

```
# allow class batch to run up the 3 simultaneous jobs
# allow any user to use up to 8 total nodes within class
CLASSCFG[batch] MAXJOB=3 MAXNODE[USER]=8
# allow users steve and bob to use up to 3 and 4 total processors respectively within
class
CLASSCFG[fast] MAXPROC[USER:steve]=3 MAXPROC[USER:bob]=4
```

Multi-dimensional policies cannot be applied on *DEFAULT* credentials.

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

	Account limits	Class limits	Group limits	QoS limits	User limits
MAXIJOB	**QoS**: MAXIJOB[QOS]=hlmt MAXIJOB[QOS:qosname]=hlmt			**Account**: MAXIJOB[ACCT]=hlmt MAXIJOB[ACCT:acctname]=hlmt	
MAXIPROC	**QoS**: MAXIPROC[QOS]=hlmt MAXIPROC[QOS:qosname]=hlmt			**Account**: MAXIPROC[ACCT]=hlmt MAXIPROC[ACCT:acctname]=hlmt	
MAXJOB	**Class**: MAXJOB[CLASS]=slmt,hlmt MAXJOB[CLASS:classname]=hlmt **Group**: MAXJOB[GROUP]=slmt,hlmt MAXJOB[GROUP:groupname]=hlmt **QoS**: MAXJOB[QOS]=slmt,hlmt MAXJOB[QOS:qosname]=hlmt **User**: MAXJOB[USER]=slmt,hlmt MAXJOB[USER:username]=hlmt	**Account**: MAXJOB[ACCT]=slmt,hlmt MAXJOB[ACCT:acctname]=hlmt **Group**: MAXJOB[GROUP]=slmt,hlmt MAXJOB[GROUP:groupname]=hlmt **QoS**: MAXJOB[QOS:qosname]=hlmt **User**: MAXJOB[USER]=slmt,hlmt MAXJOB[USER:username]=hlmt	**Account**: MAXJOB[ACCT]=slmt,hlmt MAXJOB[ACCT:acctname]=hlmt **Class**: MAXJOB[CLASS]=slmt,hlmt MAXJOB[CLASS:classname]=hlmt **User**: MAXJOB[USER]=slmt,hlmt MAXJOB[USER:username]=hlmt	**Account**: MAXJOB[ACCT]=slmt,hlmt MAXJOB[ACCT:acctname]=hlmt **Class**: MAXJOB[CLASS]=slmt,hlmt MAXJOB[CLASS:acctname]=hlmt **Group**: MAXJOB[GROUP]=slmt,hlmt MAXJOB[GROUP:groupname]=hlmt **User**: MAXJOB[USER]=slmt,hlmt MAXJOB[USER:username]=hlmt	**Account**: MAXJOB[ACCT]=slmt,hlmt MAXJOB[ACCT:acctname]=hlmt **Class**: MAXJOB[CLASS]=slmt,hlmt MAXJOB[CLASS:classname]=hlmt **Group**: MAXJOB[GROUP]=slmt,hlmt MAXJOB[GROUP:groupname]=hlmt **QoS**: MAXJOB[QOS]=slmt,hlmt MAXJOB[QOS:qosname]=hlmt

	Account limits	Class limits	Group limits	QoS limits	User limits
MAXMEM	**Class**: MAXMEM [CLASS] =slmt,hlmt MAXMEM [CLASS:classname]=hlmt **Group**: MAXMEM [GROUP] =slmt,hlmt MAXMEM [GROUP:groupname]=hlmt **QoS**: MEM[USER] =slmt,hlmt MAXMEM [USER:username]=hlmt **User**: MAXMEM[USER] =slmt,hlmt MAXMEM [USER:username]=hlmt	**Account**: MAXMEM[ACCT] =slmt,hlmt MAXMEM [ACCT:acctname] =hlmt **Group**: MAXMEM [GROUP] =slmt,hlmt MAXMEM [GROUP:groupname]=hlmt **QoS**: MAXMEM [QOS:qosname] =hlmt **User**: MAXMEM[USER] =slmt,hlmt MAXMEM [USER:username]=hlmt	**Account**: MAXMEM [ACCT] =slmt,hlmt MAXMEM [ACCT:acctname]=hlmt **Class**: MAXMEM [CLASS] =slmt,hlmt MAXMEM [CLASS:classname]=hlmt **User**: MAXMEM [USER] =slmt,hlmt MAXMEM [USER:username]=hlmt	**Account**: MAXMEM[ACCT] =slmt,hlmt MAXMEM [ACCT:acctname] =hlmt **Class**: MAXMEM [CLASS] =slmt,hlmt MAXMEM [CLASS:classname]=hlmt **Group**: MAXMEM [GROUP] =slmt,hlmt MAXMEM [GROUP:groupname]=hlmt **User**: MAXMEM[USER] =slmt,hlmt MAXMEM [USER:username]=hlmt	**Account**: MAXMEM[ACCT] =slmt,hlmt MAXMEM [ACCT:acctname] =hlmt **Class**: MAXMEM [CLASS] =slmt,hlmt MAXMEM [CLASS:classname]=hlmt **Group**: MAXMEM [GROUP] =slmt,hlmt MAXMEM [GROUP:groupname]=hlmt **QoS**: MAXMEM[QOS] =slmt,hlmt MAXMEM [QOS:qosname] =hlmt

	Account limits	Class limits	Group limits	QoS limits	User limits
MAXNODE	**Class**: MAXNODE [CLASS] =slmt,hlmt MAXNODE [CLASS:classname]=hlmt **Group**: MAXNODE [GROUP] =slmt,hlmt MAXNODE [GROUP:groupname]=hlmt **QoS**: MAXNODE[QOS] =slmt,hlmt MAXNODE [QOS:qosname] =hlmt **User**: MAXNODE [USER] =slmt,hlmt MAXNODE [USER:username]=hlmt	**Account**: MAXNODE [ACCT] =slmt,hlmt MAXNODE [ACCT:acctname] =hlmt **Group**: MAXNODE [GROUP] =slmt,hlmt MAXNODE [GROUP:groupname]=hlmt **QoS**: MAXNODE [QOS:qosname] =hlmt **User**: MAXNODE [USER] =slmt,hlmt MAXNODE [USER:username]=hlmt	**Account**: MAXNODE [ACCT] =slmt,hlmt MAXNODE [ACCT:acctname]=hlmt **Class**: MAXNODE [CLASS] =slmt,hlmt MAXNODE [CLASS:classname]=hlmt **User**: MAXNODE [USER] =slmt,hlmt MAXNODE [USER:username]=hlmt	**Account**: MAXNODE [ACCT] =slmt,hlmt MAXNODE [ACCT:acctname] =hlmt **Class**: MAXNODE [CLASS] =slmt,hlmt MAXNODE [CLASS:classname]=hlmt **Group**: MAXNODE [GROUP] =slmt,hlmt MAXNODE [GROUP:groupname]=hlmt **User**: MAXNODE [USER] =slmt,hlmt MAXNODE [USER:username]=hlmt	**Account**: MAXNODE [ACCT] =slmt,hlmt MAXNODE [ACCT:acctname] =hlmt **Class**: MAXNODE [CLASS] =slmt,hlmt MAXNODE [CLASS:classname]=hlmt **Group**: MAXNODE [GROUP] =slmt,hlmt MAXNODE [GROUP:groupname]=hlmt **QoS**: MAXNODE[QOS] =slmt,hlmt MAXNODE [QOS:qosname] =hlmt

	Account limits	Class limits	Group limits	QoS limits	User limits
MAXPROC	**Class**: MAXPROC[CLASS]=slmt,hlmt MAXPROC[CLASS:classname]=hlmt **Group**: MAXPROC[GROUP]=slmt,hlmt MAXPROC[GROUP:groupname]=hlmt **QoS**: MAXPROC[QOS]=slmt,hlmt MAXPROC[QOS:qosname]=hlmt **User**: MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=hlmt	**Account**: MAXPROC[ACCT]=slmt,hlmt MAXPROC[ACCT:acctname]=hlmt **Group**: MAXPROC[GROUP]=slmt,hlmt MAXPROC[GROUP:groupname]=hlmt **QoS**: MAXPROC[QOS:qosname]=hlmt **User**: MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=hlmt	**Account**: MAXPROC[ACCT]=slmt,hlmt MAXPROC[ACCT:acctname]=hlmt **Class**: MAXPROC[CLASS]=slmt,hlmt MAXPROC[CLASS:classname]=hlmt **User**: MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=hlmt	**Account**: MAXPROC[ACCT]=slmt,hlmt MAXPROC[ACCT:acctname]=hlmt **Class**: MAXPROC[CLASS]=slmt,hlmt MAXPROC[CLASS:classname]=hlmt **Group**: MAXPROC[GROUP]=slmt,hlmt MAXPROC[GROUP:groupname]=hlmt **User**: MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=hlmt	**Account**: MAXPROC[ACCT]=slmt,hlmt MAXPROC[ACCT:acctname]=hlmt **Class**: MAXPROC[CLASS]=slmt,hlmt MAXPROC[CLASS:classname]=hlmt **Group**: MAXPROC[GROUP]=slmt,hlmt MAXPROC[GROUP:groupname]=hlmt **QoS**: MAXPROC[QOS]=slmt,hlmt MAXPROC[QOS:qosname]=hlmt

	Account limits	Class limits	Group limits	QoS limits	User limits
MAXPS	**Class**: MAXPS[CLASS]=slmt,hlmt MAXPS[CLASS:classname]=hlmt **Group**: MAXPS[GROUP]=slmt,hlmt MAXPS[GROUP:groupname]=hlmt **QoS**: MAXPS[QOS]=slmt,hlmt MAXPS[QOS:qosname]=hlmt **User**: MAXPS[USER]=slmt,hlmt MAXPS[USER:username]=hlmt	**Account**: MAXPS[ACCT]=slmt,hlmt MAXPS[ACCT:acctname]=hlmt **Group**: MAXPS[GROUP]=slmt,hlmt MAXPS[GROUP:groupname]=hlmt **QoS**: MAXPS[QOS:qosname]=hlmt **User**: MAXPS[USER]=slmt,hlmt MAXPS[USER:username]=hlmt	**Account**: MAXPS[ACCT]=slmt,hlmt MAXPS[ACCT:acctname]=hlmt **Class**: MAXPS[CLASS]=slmt,hlmt MAXPS[CLASS:classname]=hlmt **User**: MAXPS[USER]=slmt,hlmt MAXPS[USER:username]=hlmt	**Account**: MAXPS[ACCT]=slmt,hlmt MAXPS[ACCT:acctname]=hlmt **Class**: MAXPS[CLASS]=slmt,hlmt MAXPS[CLASS:classname]=hlmt **Group**: MAXPS[GROUP]=slmt,hlmt MAXPS[GROUP:groupname]=hlmt **User**: MAXPS[USER]=slmt,hlmt MAXPS[USER:username]=hlmt	**Account**: MAXPS[ACCT]=slmt,hlmt MAXPS[ACCT:acctname]=hlmt **Class**: MAXPS[CLASS]=slmt,hlmt MAXPS[CLASS:classname]=hlmt **Group**: MAXPS[GROUP]=slmt,hlmt MAXPS[GROUP:groupname]=hlmt **QoS**: MAXPS[QOS]=slmt,hlmt MAXPS[QOS:qosname]=hlmt

	Account limits	Class limits	Group limits	QoS limits	User limits
MAXWC	**Class**: MAXWC[CLASS]=slmt,hlmt MAXWC[CLASS:classname]=hlmt **Group**: MAXWC[GROUP]=slmt,hlmt MAXWC[GROUP:groupname]=hlmt **QoS**: MAXWC[QOS]=slmt,hlmt MAXWC[QOS:qosname]=hlmt **User**: MAXWC[USER]=slmt,hlmt MAXWC[USER:username]=hlmt	**Account**: MAXWC[ACCT]=slmt,hlmt MAXWC[ACCT:acctname]=hlmt **Group**: MAXWC[GROUP]=slmt,hlmt MAXWC[GROUP:groupname]=hlmt **QoS**: MAXWC[QOS:qosname]=hlmt **User**: MAXWC[USER]=slmt,hlmt MAXWC[USER:username]=hlmt	**Account**: MAXWC[ACCT]=slmt,hlmt MAXWC[ACCT:acctname]=hlmt **Class**: MAXWC[CLASS]=slmt,hlmt MAXWC[CLASS:classname]=hlmt **User**: MAXWC[USER]=slmt,hlmt MAXWC[USER:username]=hlmt	**Account**: MAXWC[ACCT]=slmt,hlmt MAXWC[ACCT:acctname]=hlmt **Class**: MAXWC[CLASS]=slmt,hlmt MAXWC[CLASS:classname]=hlmt **Group**: MAXWC[GROUP]=slmt,hlmt MAXWC[GROUP:groupname]=hlmt **User**: MAXWC[USER]=slmt,hlmt MAXWC[USER:username]=hlmt	**Account**: MAXWC[ACCT]=slmt,hlmt MAXWC[ACCT:acctname]=hlmt **Class**: MAXWC[CLASS]=slmt,hlmt MAXWC[CLASS:classname]=hlmt **Group**: MAXWC[GROUP]=slmt,hlmt MAXWC[GROUP:groupname]=hlmt **QoS**: MAXWC[QOS]=slmt,hlmt MAXWC[QOS:qosname]=hlmt

Override Limits

Like all job credentials, the QoS object may be associated with resource usage limits. However, this credential can also be given special override limits that supersede the limits of other credentials, effectively causing all other limits of the same type to be ignored. See QoS Usage Limits and Overrides for a complete list of policies that can be overridden. The following configuration provides an example of this in the last line:

```
USERCFG[steve]     MAXJOB=2    MAXNODE=30
GROUPCFG[staff]    MAXJOB=5
CLASSCFG[DEFAULT] MAXNODE=16
CLASSCFG[batch]    MAXNODE=32
QOSCFG[hiprio]     OMAXJOB=3   OMAXNODE=64
```

*Only 3 hiprio QoS jobs may run simultaneously and hiprio QoS jobs may run with up to 64 nodes per credential ignoring other credential **MAXNODE** limits.*

Given the preceding configuration, assume a job is submitted with the credentials, user *steve*, group *staff*, class *batch*, and QoS *hiprio*.

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

- Total nodes used by user *steve* does not exceed *64*.
- Total active jobs associated with user *steve* does not exceed *2*.
- Total active jobs associated with group *staff* does not exceed *5*.
- Total nodes dedicated to class *batch* does not exceed *64*.
- Total active jobs associated with QoS *hiprio* does not exceed *3*.

While the preceding example is a bit complicated for most sites, similar combinations may be required to enforce policies found on many systems.

Idle Job Limits

Idle (or queued) job limits control which jobs are eligible for scheduling. To be eligible for scheduling, a job must meet the following conditions:

- Be idle as far as the resource manager is concerned (no holds).
- Have all job prerequisites satisfied (no outstanding job or data dependencies).
- Meet all idle job throttling policies.

If a job fails to meet any of these conditions, it will not be considered for scheduling and will not accrue service based job prioritization. (See service component and JOBPRIOACCRUALPOLICY.) The primary purpose of idle job limits is to ensure fairness among competing users by preventing queue stuffing and other similar abuses. Queue stuffing occurs when a single entity submits large numbers of jobs, perhaps thousands, all at once so they begin accruing queue time based priority and remain first to run despite subsequent submissions by other users.

Idle limits are specified in a manner almost identical to active job limits with the insertion of the capital letter *I* into the middle of the limit name. Below are examples of the **MAXIJOB** and **MAXINODE** limits, which are idle limit equivalents to the **MAXJOB** on page 324 and **MAXNODE** on page 324 limits:

MAXIJOB	
Units	# of jobs
Description	Limits the number of idle (eligible) jobs a credential may have at any given time.
Example	`USERCFG[DEFAULT]  MAXIJOB=8` `GROUPCFG[staff]  MAXIJOB=2,4`

MAXINODE	
Units	# of nodes

MAXINODE	
Description	Limits the total number of compute nodes that can be requested by jobs in the eligible/idle queue at any time. Once the limit is exceeded, the remaining jobs will be placed in the blocked queue. The number of nodes is determined by *<tasks>* / *<maximumProcsOnOneNode>* or, if using **JOBNODEMATCHPOLICY** on page 856 *EXACTNODE*, by the number of nodes requested.
Example	`USERCFG[DEFAULT]  MAXINODE=2`

Idle limits can constrain the total number of jobs considered to be eligible on a per credential basis. Further, like active job limits, idle job limits can also constrain eligible jobs based on aggregate requested resources. This could, for example, allow a site to indicate that for a given user, only jobs requesting up to a total of 64 processors, or 3200 processor-seconds would be considered at any given time. Which jobs to select is accomplished by prioritizing all idle jobs and then adding jobs to the eligible list one at a time in priority order until jobs can no longer be added. This eligible job selection is done only once per scheduling iteration, so, consequently, idle job limits only support a single hard limit specification. Any specified soft limit is ignored.

All single dimensional job limit types supported as active job limits are also supported as idle job limits. In addition, Moab also supports MAXIJOB[USER] and MAXIPROC[USER] policies on a per class basis. (See Basic Fairness Policies.)

Example:

```
USERCFG[steve]    MAXIJOB=2
GROUPCFG[staff]   MAXIJOB=5
CLASSCFG[batch]   MAXIJOB[USER]=2 MAXIJOB[USER:john]=6
QOSCFG[hiprio]    MAXIJOB=3
```

Hard and Soft Limits

Hard and soft limit specification allows a site to balance both fairness and utilization on a given system. Typically, throttling limits are used to constrain the quantity of resources a given credential (such as user or group) is allowed to consume. These limits can be very effective in enforcing fair usage among a group of users. However, in a lightly loaded system, or one in which there are significant swings in usage from project to project, these limits can reduce system utilization by blocking jobs even when no competing jobs are queued.

Soft limits help address this problem by providing additional scheduling flexibility. They allow sites to specify two tiers of limits; the more constraining limits soft limits are in effect in heavily loaded situations and reflect tight fairness constraints. The more flexible hard limits specify how flexible the scheduler can be in selecting jobs when there are idle resources available after all jobs meeting the tighter soft limits have started. Soft and hard limits are specified in the format *[<SOFTLIMIT>,] <HARDLIMIT>*. For example, a given site may want to use the following configuration:

```
USERCFG[DEFAULT]  MAXJOB=2,8
```

*With this configuration, the scheduler would select all jobs that meet the per user **MAXJOB** limit of 2. It would then attempt to start and reserve resources for all of these selected jobs. If after doing so there still remain available resources, the scheduler would then select all jobs that meet the less constraining hard per user **MAXJOB** limit of 8 jobs. These jobs would then be scheduled and reserved as available resources allow.*

If no soft limit is specified or the soft limit is less constraining than the hard limit, the soft limit is set equal to the hard limit.

Example:

```
USERCFG[steve]     MAXJOB=2,4 MAXNODE=15,30
GROUPCFG[staff]    MAXJOB=2,5
CLASSCFG[DEFAULT] MAXNODE=16,32
CLASSCFG[batch]    MAXNODE=12,32
QOSCFG[hiprio]     MAXJOB=3,5 MAXNODE=32,64
```

Job preemption status can be adjusted based on whether the job violates a soft policy using the ENABLESPVIOLATIONPREEMPTION parameter.

Per-partition Limits

Per-partition scheduling can set limits and enforce credentials and polices on a per-partition basis.

To enable per-partition scheduling, add the following to `moab.cfg`:

```
PERPARTITIONSCHEDULING TRUE
JOBMIGRATEPOLICY JUSTINTIME
```

With per-partition scheduling, it is recommended that limits go on the specific partitions and not on the global level. If limits are specified on both levels, Moab will take the more constricting of the limits. Also, please note that a DEFAULT policy on the global partition is not overridden by any policy on a specific partition.

Per-partition Limits

You can configure per-job limits and credential usage limits on a per-partition basis in the `moab.cfg` file. Here is a sample configuration for partitions *g02* and *g03* in `moab.cfg`.

```
PARCFG[g02]   CONFIGFILE=/opt/moab/parg02.cfg
PARCFG[g03]   CONFIGFILE=/opt/moab/parg03.cfg
```

You can then add per-partition limits in each partition configuration file:

```
# /opt/moab/parg02.cfg
CLASSCFG[pbatch]   MAXJOB=5
```

```
# /opt/moab/parg03.cfg
CLASSCFG[pbatch]   MAXJOB=10
```

You can configure Moab so that jobs submitted to any partition besides *g02*and *g03* get the default limits in `moab.cfg`:

```
stl

CLASSCFG[pbatch]  MAXJOB=2
```

Supported Credentials and Limits

The user, group, account, QoS, and class credentials are supported in per-partition scheduling.

The following per-job limits are supported:

- MAX.NODE
- MAX.WCLIMIT
- MAX.PROC

The following credential usage limits are supported:

- MAXJOB
- MAXNODE
- MAXPROC
- MAXWC
- MAXSUBMITJOBS

Multi-dimensional limits are supported for the listed credentials and per-job limits. For example:

```
CLASSCFG[pbatch]   MAXJOB[user:frank]=10
```

Usage-based Limits

Resource usage limits constrain the amount of resources a given job may consume. These limits are generally proportional to the resources requested and may include walltime, any standard resource, or any specified generic resource. The parameter RESOURCELIMITPOLICY controls which resources are limited, what limit policy is enforced per resource, and what actions the scheduler should take in the event of a policy violation.

Configuring Actions

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

If walltime is the resource to be limited, be sure that the resource manager is configured to not interfere if a job surpasses its given walltime. For TORQUE, this is done by using $ignwalltime in the configuration on each MOM node.

Format

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

Resource	Description
CPUTIME	Maximum total job proc-seconds used by any single job (allows scheduler enforcement of cpulimit).
DISK	Local disk space (in MB) used by any single job task.
JOBMEM	Maximum real memory/RAM (in MB) used by any single job. JOBMEM will only work with the MAXMEM flag.
JOBPROC	Maximum processor load associated with any single job. You must set **MAXPROC** on page 325 to use **JOBPROC**.
MEM	Maximum real memory/RAM (in MB) used by any single job task.
MINJOBPROC	Minimum processor load associated with any single job (action taken if job is using 5% or less of potential CPU usage).
NETWORK	Maximum network load associated with any single job task.
PROC	Maximum processor load associated with any single job task.
SWAP	Maximum virtual memory/SWAP (in MB) used by any single job task.
WALLTIME	Requested job walltime.

Policy	Description
ALWAYS	take action whenever a violation is detected
EXTENDEDVIOLATION	take action only if a violation is detected and persists for greater than the specified time limit
BLOCKEDWORKLOADONLY	take action only if a violation is detected and the constrained resource is required by another job

Action	Description
CANCEL	terminate the job

Action	Description
CHECKPOINT	checkpoint and terminate job
MIGRATE	requeue the job and require a different set of hosts for execution
NOTIFY	notify admins and job owner regarding violation
REQUEUE	terminate and requeue the job
SUSPEND	suspend the job and leave it suspended for an amount of time defined by the MINADMINSTIME parameter

Example 6-1: 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 6-2: 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 6-3: Cancel jobs that use 5% or less of potential CPU usage for more than 5 minutes

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

Example 6-4: Migrating a job when it blocks other workload

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

Specifying Hard and Soft Policy Violations

Moab is able to perform different actions for both hard and soft policy violations. In most resource management systems, a mechanism does not exist to allow the user to specify both hard and soft limits. To address this, Moab provides the RESOURCELIMITMULTIPLIER parameter that allows per partition and per resource multiplier factors to be specified to generate the actual hard and soft limits to be used. If the factor is less than one, the soft limit will be lower than the specified value and a Moab action will be taken before the specified limit is reached. If the factor is greater than one, the hard limit will be set higher than the specified limit allowing a buffer space before the hard limit action is taken.

In the following example, job owners will be notified by email when their memory reaches 100% of the target, and the job will be canceled if it reaches 125% of the target. For wallclock usage, the job will be requeued when it reaches 90% of the specified limit if another job is waiting for its resources, and it will be checkpointed when it reaches the full limit.

```
RESOURCELIMITPOLICY       MEM:ALWAYS,ALWAYS:NOTIFY,CANCEL
RESOURCELIMITPOLICY       WALLTIME:BLOCKEDWORKLOADONLY,ALWAYS:REQUEUE,CHECKPOINT
RESOURCELIMITMULTIPLIER   MEM:1.25,WALLTIME:0.9
```

Constraining Walltime Usage

While Moab constrains walltime using the parameter RESOURCELIMITPOLICY like other resources, it also allows walltime exception policies which are not available with other resources. In particular, Moab allows jobs to exceed the requested wallclock limit by an amount specified on a global basis using the JOBMAXOVERRUN parameter or on a per credential basis using the **OVERRUN** attribute of the CLASSCFG parameter.

```
JOBMAXOVERRUN    00:10:00
CLASSCFG[debug]  overrun=00:00:30
```

Related topics

- RESOURCELIMITPOLICY parameter
- FSTREE parameter (set usage limits within share tree hierarchy)
- Credential Overview
- JOBMAXOVERRUN parameter
- WCVIOLATIONACTION parameter
- RESOURCELIMITMULTIPLIER parameter

6.3 Fairshare

Fairshare allows historical resource utilization information to be incorporated into job feasibility and priority decisions. This feature allows site administrators to set system utilization targets for users, groups, accounts, classes, and QoS levels. Administrators can also specify the time frame over which resource utilization is evaluated in determining whether the goal is being reached. Parameters allow sites to specify the utilization metric, how historical information is aggregated, and the effect of fairshare state on scheduling behavior. You can specify fairshare targets for any credentials (such as user, group, and class) that administrators want such information to affect.

- Fairshare Parameters
  - FSPOLICY - Specifying the Metric of Consumption
  - Specifying Fairshare Timeframe
  - Managing Fairshare Data
- Using Fairshare Information
  - Fairshare Targets
  - Fairshare Caps
  - Priority-Based Fairshare

    - Per-Credential Fairshare Weights
    - Extended Fairshare Examples
- Hierarchical Fairshare/Share Trees
    - Defining the Tree
    - Controlling Tree Evaluation
- Importing Fairshare Data

Fairshare Parameters

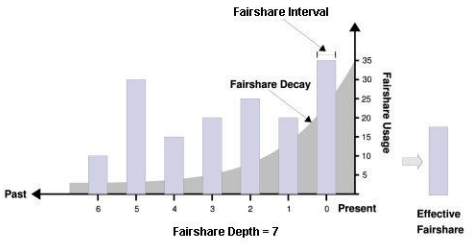
Fairshare is configured at two levels. First, at a system level, configuration is required to determine how fairshare usage information is to be collected and processed. Second, some configuration is required at the credential level to determine how this fairshare information affects particular jobs. The following are system level parameters:

Parameter	Description
FSINTERVAL	Duration of each fairshare window.
FSDEPTH	Number of fairshare windows factored into current fairshare utilization.
FSDECAY	Decay factor applied to weighting the contribution of each fairshare window.
FSPOLICY	Metric to use when tracking fairshare usage.

Credential level configuration consists of specifying fairshare utilization targets using the *CFG suite of parameters, including ACCOUNTCFG, CLASSCFG, GROUPCFG, QOSCFG, and USERCFG.

If global (multi-cluster) fairshare is used, Moab must be configured to synchronize this information with an identity manager.

Image 6-1: Effective fairshare over 7 days



FSPOLICY - Specifying the Metric of Consumption

As Moab runs, it records how available resources are used. Each iteration (RMPOLLINTERVAL seconds) it updates fairshare resource utilization statistics. Resource utilization is tracked in accordance with the FSPOLICY parameter allowing various aspects of resource consumption information to be measured. This

parameter allows selection of both the types of resources to be tracked as well as the method of tracking. It provides the option of tracking usage by dedicated or consumed resources, where dedicated usage tracks what the scheduler assigns to the job and consumed usage tracks what the job actually uses.

Metric	Description
DEDICATEDPES	Usage tracked by processor-equivalent seconds dedicated to each job. Useful in dedicated and shared nodes environments.
DEDICATEDPS	Usage tracked by processor seconds dedicated to each job. Useful in dedicated node environments.
UTILIZEDPS	Usage tracked by processor seconds used by each job. Useful in shared node/SMP environments.

Example 6-5:

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.

Specifying Fairshare Timeframe

When configuring fairshare, it is important to determine the proper timeframe that should be considered. Many sites choose to incorporate historical usage information from the last one to two weeks while others are only concerned about the events of the last few hours. The correct setting is very site dependent and usually incorporates both average job turnaround time and site mission policies.

With Moab's fairshare system, time is broken into a number of distinct fairshare windows. Sites configure the amount of time they want to consider by specifying two parameters, FSINTERVAL and FSDEPTH. The **FSINTERVAL** parameter specifies the duration of each window while the **FSDEPTH** parameter indicates the number of windows to consider. Thus, the total time evaluated by fairshare is simply **FSINTERVAL** * **FSDEPTH**.

Many sites want to limit the impact of fairshare data according to its age. The FSDECAY parameter allows this, causing the most recent fairshare data to contribute more to a credential's total fairshare

usage than older data. This parameter is specified as a standard decay factor, which is applied to the fairshare data. Generally, decay factors are specified as a value between 1 and 0 where a value of *1* (the default) indicates no decay should be specified. The smaller the number, the more rapid the decay using the calculation `WeightedValue = Value * <DECAY> ^ <N>` where *<N>* is the window number. The following table shows the impact of a number of commonly used decay factors on the percentage contribution of each fairshare window.

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

While selecting how the total fairshare time frame is broken up between the number and length of windows is a matter of preference, it is important to note that more windows will cause the decay factor to degrade the contribution of aged data more quickly.

Managing Fairshare Data

Using the selected fairshare usage metric, Moab continues to update the current fairshare window until it reaches a fairshare window boundary, at which point it rolls the fairshare window and begins updating the new window. The information for each window is stored in its own file located in the Moab statistics directory. Each file is named `FS.<EPOCHTIME>[.<PNAME>]` where *<EPOCHTIME>* is the time the new fairshare window became active (see sample data file) and *<PNAME>* is only used if per-partition share trees are configured. Each window contains utilization information for each entity as well as for total usage.

Historical fairshare data is recorded in the fairshare file using the metric specified by the FSPOLICY parameter. By default, this metric is processor-seconds.

Historical fairshare data can be directly analyzed and reported using the midag -f -v command.

When Moab needs to determine current fairshare usage for a particular credential, it calculates a decay-weighted average of the usage information for that credential using the most recent fairshare intervals where the number of windows evaluated is controlled by the FSDEPTH parameter. For example, assume the credential of interest is user *john* and the following parameters are set:

```
FSINTERVAL   12:00:00
 FSDEPTH      4
 FSDECAY      0.5
```

Further assume that the fairshare usage intervals have the following usage amounts:

Fairshare interval	Total user *john* usage	Total cluster usage
0	60	110
1	0	125
2	10	100
3	50	150

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

*Usage = (60 * 1 + .5^1 * 0 + .5^2 * 10 + .5^3 * 50) / (110 + .5^1*125 + .5^2*100 + .5^3*150)*

The current fairshare usage is relative to the actual resources delivered by the system over the timeframe evaluated, not the resources available or configured during that time.

Historical fairshare data is organized into a number of data files, each file containing the information for a length of time as specified by the FSINTERVAL parameter. Although **FSDEPTH**, **FSINTERVAL**, and **FSDECAY** can be freely and dynamically modified, such changes may result in unexpected fairshare status for a period of time as the fairshare data files with the old **FSINTERVAL** setting are rolled out.

Using Fairshare Information

Fairshare Targets

Once the global fairshare policies have been configured, the next step involves applying resulting fairshare usage information to affect scheduling behavior. As mentioned in the Fairshare Overview, by specifying fairshare targets, site administrators can configure how fairshare information impacts scheduling behavior. The targets can be applied to user, group, account, QoS, or class credentials using the **FSTARGET** attribute of *CFG credential parameters. These targets allow fairshare information to affect job priority and each target can be independently selected to be one of the types documented in the following table:

Target type - Ceiling	
Target modifier	-
Job impact	Priority
Format	Percentage Usage

Target type - Ceiling	
Description	Adjusts job priority down when usage exceeds target. See **How violated ceilings and floors affect fairshare-based priority** on page 291 for more information on how ceilings affect job priority.

Target type - Floor	
Target modifier	+
Job impact	Priority
Format	Percentage Usage
Description	Adj usts job priority up when usage falls below target. See **How violated ceilings and floors affect fairshare-based priority** on page 291 for more information on how floors affect job priority.

Target type - Target	
Target modifier	N/A
Job impact	Priority
Format	Percentage Usage
Description	Adjusts job priority when usage does not meet target.

Setting a fairshare target value of *0* indicates that there is no target and that the priority of jobs associated with that credential should not be affected by the credential's previous fairshare target. If you want a credential's cluster usage near 0%, set the target to a very small value, such as *0.001*.

Example

The following example increases the priority of jobs belonging to user *john* until he reaches 16.5% of total cluster usage. All other users have priority adjusted both up and down to bring them to their target usage of 10%:

```
FSPOLICY          DEDICATEDPS
 FSWEIGHT          1
 FSUSERWEIGHT      100
 USERCFG[john]     FSTARGET=16.5+
```

```
USERCFG[DEFAULT] FSTARGET=10
...
```

Fairshare Caps

Where fairshare targets affect a job's priority and position in the eligible queue, fairshare caps affect a job's eligibility. Caps can be applied to users, accounts, groups, classes, and QoSes using the **FSCAP** attribute of *CFG credential parameters and can be configured to modify scheduling behavior. Unlike fairshare targets, if a credential reaches its fairshare cap, its jobs can no longer run and are thus removed from the eligible queue and placed in the blocked queue. In this respect, fairshare targets behave like soft limits and fairshare caps behave like hard limits. Fairshare caps can be absolute or relative as described in the following table. If no modifier is specified, the cap is interpreted as relative.

Absolute Cap	
Cap Modifier:	^
Job Impact:	Feasibility
Format:	Absolute Usage
Description:	Constrains job eligibility as an absolute quantity measured according to the scheduler charge metric as defined by the FSPOLICY parameter

Relative Cap	
Cap Modifier:	%
Job Impact:	Feasibility
Format:	Percentage Usage
Description:	Constrains job eligibility as a percentage of total delivered cycles measured according to the scheduler charge metric as defined by the FSPOLICY parameter.

Example

The following example constrains the *marketing* account to use no more than *16,500* processor seconds during any given floating one week window. At the same time, all other accounts are constrained to use no more than *10%* of the total delivered processor seconds during any given one week window.

```
FSPOLICY           DEDICATEDPS
FSINTERVAL         12:00:00
FSDEPTH            14
```

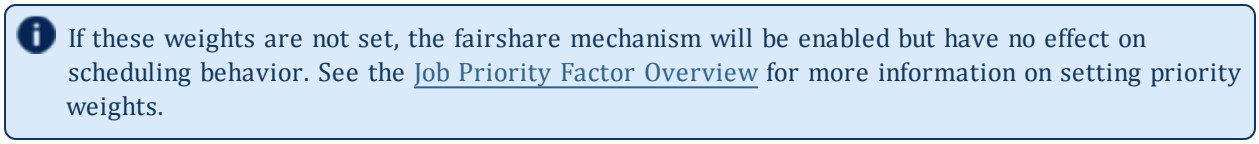
```
ACCOUNTCFG[marketing] FSCAP=16500^
ACCOUNTCFG[DEFAULT]   FSCAP=10
...
```

Priority-Based Fairshare

The most commonly used type of fairshare is priority based fairshare. In this mode, fairshare information does not affect whether a job can run, but rather only the job's priority relative to other jobs. In most cases, this is the desired behavior. Using the standard fairshare target, the priority of jobs of a particular user who has used too many resources over the specified fairshare window is lowered. Also, the standard fairshare target increases the priority of jobs that have not received enough resources.

While the standard fairshare target is the most commonly used, Moab can also specify fairshare ceilings and floors. These targets are like the default target; however, ceilings only adjust priority down when usage is too high and floors only adjust priority up when usage is too low.

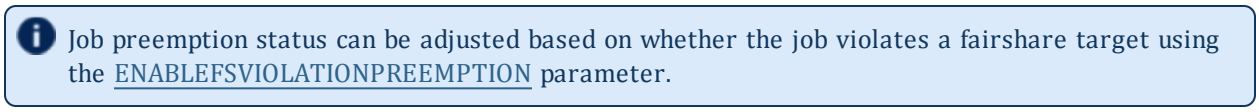
Since fairshare usage information must be integrated with Moab's overall priority mechanism, it is critical that the corresponding fairshare priority weights be set. Specifically, the FSWEIGHT component weight parameter and the target type subcomponent weight (such as FSACCOUNTWEIGHT, FSCLASSWEIGHT, FSGROUPWEIGHT, FSQOSWEIGHT, and FSUSERWEIGHT) is specified.



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

Example

```
 # set relative component weighting
FSWEIGHT      1
 FSUSERWEIGHT  10
 FSGROUPWEIGHT 50

 FSINTERVAL  12:00:00
 FSDEPTH     4
 FSDECAY     0.5
 FSPOLICY    DEDICATEDPS
 # all users should have a FS target of 10%
 USERCFG[DEFAULT] FSTARGET=10.0
 # user john gets extra cycles
 USERCFG[john]    FSTARGET=20.0
 # reduce staff priority if group usage exceed 15%
 GROUPCFG[staff]  FSTARGET=15.0-
 # give group orion additional priority if usage drops below 25.7%
 GROUPCFG[orion]  FSTARGET=25.7+
```



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

Credential-Specific Fairshare Weights

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

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

If specified, a per-credential fairshare weight is added to the global component fairshare weight.

The **FSWEIGHT** attribute is only enabled for ACCOUNT, GROUP, and QOS credentials.

Extended Fairshare Examples

Example 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.

```
#global general usage limits - negative priority jobs are considered in scheduling
ENABLENEGJOBPRIORITY   TRUE
# site policy - no job can last longer than 8 hours
USERCFG[DEFAULT] MAX.WCLIMIT=8:00:00
# Note:  default user FS target only specified to apply default user-to-user balance
USERCFG[DEFAULT] FSTARGET=1
# high-level fairshare config
FSPOLICY       DEDICATEDPS
FSINTERVAL     12:00:00
FSDEPTH        32 #recycle FS every 16 days
FSDECAY        0.8 #favor more recent usage info
# QoS config
QOSCFG[inst]    FSTARGET=25
QOSCFG[supp]    FSTARGET=5
QOSCFG[premium] FSTARGET=70
# account config (QoS access and fstargets)
# Note:  user-to-account mapping handled via allocation manager
# Note:  FS targets are percentage of total cluster, not percentage of QOS
ACCOUNTCFG[cs]   QLIST=inst    FSTARGET=10
ACCOUNTCFG[math] QLIST=inst    FSTARGET=15

ACCOUNTCFG[phys] QLIST=supp    FSTARGET=5
ACCOUNTCFG[eng]  QLIST=premium FSTARGET=70
# handle per-user priority exceptions
USERCFG[tom]  PRIORITY=100
USERCFG[john] PRIORITY=35
# define overall job priority
USERWEIGHT       10  # user exceptions
# relative FS weights (Note: QOS overrides ACCOUNT which overrides USER)
FSUSERWEIGHT     1
FSACCOUNTWEIGHT  10
FSQOSWEIGHT      100
# apply XFactor to balance cycle delivery by job size fairly
# Note:  queuetime factor also on by default (use QUEUETIMEWEIGHT to adjust)
XFACTORWEIGHT    100
# enable preemption
PREEMPTPOLICY  REQUEUE
```

```
# temporarily allow phys to preempt math
ACCOUNTCFG[phys] JOBFLAGS=PREEMPTOR PRIORITY=1000
ACCOUNTCFG[math] JOBFLAGS=PREEMPTEE
```

Hierarchical Fairshare/Share Trees

Moab supports arbitrary depth hierarchical fairshare based on a share tree. In this model, users, groups, classes, and accounts can be arbitrarily organized and their usage tracked and limited. Moab extends common share tree concepts to allow mixing of credential types, enforcement of ceiling and floor style usage targets, and mixing of hierarchical fairshare state with other priority components.

Defining the Tree

The FSTREE parameter can be used to define and configure the share tree used in fairshare configuration. This parameter supports the following attributes:

SHARES	
Format:	<COUNT>[@<PARTITION>][,<COUNT>[@<PARTITION>]]... where <COUNT> is a double and <PARTITION> is a specified partition name.
Description:	Specifies the node target usage or share.
Example:	`FSTREE[Eng]  SHARES=1500.5` `FSTREE[Sales] SHARES=2800`

MEMBERLIST	
Format:	Comma delimited list of child nodes of the format `[<OBJECT_TYPE>]:<OBJECT_ID>` where object types are only specified for *leaf nodes* associated with **user**, **group**, **class**, **qos**, or **acct** credentials.
Description:	Specifies the tree objects associated with this node.

Example:	`FSTREE[root]  SHARES=100  MEMBERLIST=Eng,Sales`
	`FSTREE[Eng]  SHARES=1500.5  MEMBERLIST=user:john,user:steve,user:bob`
	`FSTREE[Sales]  SHARES=2800  MEMBERLIST=Sales1,Sales2,Sales3`
	`FSTREE[Sales1] SHARES=30  MEMBERLIST=user:kellyp,user:sam`
	`FSTREE[Sales2] SHARES=10  MEMBERLIST=user:ux43,user:ux44,user:ux45`
	`FSTREE[Sales3] SHARES=60  MEMBERLIST=user:robert,user:tjackson`

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

Controlling Tree Evaluation

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

Policy	Description
FSTREETIERMULTIPLIER	Decreases the value of sub-level usage discrepancies. It can be a positive or negative value. When positive, the parent's usage in the tree takes precedence; when negative, the child's usage takes precedence. The usage amount is not changed, only the coefficient used when calculating the value of fstree usage in priority. When using this parameter, it is recommended that you research how it changes the values in `mdiag -p` to determine the appropriate use.
FSTREECAP	Caps lower level usage factors to prevent them from exceeding upper tier discrepancies.

Using FS Floors and Ceilings with Hierarchical Fairshare

All standard fairshare facilities including target floors, target ceilings, and target caps are supported when using hierarchical fairshare.

Multi-Partition Fairshare

Moab supports independent, per-partition hierarchical fairshare targets allowing each partition to possess independent prioritization and usage constraint settings. This is accomplished by setting the **PERPARTITIONSCHEDULING** attribute of the **FSTREE** parameter to *TRUE* in `moab.cfg` and setting `partition="name"` in your `<fstree>` leaf.

```
FSTREE[tree]
  <fstree>
    <tnode partition="slave1" name="root" type="acct" share="100" limits="MAXJOB=6">
      <tnode name="accta" type="acct" share="50" limits="MAXSUBMITJOBS=2 MAXJOB=1">
        <tnode name="fred" type="user" share="1" limits="MAXWC=1:00:00">
        </tnode>
      </tnode>
      <tnode name="acctb" type="acct" share="50" limits="MAXSUBMITJOBS=4 MAXJOB=3">
        <tnode name="george" type="user" share="1" >
        </tnode>
      </tnode>
    </tnode>
    <tnode partition="slave2" name="root" type="acct" share="100"
limits="MAXSUBMITJOBS=6 MAXJOB=5">
      <tnode name="accta" type="acct" share="50">
        <tnode name="paul" type="user" share="1">
        </tnode>
      </tnode>
      <tnode name="acctb" type="acct" share="50">
        <tnode name="ringo" type="user" share="1">
        </tnode>
      </tnode>
    </tnode>
  </fstree>
```

If no partition is specified for a given share value, then this value is assigned to the global partition. If a partition exists for which there are no explicitly specified shares for any node, this partition will use the share distribution assigned to the global partition.

Dynamically Importing Share Tree Data

Share trees can be centrally defined within a database, flat file, information service, or other system and this information can be dynamically imported and used within Moab by setting the **FSTREE** parameter within the **Identity Managers** on page 695. This interface can be used to load current information at startup and periodically synchronize this information with the master source.

To create a fairshare tree in a separate XML file and import it into Moab

1. Create a file to store your fairshare tree specification. Give it a descriptive name and store it in your Moab home directory (`$MOABHOMEDIR` or `$MOABHOMEDIR/etc`). In this example, the file is called `fstree.dat`.
2. In the first line of `fstree.dat`, set **FSTREE**`[myTree]` to indicate that this is a fairshare file.
3. Build a tree in XML to match your needs. For example:

```
FSTREE[myTree]
<fstree>
<tnode name="root" share="100">
<tnode name="john" type="user" share="50" limits="MAXJOB=8 MAXPROC=24
MAXWC=01:00:00"></tnode>
<tnode name="jane" type="user" share="50" limits="MAXJOB=5"></tnode>
</tnode>
</fstree>
```

This configuration creates a fairshare tree in which users share a value of 100. Users `john` and `jane` share the value equally, because each has been given 50.

Because 100 is an arbitrary number, users `john` and `jane` could be assigned `10000` and `10000` respectively and still have a 50% share under the parent leaf. To keep the example simple, however, it is recommended that you use 100 as your arbirary share value and distribute the share as percentages. In this case, `john` and `jane` each have 50%.

If the users' numbers do not add up to at least the fairshare value of 100, the remaining value is shared among all users under the tree. For instance, if the tree had a value of 100, user `john` had a value of 50, and user `jane` had a value of 25, then 25% of the fairshare tree value would belong to all other users associated with the tree. By default, tree leaves do not limit who can run under them.

Each value specified in the `tnode` elements must be contained in quotation marks.

4. Optional: Share trees defined within a flat file can be cumbersome; consider running tidy for xml to improve readability. Sample usage:

```
> tidy -i -xml goldy.cfg <filename> <output file>

# Sample output

FSTREE[myTree]
<fstree>
  <tnode name="root" share="100">
    <tnode name="john"   type="user" share="50" limits="MAXJOB=8
    MAXPROC=24 MAXWC=01:00:00">
    </tnode>
    <tnode name="jane" type="user" share="50" limits="MAXJOB=5">
    </tnode>
  </tnode>
</fstree>
```

5. Link the new file to Moab using the IDCFG parameter in your Moab configuration file.

```
IDCFG[myTree] server="FILE:///$MOABH OMEDIR/etc/fstree.dat" REFRESHPERIOD=INFINITY
```

*Moab imports the `myTree` fairshare tree from the `fstree.dat` file. Setting **REFRESHPERIOD** to INFINITY causes Moab to read the file each time it starts or restarts, but other settings (hour, day, month) cause Moab to read the file more often (See Refreshing Identity Manager Data for more information).*

6. To view your fairshare tree configuration, run mdiag -f. If it is configured correctly, the tree information will appear beneath all the information about your fairshare settings configured in `moab.cfg`.

```
> mdiag -f
Share Tree Overview for partition 'ALL'
Name          Usage    Target                (FSFACTOR)
----          -----    ------               ------------
root        100.00    100.00  of   100.00 (node: 1171.81) (0.00)
- john       16.44     50.00  of   100.00 (user: 192.65) (302.04) MAXJOB=8
MAXPROC=24 MAXWC=3600
- jane       83.56     50.00  of   100.00 (user: 979.16) (-302.04)  MAXJOB=5
```

The settings you configured in fstree.dat appear in the output. The tree of 100 is shared equally between users `john` and `jane`.

Specifying Share Tree Based Limits

Limits can be specified on internal nodes of the share tree using standard credential limit semantics. The following credential usage limits are valid:

- MAXIJOB (Maximum number of idle jobs allowed for the credential)
- **MAXJOB** on page 324
- **MAXMEM** on page 324
- **MAXNODE** on page 324
- **MAXPROC** on page 325
- **MAXSUBMITJOBS** on page 326
- **MAXWC** on page 326

Example 6-6: FSTREE limits example

```
FSTREE[myTree]
<fstree>
  <tnode name="root" share="100">
   <tnode name="john"   type="user" share="50" limits="MAXJOB=8
    MAXPROC=24 MAXWC=01:00:00">
    </tnode>
    <tnode name="jane" type="user" share="50" limits="MAXJOB=5">
    </tnode>
  </tnode>
</fstree>
```

Other Uses of Share Trees

If a share tree is defined, it can be used for purposes beyond fairshare, including organizing general usage and performance statistics for reporting purposes (see showstats -T), enforcement of tree node based usage limits, and specification of resource access policies.

Related topics

- mdiag -f command (provides diagnosis and monitoring of the fairshare facility)
- FSENABLECAPPRIORITY parameter
- ENABLEFSPREEMPTION parameter
- FSTARGETISABSOLUTE parameter

6.3.1 Sample FairShare Data File

FS.<EPOCHTIME>

```
# FS Data File (Duration:  43200 seconds)  Starting: Sat Jul  8 06:00:20
user               jvella   134087.910
user             reynolds    98283.840
user               gastor    18751.770
user               uannan   145551.260
user              mwillis   149279.140
...
group              DEFAULT   411628.980
group              RedRock  3121560.280
group               Summit   500327.640
group               Arches  3047918.940
acct        Administration   653559.290
acct           Engineering  4746858.620
acct                Shared    75033.020
acct              Research  1605984.910
qos               Deadline  2727971.100
qos           HighPriority  4278431.720
qos               STANDARD    75033.020
class                batch  7081435.840
sched             iCluster  7081435.840
```

The total usage consumed in this time interval is 7081435.840 processor-seconds. Since every job in this example scenario had a user, group, account, and QoS assigned to it, the sum of the usage of all members of each category should equal the total usage value: USERA + USERB + USERC + USERD = GROUPA + GROUPB = ACCTA + ACCTB + ACCTC = QOS0 + QOS1 + QOS2 = SCHED.

6.4 Charging and Allocation Management

- Charging and Allocation Management Overview
  - Configuring the Allocation Manager Interface
  - Allocation Management Policies
  - Charge Metrics
- Allocation Manager Types
  - MAM (Moab Accounting Manager)
  - Native Allocation Manager

Charging and Allocation Management Overview

Either Moab HPC Suite 7.0 - Enterprise Edition or Moab Cloud Suite 7.0 are required for support of charging and allocation management capabilities.

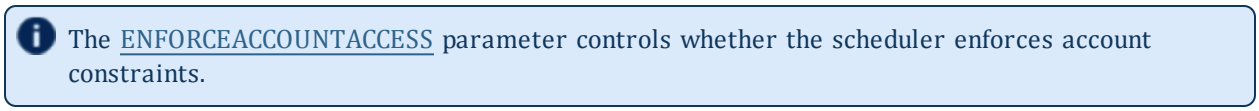
An allocation manager is a software system that manages resource allocations. A resource allocation grants a job a right to use a particular amount of resources. While full details of each allocation manager may be found within its respective documentation, the following brief review highlights a few 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. These allocations can be used for particular clusters or globally. They can also have time frames associated with them to establish an allocation cycle. All transaction information is typically stored in a database or directory server allowing extensive statistical and allocation tracking.

When using an allocation 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.

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.



Supported allocation managers include MAM (Moab Accounting Manager) and Native.

Moab Accounting Manager is a commercial charge-back accounting system that can be used in cloud or HPC environments. It is based on the Gold Allocation Manager.

The MAM allocation manager type (AMCFG[*mam*] TYPE=*MAM*) uses the direct SSS wire protocol originally used by Gold.

The Native allocation manager type (AMCFG[*mam*] TYPE=*Native*) invokes scripts to create a customization layer between Moab Workload Manager and Moab Accounting Manager. The interface is used for cloud contexts and can be used in some hpc contexts where greater accounting customization is required. The native interface can also be customized to interact with third-party allocation manager system. Moab makes calls to the scripts that handle the interaction with the external system.

The MAM interface is faster while the native interface allows a higher level of customization.

Configuring the Allocation Manager Interface

Configure Moab to use the Moab Accounting Manager by running `./configure` with the applicable options when installing Moab:

- `--with-am[=TYPE]` - enables accounting management with the specified accounting manager type (mam or native) [mam]
- `--with-am-dir=DIR` - uses the specified prefix directory for the accounting manager if installed in a non-default location
- `--with-cloud[=EDITION]` - specifies cloud edition [xcat]

The `--with-am` option specifies the accounting manager type that you want to use as either mam, which is the default, or native. Specifying this option will add the necessary entries into the `moab.cfg` file and cause the install process to copy configuration files, scripts, and libraries into place.

Use `--with-am-dir` to specify the prefix directory for Moab Accounting Manager if the native type is being used and it has been installed in a non-default location.

The `--with-cloud` option specifies that you are installing Moab in a cloud context (HPC is the default context) and makes some adjustments to the configuration files and interface scripts necessary for contextually appropriate charging behaviors. This option includes automatically setting the `--with-am=native` option, since the mam accounting manager type is not supported in the cloud context. If you are specifying a cloud context but do not wish to use the accounting manager, use the `--without-am` configure option.

The following is an example of configuring Moab charging for HPC:

```
./configure --with-am
```

The following is an example of configuring Moab charging for cloud:

```
./configure --with-cloud
```

If using a native allocation manager type, it will also be necessary to run `make perldeps` as root when installing so that the prerequisite bundled Perl modules are installed.

```
make perldeps
```

If you want to configure one of the other types of allocation manager, follow the instructions in the appropriate section.

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:

CHARGEPOLICY on page 356
CREATEFAILUREACTION on page 357
CREATEURL on page 357
DELETEURL on page 358
ENDURL on page 358
FALLBACKACCOUNT on page 358
FALLBACKQOS on page 359
FLAGS on page 359
FLUSHINTERVAL on page 360
NODECHARGEPOLICY on page 360
PAUSEURL on page 361
QUOTEURL on page 361
SERVER on page 361
SOCKETPROTOCOL on page 362
STARTFAILUREACTION on page 362
STARTURL on page 362
TIMEOUT on page 363
UPDATEURL on page 363
VALIDATEJOBSUBMISSION on page 363
WIREPROTOCOL on page 364

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. When you use the Native accounting manager interface, Moab ignores the configured **CHARGEPOLICY** and instead uses **DEBITALLWC** to calculate charges. The DEBITSUCCESSFUL* policies require TORQUE to work. Additionally, the job scripts must return a negative number as the exit code on failure in order to be ignored for charging.

CHARGEPOLICY

Example

```
AMCFG[mam] 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 the allocation manager to calculate charges.

CREATEFAILUREACTION

Format	<AMFailureAction>[,<FundsFailureAction>] where the action is one of CANCEL, DEFER, HOLD, or IGNORE
Default	**IGNORE,IGNORE**
Description	Before creating a job that should be tracked or charged within the accounting manager, Moab contacts the accounting manager for authorization. If the job creation is rejected due to lack of funds, Moab applies the FundsFailureAction to the job. For any other rejection reason including a connection problem, Moab applies the AMFailureAction to the job. If you do not specify a FundsFailureAction, Moab will apply the AMFailureAction for an insufficient funds failure. If the action is set to CANCEL, Moab cancels the job; DEFER, defers the job; HOLD, puts the job on hold; and IGNORE, ignores the failure and continues to start the job. This parameter applies to both the mam and native accounting manager types.
Example	`AMCFG[mam] CREATEFAILUREACTION=HOLD` *A job will be placed on hold when submitted if there are insufficient funds for it to start.*

CREATEURL

Format	exec://<fullPathToCreateScript>
Default	---
Description	If you use the native accounting manager interface, Moab runs this script to create a chargeable job or reservation in order to determine whether it should be created. For jobs, the CREATEFAILUREACTION parameter specifies the action that should be taken if the authorization fails (such as for insufficient funds).

CREATEURL

Example	`AMCFG[mam] CREATEURL=exec://$TOOLSDIR/mam/usage.create.mam.pl` *Moab calls the `usage.create.mam.pl` script for authorization before starting a job or reservation.*

DELETEURL

Format	exec://<fullPathToDeleteScript>
Default	---
Description	If you use the native accounting manager interface, Moab runs this script as needed to clean up after an interrupted job or reservation life-cycle. The default behavior is to remove outstanding liens.
Example	`AMCFG[mam] DELETEURL=exec://$TOOLSDIR/mam/usage.delete.mam.pl` *Moab calls the `usage.delete.mam.pl` script to clean up after an interrupted job or reservation.*

ENDURL

Format	exec://<fullPathToEndScript>
Default	---
Description	If you use the native accounting manager interface, Moab runs this script after the end of a chargeable job or reservation in order to make a final charge or update the accounting record. The default behavior is to make a prorated charge for the job or reservation.
Example	`AMCFG[mam] ENDURL=exec://$TOOLSDIR/mam/usage.end.mam.pl` *Calls the `usage.end.mam.pl` script to make the final charge for a job or reservation.*

FALLBACKACCOUNT

Format	STRING

FALLBACKACCOUNT	
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	`AMCFG[mam] FALLBACKACCOUNT=freecycle` *Moab assigns the account freecycle to jobs that do not have adequate allocations in their primary account.*

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	`AMCFG[mam] FALLBACKQOS=freecycle` *Moab assigns the QoS freecycle to jobs that do not have adequate allocations in their primary QoS.*

FLAGS	
Format	<STRING>
Default	---
Description	AMCFG flags are used to enable special services.

FLAGS

Example

```
AMCFG[mam] FLAGS=LOCALCOST
```

Moab calculates the charge for the job locally and sends that as a charge to the allocation manager, which then charges that amount for the job.

FLUSHINTERVAL

Format	HOUR, DAY, WEEK, MONTH (or NONE)
Default	NONE
Description	Indicates the amount of time between allocation manager debits for long running reservation and job based charges. Only accepts intervals such as HOUR, DAY, WEEK, MONTH, or NONE.

Example

```
AMCFG[mam] FLUSHINTERVAL=DAY
```

Moab updates its charges every 24 hours for long running jobs and reservations.

NODECHARGEPOLICY

Format	One of **AVG**, **MAX**, or **MIN**
Default	**MIN**
Description	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.) This feature can only be used in conjunction with the AMCFG[] LOCALCOST flag which limits its use to cases where Moab calculates the full charge to be used by Moab Accounting Manager.

Example

```
NODECFG[node01]  CHARGERATE=1.5
NODECFG[node02]  CHARGERATE=1.75
AMCFG[mam] NODECHARGEPOLICY=MAX
```

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

PAUSEURL	
Format	EXEC://<fullPathToPauseScript>
Default	---
Description	If you use the native accounting manager interface, Moab runs this script after preempting a job that might be resumed later. The default behavior is to make an incremental charge but not create a fresh lien.
Example	`AMCFG[mam] PAUSEURL=exec://$TOOLSDIR/mam/usage.pause.mam.pl` *Moab calls the `usage.pause.mam.pl` script after pausing a job.*

QUOTEURL	
Format	exec://<fullPathToQuoteScript>
Default	---
Description	If you use the native accounting manager interface, Moab runs the specified script to determine the amount the accounting manager will charge for a job or reservation.
Example	`AMCFG[mam] QUOTEURL=exec://$TOOLSDIR/mam/usage.quote.mam.pl` *Moab calls the `usage.quote.mam.pl` script when it needs to determine the cost for a job or reservation.*

SERVER	
Format	URL
Default	N/A
Description	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.
Example	`AMCFG[mam] SERVER=mam://tiny.supercluster.org:4368`

SOCKETPROTOCOL	
Format:	One of **SUTCP**, **SSS-HALF**, **HTTP**, or **SSS-CHALLENGE**
Default:	**SSS-HALF**
Description:	Specifies the socket protocol to be used for scheduler-allocation manager communication.
Example:	`AMCFG[mam] SOCKETPROTOCOL=SSS-CHALLENGE`

STARTFAILUREACTION	
Format:	<AMFailureAction>[,<FundsFailureAction>] where the action is one of **CANCEL**, **DEFER**, **HOLD**, **IGNORE**, or **RETRY**
Default:	**IGNORE,IGNORE**
Description:	Moab applies *<FundsFailureAction>* to any job that should be tracked or charged within the account manager if it is rejected due to insufficient funds for the given user and account. Moab applies *<AMFailureAction>* to a job if the account manager rejects it for any other reason. If you do not specify a *<FundsFailureAction>*, Moab will apply the *<AMFailureAction>* for an insufficient funds failure. If the action is set to **CANCEL**, Moab cancels the job; **DEFER**, defers the job; **HOLD**, puts the job on hold; **IGNORE**, ignores the failure and continues to start the job; and **RETRY**, does not start the job on this attempt but attempts to start the job at the next opportunity. STARTFAILUREACTION applies to both the mam and native accounting manager type.
Example:	`AMCFG[mam] STARTFAILUREACTION=HOLD` *A job will be placed on hold if there are insufficient funds when it is time for it to start.*

STARTURL	
Format:	exec://<fullPathToStartScript>
Default:	---
Description:	If you use the native accounting manager interface, Moab runs this script on a chargeable job or reservation to determine whether it should start. For jobs, the STARTFAILUREACTION attribute specifies the action that Moab should take if the authorization fails (such as for insufficient funds).

STARTURL	
Example:	`AMCFG[mam] STARTURL=exec://$TOOLSDIR/mam/usage.start.mam.pl` *Moab calls the `usage.start.mam.pl` script for authorization before starting a job or reservation.*

TIMEOUT	
Format:	[[[DD:]HH:]MM:]SS
Default:	15
Description:	Specifies the maximum delay allowed for scheduler-allocation manager communications.
Example:	`AMCFG[mam] TIMEOUT=30`

UPDATEURL	
Format	exec://<fullPathToUpdateScript>
Description	If you use the native accounting manager interface and have **FLUSHINTERVAL** set, Moab runs this script every flush interval for each chargeable job or reservation to determine if it should continue. The default behavior is to charge for the previous charge interval and create a lien for the next.
Example	`AMCFG[mam] UPDATEURL=exec://$TOOLSDIR/mam/usage.update.mam.pl` *Moab calls the `usage.update.mam.pl` script for authorization to continue a job or reservation.*

VALIDATEJOBSUBMISSION	
Format:	<BOOLEAN>
Default:	*FALSE*

VALIDATEJOBSUBMISSION	
Description:	If set to *TRUE*, when a new job is submitted, Moab will execute the **CREATEURL** script (for **TYPE**=*Native*) or seek a job quote from Moab Accounting Manager (**TYPE**=*MAM*) before allowing the job to be submitted. Otherwise, the fund validation step is just utilized by reservations and fallback account checks. If the call fails (for example, if the user's account does not have sufficient funds or specifies an invalid account.), Moab applies the **CREATEFAILUREACTION**.
Example:	`AMCFG[mam] VALIDATEJOBSUBMISSION=TRUE` *Moab calls the `usage.update.mam.pl` script for authorization to continue a job or reservation.*

WIREPROTOCOL	
Format:	One of **AVP**, **HTML**, **SSS2**, or **XML**
Default:	**XML**
Description:	Specifies the wire protocol to be used for scheduler-allocation manager communication.
Example:	`AMCFG[mam] WIREPROTOCOL=SSS2`

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.

In the case of the Moab Accounting Manager, 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. 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:mam] KEY=secret_key AUTHTYPE=HMAC64
```

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.

Flag Name	Description
ACCOUNTFAILASFUNDS	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.
LOCALCOST	Moab calculates the charge for the job locally and sends that as a charge to the allocation manager, which then charges the amount for the job.
STRICTQUOTE	Sends an estimated process count to the allocation manager when an initial quote is requested for a newly-submitted job.

Allocation Management Policies

In most cases, the scheduler interfaces with a peer service. 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.

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.
- *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 using PE weighted wallclock time dedicated as the usage metric.
- *DEBITSUCCESSFULBLOCKED* - Charges only jobs that successfully complete based on processors dedicated and blocked according to node access policy or QoS node exclusivity.

When you use the Native accounting manager interface, Moab ignores the configured **CHARGEPOLICY** and instead uses **DEBITALLWC** to calculate charges.

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 Manager Types

Moab supports two allocation manager types: MAM (Moab Accounting Manager) and Native.

MAM (Moab Accounting Manager)

Moab Accounting Manager is an accounting management system that provides usage tracking, charge accounting, and allocation enforcement for resource or service usage in cloud and technical computing environments. It acts like a bank in which credits are deposited into accounts with constraints

designating which entities may access the account. As resources or services are utilized, accounts are charged and usage recorded. MAM supports familiar operations such as deposits, withdrawals, transfers, and refunds and provides balance and usage feedback to users, managers, and system administrators.

To configure Moab to use the MAM interface for allocation management, use the configure options as described in the Configuring the Allocation Manager Interface section.

Example 6-7:

```
./configure -with-am=mam ...
```

Consequently, `make install` will add the essential configuration and connection entries into the `moab.cfg` and `moab-private.cfg` files.

The following are typical entries in the Moab configuration files for using the MAM interface in the HPC context:

`moab.cfg`:

```
AMCFG[mam] TYPE=MAM HOST=localhost PORT=7112
AMCFG[mam] STARTFAILUREACTION=HOLD CHARGEPOLICY=DEBITALLWC
```

`moab-private.cfg`:

```
CLIENTCFG[AM:mam] KEY=secret_key AUTHTYPE=HMAC64
```

Synchronize the secret key with Moab Accounting Manager by copying the value of the `token.value` parameter from the `MAM_PREFIX/etc/site.conf` file which is randomly generated during the Moab Accounting Manager install process.

Moab Accounting Manager should be installed, started, and initialized. See the Getting Started chapter of the Moab Accounting Manager Administrator Guide for examples of how to initialize MAM for your initial mode of operation.

Native

When utilizing the Native allocation manager interface, Moab calls scripts to perform its accounting and allocation functions instead of communicating directly with the allocation manager, as is the case with MAM. The default scripts are designed to interact with Moab Accounting Manager and are typically used in the cloud context. You can customize these scripts to the needs of your site, either to provide additional flexibility in accounting and allocation management with Moab Accounting Manager or to interact with a third-party accounting or allocation management system.

Moab will invoke the Native Allocation Manager Interface (NAMI) scripts (see the **AMCFG** parameter documentation for **QUOTEURL**, **CREATEURL**, **STARTURL**, **UPDATEURL**, **PAUSEURL**, **ENDURL**, **DELETEURL**) by passing the job or reservation information via XML to the standard input of the script. The script should return a return code (zero for success), data on standard out and messages on standard error. A failure in **CREATEURL**, **STARTURL**, or **UPDATEURL** should result in the application of the **CREATEFAILUREACTION**, **STARTFAILUREACTION** or **UPDATEFAILUREACTION**, respectively.

To configure Moab to use the Native interface for allocation management, use the configure options as described in the Configuring the Allocation Manager Interface section.

Example 6-8:

```
./configure --with-am=native ...
```

Consequently, `make install` will add the essential allocation manager entries into `moab.cfg` and install the accounting-related scripts (`$PREFIX/tools/mam/usage.*.mam.pl`) and configuration files (`$MOABHOMEDIR/etc/nami.cfg`) in the correct locations.

The following are typical entries in the Moab configuration files for using the Native interface for the HPC context:

`moab.cfg`:

```
AMCFG[mam] TYPE=NATIVE
AMCFG[mam] CREATEURL=exec://$TOOLSDIR/mam/usage.create.mam.pl
AMCFG[mam] STARTURL=exec://$TOOLSDIR/mam/usage.start.mam.pl
AMCFG[mam] UPDATEURL=exec://$TOOLSDIR/mam/usage.update.mam.pl
AMCFG[mam] PAUSEURL=exec://$TOOLSDIR/mam/usage.pause.mam.pl
AMCFG[mam] ENDURL=exec://$TOOLSDIR/mam/usage.end.mam.pl
AMCFG[mam] DELETEURL=exec://$TOOLSDIR/mam/usage.delete.mam.pl
AMCFG[mam] STARTFAILUREACTION=HOLD
```

`nami.cfg`:

```
Context                             hpc
LOG[config]                         path=/opt/mam/log/nami.log
LOG[config]                         loglevel=debug
LOG[config]                         includeGoldLog=false
LOG[config]                         permissions=666
LOG[config]                         maxSize=10000000
LOG[config]                         rolloverLimit=7)
```

To view your current URL and FAILUREACTION information, run mdiag -R -v (even more information may be available in `mdiag -R -v --xml`). The following shows sample output from running the mdiag command:

```
AM[mam]  Type: native  State: 'Active'
ValidateJobSubmission:     FALSE
FlushInterval:  1:00:00
ChargePolicy:   DEBITALLWC
Quote URL:
Create URL:     CREATEURL=exec://$TOOLSDIR/mam/usage.create.mam.pl
Start URL:      STARTURL=exec://$TOOLSDIR/mam/usage.start.mam.pl
Update URL:     UPDATEURL=exec://$TOOLSDIR/mam/usage.update.mam.pl
Pause URL:      PAUSEURL=exec://$TOOLSDIR/mam/usage.pause.mam.pl
End URL:        ENDURL=exec://$TOOLSDIR/mam/usage.end.mam.pl
Delete URL:     DELETEURL=exec://$TOOLSDIR/mam/bank.delete.mam.pl
```

Moab Accounting Manager should be installed, started, and initialized. The simplest procedure is to install it on the same server as Moab Workload Manager so that the Moab Accounting Manager can share libraries and configuration files with the Moab Workload Manager and Moab Accounting Manager scripts. See the Getting Started chapter of the Moab Accounting Manager User Guide for examples of how to initialize MAM for your initial mode of operation.

Related topics

- Per Class DISABLEAM attribute
- Charging for Reservations
- ENFORCEACCOUNTACCESS parameter

6.5 Charging a Workflow

The first thing you need to do is install Moab Accounting Manager. You can do this through the Adaptive Computing website. After Moab Accounting Manager is installed you can look at the `createExampleGoldState.pl` script to see examples of how to define users, projects, accounts, and charge rates in Moab Accounting Manager. You can also run the script to set up a few test users, accounts, and charge rates in Moab Accounting Manager. If ever you want to clear the sqlite database in Moab Accounting Manager, run `./resetdb` as root with the first parameter being the accounting admin user and it will restore the accounting state the same as a fresh install.

Add NAMI URLs to moab.cfg

The out-of-the-box-solution doesn't use the create or modify NAMI URL. Moab uses the reserve URL to reserve funds and charge for setup costs, the charge URL to reserve funds and charge for reoccurring costs, and the delete URL to unreserve the funds. *These three URLs must be defined in `moab.cfg` in order to charge a workflow.* You should set a Reserve Failure Action in `moab.cfg` (the bottom line in the example) in case the customer doesn't have enough funds to proceed. Therefore a sample `moab.cfg` might look like this in the NAMI section:

```
AMCFG[nami] TYPE=NATIVE
AMCFG[nami] StartURL=exec:///opt/moab/tools/mam/usage.start.mam.pl
AMCFG[nami] UpdateURL=exec:///opt/moab/tools/mam/usage.update.mam.pl
AMCFG[nami] PauseURL=exec:///opt/moab/tools/mam/usage.pause.mam.pl
AMCFG[nami] EndURL=exec:///opt/moab/tools/mam/usage.end.mam.pl
AMCFG[nami] DeleteURL=exec:///opt/moab/tools/mam/usage.delete.mam.pl
AMCFG[nami] FLUSHINTERVAL=hour
AMCFG[nami] StartFailureAction=HOLD
```

How Workflows Are Expressed in nami.cfg

The `nami.cfg` file was designed to be similar to the `moab.cfg`. Each line is parsed as either `Predicate[constant]` **Attribute**=*value* or *key* *value*. Each constant in the JobTemplate predicate identifies specific job templates in `moab.cfg`. In respect to the workflow, `nami.cfg` needs to know about:

- (required) What job template in the workflow is the infinite job that you will charge based on a recurring bill cycle. In order to specify this you must set the **Recurring** attribute to *True*. For example:

  ```
  JobTemplate[newvm]              Recurring=True
  ```

- (required) What job template starts off the workflow where you will reserve funds for the

eventual infinite job's first recurring bill as well as any setup costs that might occur. For example:

```
JobTemplate[newvm]              StartJob=newvmprovision
```

- (required) What job templates follow the infinite job. This is important to register these template names with NAMI in case of a job failure; otherwise, NAMI will simply ignore them.

```
JobTemplate[newvm]               TemplateDepend=newvmprovision
JobTemplate[newvm]               TemplateDepend=vmstoragehookup
```

- (optional) Moab Accounting Manager VBF charge rate associated with provision jobs. These charge rates will be fixed setup costs per provision job that will be summed up and reserved with the first recurring bill at the run time of the start job. They will be charged at run time of the infinite job to ensure that all provision jobs ran successfully. It is important to note that the run time of the infinite job is also when the bill cycle starts. The first recurring bill is not charged with the setup costs at the beginning of the first bill cycle but rather at the end of each bill cycle after the resources are used. The following is an example of configuring multiple set up costs:

```
JobTemplate[newvmprovision]         GoldChargeRateName=VmSetUpCost
                                       JobTemplate[vmstoragehookup]       GoldChargeRateName=VmS
```

- (optional) In case you want to toggle billing on and off for a workflow, you can use the **TrackUsage** attribute. The attribute only needs to be set on the recurring job for each workflow. Since having billing on is the default behavior, here is an example of turning billing off:

```
JobTemplate[newvm]                      TrackUsage=False
```

Workflow from moab.cfg to nami.cfg Example

moab.cfg

```
JOBCFG[newvmprovision]    GENERICSYSJOB=AType=exec,Action="$TOOLSDIR/newvmprovision.pl
$*.SM",EType=start,Timeout=2:00,Flags=objectxmlstdin
JOBCFG[newvmprovision]    INHERITRES=TRUE
JOBCFG[newvmprovision]    FLAGS=NORMSTART

JOBCFG[vmstoragehookup]
GENERICSYSJOB=AType=exec,Action="$TOOLSDIR/vmstoragehookup.pl $.SM
$.SMMAP",EType=start,Timeout=2:00,Flags=objectxmlstdin
JOBCFG[vmstoragehookup]   INHERITRES=TRUE
JOBCFG[vmstoragehookup]   FLAGS=NORMSTART
JOBCFG[vmstoragehookup]   TEMPLATEDEPEND=AFTEROK:newvmprovision

JOBCFG[newvm]             FLAGS=VMTRACKING    SELECT=true
JOBCFG[newvm]             TEMPLATEDEPEND=AFTEROK:vmstoragehookup
JOBCFG[newvm]             FLAGS=NORMSTART
JOBCFG[newvm]             DESTROYTEMPLATE=destroyvm
JOBCFG[newvm]             MIGRATETEMPLATE=migratevm
```

nami.cfg

```
JobTemplate[newvm]                Recurring=True
JobTemplate[newvm]                StartJob=newvmprovision
JobTemplate[newvm]                TrackUsage=True
JobTemplate[newvm]                TemplateDepend=newvmprovision
```

```
JobTemplate[newvm]                  TemplateDepend=vmstoragehookup
JobTemplate[newvmprovision]         GoldChargeRateName=VmSetUpCost
JobTemplate[vmstoragehookup]        GoldChargeRateName=VmStorageHookUpCost
```

Understanding UsageAttributes in nami.cfg

Recurring jobs are represented in Moab Accounting Manager as usage records. A usage record helps constrain what account to charge, how much to charge, as well as keep record of additional information about the job. The `nami.cfg` file defines what the usage record will like for the recurring job. Each constant in the UsageAttribute can be an account constraint, charge rate name, usage record property name, or any combination of the three. Because the constant can be a usage record property (something that gets recorded and shown when you do a glsusage) and a charge rate name, most people make charge rate names match the usage record property. For example, *Processors* is something that you probably want to both record and charge for, so it is easier if the charge rate name and usage record attribute name are both called *Processors* and `nami.cfg` has the line:

```
UsageAttribute[Processors]              MoabXMLAttribute=Processors
```

The `nami.cfg` file interprets this to mean that the value in the *Processors* tag sent from Moab should be the same value passed in the *Processors* tag in the usage record sent to Moab Accounting Manager. Moab Accounting Manager will then recognize that it is the usage record property name, so it will record it and show it when you do a glsusage command and will also recognize that it is a charge rate so it will charge it accordingly. Other attributes like user and project help constrain which account gets charged. However, there is nothing preventing you from making additional charge rates in Moab Accounting Manager for them as well.

Currently, the NAMI scripts allow anything in the Moab job XML to be sent to Moab Accounting Manager, including gres and user-defined variables. Gres and variables are expressed with either a *gres:* or *var:* prepended to the name of what it is representing. For example you might see the following in `nami.cfg`:

```
UsageAttribute[Disk]                MoabXMLAttribute=GRes:Os
UsageAttribute[StorageHost]         MoabXMLAttribute=Var:SH
```

Other Important nami.cfg Parameters

SetupTime - The amount of time (in seconds) beyond the bill cycle time that the reservation for the setup cost and first bill cycle should last before it destroys itself (defaults to 43200). Here is an example of how to define it in `nami.cfg`:

```
SetupTime                                   3213 (Defaults to 43200)
```

BillCycle - The amount of time of the bill cycle. It helps determine how much funds to reserve for each bill cycle. Possible values are hour, day, week, month (the default is day). If you want to over- or under-reserve funds for the bill cycle, you can also configure the default values for hour (60), day (86400), week (604800), and month (2630000). Be careful to understand that if you use this with RoundChargeDuration, the charge duration will be rounded to the modified bill cycle which is probably not what you wanted to do. Configuring the bill cycle time is intended to increase or decrease the barrier of entry for your job and assumes that the charge duration is not rounded. Here is an example of setting the bill cycle and changing the time duration of *Month*.

```
BILLCYCLE                  Month    (Defaults to Day)
BILLCYCLE[Month]           2640000 (Defaults to average seconds in a month)
```

RoundChargeDuration - If set to *True*, then the charge duration sent from Moab will be rounded up or down to match what was quoted within the default bounds of plus or minus 0 seconds. Otherwise, charge duration will not be rounded. The bound to round the charge duration is configurable with the **Bound** attribute. Here is an example of rounding the charge duration if it is within plus or minus a half a day:

```
RoundChargeDuration            True    (Defaults to False)
RoundChargeDuration[config]    Bound=43200  (Defaults to 0)
```

Log - If set to *True*, the `nami.log` will also include everything that is written to the `gold.log` in `/opt/mam/log/`. An example log configuration is:

```
LOG[config]     path=../log/nami.log    (Defaults to ../log/nami.log)
LOG[config]     loglevel=debug          (Possible values are fatal, error, warn, info,
debug and it defaults to info)
LOG[config]     includeGoldLog=false    (Defaults to true)
LOG[config]     permissions=640         (Defaults to 666)
LOG[config]     maxSize=10000000        (Defaults to 10000000)
LOG[config]     rolloverLimit=7         (Defaults to 7)
```

6.6 NAMI Queuing

Using the QueueNAMIActions parameter, the NAMI (Native Allocation Manager Interface) charge and destroy actions will be queued. Other actions are still fired by themselves.

All queues on an allocation manager will be flushed each flush interval.

NAMI for Jobs

There are five main events for billing jobs with NAMI. These are shown below in the job timeline:

```
 Job:  -------Quote--|--Submit------Start--------------------
-------------End----
                     |
 Nami:   -----Quote -|--Create-----Reserve--------Charge---
Charge---------Delete--
 Input:         XML    |   XML          XML            XML
XML             XML
 Output:      #[:TID] |  [TID]
```

All scripts will have return codes checked. For charge and delete, the return code will only cause Moab to log the failure (this is also because of the queuing).

Quoting is not done by Moab, but called by Viewpoint. If they get a TID, it should be attached as a variable on the job. If a TID is returned by the create call, it will be attached as a variable.

Each script will be passed an XML description of the object (the same scripts are used for both jobs and reservations). This XML is not the same as checkpoint or checkjob XML, it is special XML specifically for charging. It is simpler than the checkjob XML.

Also note that a final charge will always be called before the delete script.

Return Values

- **Quote** - Returns a number, and an optional TID. The TID is a string and will be stored as a variable on the job (or reservation). The format is #:TID, or just the number if there is no TID. This should only be called by Viewpoint. It is Viewpoint's responsibility to attach this to the job as a variable.
- **Create** - Anything in stdout will be assumed to be a TID. Return code is checked.
- **Reserve** - Return code is checked.
- **Charge** - Return code is checked (may be batched).
- **Delete** - Return code is checked (may be batched).
- **Modify** - Return code is checked.

You must configure the NAMITransVarName parameter to save the TID. The TID will be saved under a variable with that name. The TID may not contain equal sign (=) or colon (:) characters.

Billing Failure

If any call fails, Moab will log the failed action. Also, there will be two parameters on the allocation manager, BillingCreateFailure and BillingReserveFailure, that specify the action that will be taken when an action fails. Options will be IGNORE, HOLD, and CANCEL, with IGNORE being the default. Functionality may differ between jobs and reservations. For the StartFailureAction, you can also use RETRY, which will remove the job's reservation and will put it back into the queue to be scheduled the next iteration.

The RunAlways internal flag will cause jobs to ignore billing failures.

Job Approval

One of the great things about this billing system is that we get job approval very easily. The administrator puts BillingCreateFailure to HOLD. Next, they make the create script trigger their approval system (send an email, etc.) and return FAILURE. This will make the job always be put on hold. Then the administrator takes off the hold to approve the job (or reservation).

7.0 Controlling Resource Access - Reservations, Partitions, and QoS Facilities

- **Advance Reservations** on page 375
- **Partitions** on page 419
- **Quality of Service (QoS) Facilities** on page 423

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, and QoS support.

- Reservation Overview
- Administrative Reservations
- Standing Reservations
- Reservation Policies
- Configuring and Managing Reservations
- Enabling Reservations for End-users

7.1.1 Reservation Overview

- Resources
- TimeFrame
- Access Control List
- Job to Reservation Binding
- Reservation Specification

- Reservation Behavior
- 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.

Resources

Under Moab, the resources specified for a reservation are specified by way of a task description. Conceptually, a task can be thought of as an atomic, or indivisible, collection of resources. If reservation resources are unspecified, a task is a node by default. To define a task, specify resources. The resources may include processors, memory, swap, local disk, and so forth. For example, a single task may consist of one processor, 2 GB of memory, and 10 GB of local disk.

A reservation consists of one or more tasks. In attempting to locate the resources required for a particular reservation, Moab examines all feasible resources and locates the needed resources in groups specified by the task description. An example may help clarify this concept:

Reservation A requires four tasks. Each task is defined as 1 processor and 1 GB of memory.

Node X has 2 processors and 3 GB of memory available
Node Y has 2 processors and 1 GB of memory available
Node Z has 2 processors and 2 GB of memory available

When collecting the resources needed for the reservation, Moab examines each node in turn. Moab finds that Node X can support 2 of the 4 tasks needed by reserving 2 processors and 2 GB of memory, leaving 1 GB of memory unreserved. Analysis of Node Y shows that it can only support 1 task reserving 1 processor and 1 GB of memory, leaving 1 processor unreserved. Note that the unreserved memory on Node X cannot be combined with the unreserved processor on Node Y to satisfy the needs of another task because a task requires all resources to be located on the same node. Finally, analysis finds that node Z can support 2 tasks, fully reserving all of its resources.

Both reservations and jobs use the concept of a task description in specifying how resources should be allocated. It is important to note that although a task description is used to allocate resources to a reservation, this description does not in any way constrain the use of those resources by a job. In the above example, a job requesting resources simply sees 4 processors and 4 GB of memory available in reservation A. If the job has access to the reserved resources and the resources meet the other requirements of the job, the job could use these resources according to its own task description and needs.

Currently, the resources that can be associated with reservations include processors, memory, swap, local disk, initiator classes, and any number of arbitrary resources. Arbitrary resources may include peripherals such as tape drives, software licenses, or any other site specific resource.

Time Frame

Associated with each reservation is a time frame. This specifies when the resources will be reserved or dedicated to jobs that meet the reservation's access control list (ACL). The time frame simply consists of a start time and an end time. When configuring a reservation, this information may be specified as a start time together with either an end time or a duration.

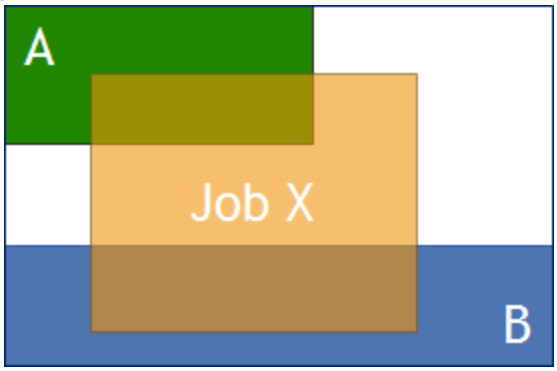
Access Control List

A reservation's access control list specifies which jobs can use a reservation. Only jobs that meet one or more of a reservation's access criteria are allowed to use the reserved resources during the reservation time frame. Currently, the reservation access criteria include the following: users, groups, accounts, classes, QoSes, job attributes, job duration, and job templates.

Job to Reservation Binding

While a reservation's ACL will allow particular jobs to use reserved resources, it does not force any job to use these resources. With each job, Moab attempts to locate the best possible combination of available resources whether these are reserved or unreserved. For example, in the following figure, note that job **X**, which meets access criteria for both reservation **A** and **B**, allocates a portion of its resources from each reservation and the remainder from resources outside of both reservations.

Image 7-1: Job X uses resources from reservations A and B



Although by default, reservations make resources available to jobs that meet particular criteria, Moab can be configured to constrain jobs to only run within accessible reservations. This can be requested by the user on a job by job basis using a resource manager extension flag, or it can be enabled administratively via a QoS flag. For example, assume two reservations were created as follows:

```
> mrsvctl -c -a GROUP==staff -d 8:00:00 -h 'node[1-4]'
reservation staff.1 created
```

```
> mrsvctl -c -a USER==john -t 2
reservation john.2 created
```

If the user "john," who happened to also be a member of the group "staff," wanted to force a job to run within a particular reservation, "john" could do so using the **FLAGS** resource manager extension. Specifically, in the case of a PBS job, the following submission would force the job to run within the "staff.1" reservation.

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

Note that for this to work, PBS needs to have resource manager extensions enabled as described in the PBS Resource Manager Extension Overview. (TORQUE has resource manager extensions enabled by

default.) If the user wants the job to run on reserved resources but does not care which, the user could submit the job with the following:

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

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

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

To lock jobs linked to a particular QoS into a reservation or reservation group, use the REQRID attribute.

Reservation Specification

There are two main types of reservations that sites typically deal with. The first, administrative reservations, are typically one-time reservations created for special purposes and projects. These reservations are created using the mrsvctl or setres commands. These reservations provide an integrated mechanism to allow graceful management of unexpected system maintenance, temporary projects, and time critical demonstrations. This command allows an administrator to select a particular set of resources or just specify the quantity of resources needed. For example an administrator could use a regular expression to request a reservation be created on the nodes "blue0[1-9]" or could simply request that the reservation locate the needed resources by specifying a quantity based request such as "TASKS==20."

The second type of reservation is called a standing reservation. It is specified using the SRCFG parameter and is of use when there is a recurring need for a particular type of resource distribution. Standing reservations are a powerful, flexible, and efficient means for enabling persistent or periodic policies such as those often enabled using classes or queues. For example, a site could use a standing reservation to reserve a subset of its compute resources for quick turnaround jobs during business hours on Monday thru Friday. The Standing Reservation Overview provides more information about configuring and using these reservations.

Reservation Behavior

As previously mentioned, a given reservation may have one or more access criteria. A job can use the reserved resources if it meets at least one of these access criteria. It is possible to stack multiple reservations on the same node. In such a situation, a job can only use the given node if it has access to each active reservation on the node.

Reservation Group

Reservations groups are ways of associating multiple reservations. This association is useful for variable name space and reservation requests. The reservations in a group inherit the variables from the reservation group head, but if the same variable is set locally on a reservation in the group, the local variable overrides the inherited variable. Variable inheritance is useful for triggers as it provides greater flexibility with automating certain tasks and system behaviors.

Jobs may be bound to a reservation group (instead of a single reservation) by using the resource manager extension ADVRES.

Infinite Jobs and Reservations

To allow infinite walltime jobs, you must have the following scheduler flag set:

```
SCHEDCFG[Moab] FLAGS=allowinfinitejobs
```

You can submit an infinite job by completing:

```
msub -l walltime=INFINITY
```

Or an infinite reservation by completing:

```
mrsvctl -c -d INFINITY
```

Infinite jobs can run in infinite reservations. Infinite walltime also works with job templates and advres.

Output XML for infinite jobs will print "INFINITY" in the ReqAWDuration, and XML for infinite rsvs will print "INFINITY" in duration and endtime.

```
<Data>
   <rsv AUser="jgardner" AllocNodeCount="1" AllocNodeList="n5"
   AllocProcCount="4" AllocTaskCount="1" HostExp="n5"
   LastChargeTime="0" Name="jgardner.1" Partition="base"
   ReqNodeList="n5:1" Resources="PROCS=[ALL]" StatCAPS="0.00"
   StatCIPS="0.00" StatTAPS="0.00" StatTIPS="0.00" SubType="Other"
   Type="User" cost="0.000000" ctime="1302127058"
   duration="INFINITY" endtime="INFINITY" starttime="1302127058">
     <ACL aff="neutral" cmp="%=" name="jgardner.1" type="RSV"></ACL>
     <ACL cmp="%=" name="jgardner" type="USER"></ACL>
     <ACL cmp="%=" name="company" type="GROUP"></ACL>
     <ACL aff="neutral" cmp="%=" name="jgardner.1" type="RSV"></ACL>
     <History>
       <event state="PROCS=4" time="1302127058"></event>
     </History>
   </rsv>
 </Data>
```

Related topics

- Reservation Allocation Policies
- Reservation Re-Allocation Policies

7.1.2 Administrative Reservations

- Annotating Administrative Reservations
- Using Reservation Profiles
- Optimizing Maintenance Reservations

Administrative reservations behave much like standing reservations but are generally created to address non-periodic, one-time issues. All administrative reservations are created using the mrsvctl -c (or setres) command and are persistent until they expire or are removed using the mrsvctl -r (or releaseres) command.

Annotating Administrative Reservations

Reservations can be labeled and annotated using comments allowing other administrators, local users, portals and other services to obtain more detailed information regarding the reservations. Naming and annotations are configured using the -n and -D options of the mrsvctl command respectively, as in the following example:

```
> mrsvctl -c -D 'testing infiniband performance' -n nettest -h 'r:agt[15-245]'
```

Using Reservation Profiles

You can set up reservation profiles to avoid manually and repetitively inputting standard reservation attributes. Profiles can specify reservation names, descriptions, ACLs, durations, host lists, triggers, flags, and other aspects that are commonly used. With a reservation profile defined, a new administrative reservation can be created that uses this profile by specifying the -P flag as in the following example.

Example 7-1:

```
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 7-2: 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
```

Optimizing Maintenance Reservations

Any reservation causes some negative impact on cluster performance as it further limits the scheduler's ability to optimize scheduling decisions. You can mitigate this impact by using flexible ACLs and triggers.

In particular, a maintenance reservation can be configured to reduce its effective reservation shadow by allowing overlap with checkpointable/preemptible jobs until the time the reservation becomes active. This can be done using a series of triggers that perform the following actions:

- Modify the reservation to disable preemption access.
- Preempt jobs that may overlap the reservation.
- Cancel any jobs that failed to properly checkpoint and exit.

The following example highlights one possible configuration:

```
RSVPROFILE[adm1] JOBATTRLIST=PREEMPTEE
RSVPROFILE[adm1] DESCRIPTION="regular system maintenance"
RSVPROFILE[adm1] TRIGGER=EType=start,Offset=-
```

```
300,AType=internal,Action="rsv:-:modify:acl:jattr-=PREEMPTEE"
RSVPROFILE[adm1] TRIGGER=EType=start,Offset=-240,AType=jobpreempt,Action="checkpoint"
RSVPROFILE[adm1] TRIGGER=EType=start,Offset=-60,AType=jobpreempt,Action="cancel"
```

```
> mrsvctl -c -P adm1 -s 12:00:00_10/03 -d 8:00:00 -h ALL
```

This reservation reserves all nodes in the cluster for a period of eight hours. Five minutes before the reservation starts, the reservation is modified to remove access to new preemptible jobs. Four minutes before the reservation starts, preemptible jobs that overlap the reservation are checkpointed. One minute before the reservation, all remaining jobs that overlap the reservation are canceled.

Reservations can also be used to evacuate virtual machines from a nodelist. To do this, you can configure a reservation profile in the `moab.cfg` file that calls an internal trigger to enable the evacuate VM logic. For example:

```
RSVPROFILE[evacvms]
TRIGGER=EType=start,AType=internal,action=node:$(HOSTLIST):evacvms
```

```
> mrsvctl -c -P evacvms -s 12:00:00_10/03 -d 8:00:00 -h ALL
```

Please note that Moab gives its best effort in evacuating VMs; however, if other reservations and policies prevent Moab from locating an alternate location for the VMs to be migrated to, then no action will occur. Administrators can attach additional triggers to the reservation profile to add evacuation logic where needed.

If your organization uses Viewpoint 7.1 or later, there is an option when creating reservations in Viewpoint to evacuate VMs from reserved nodes. This functionality assumes the reservation profile in Moab is named "evacvms." For Cloud customers, the evacvms reservation profile already exists in your `moab.cfg` file configuration by default.

You can also manually create a reservation that evacuates VMs from a nodelist by using the **EVACVMS** reservation flag. For example:

```
> mrsvctl -c -F EVACVMS -s 12:00:00_10/03 -d 8:00:00 -h ALL
```

Related topics

- Backfill
- Preemption
- mrsvctl command

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 job responsiveness and turnaround.

Related topics

- SRCFG
- mdiag -s (diagnose standing reservations)

7.1.4 Reservation Policies

- Controlling Priority Reservation Creation
- Managing Resource Failures
- Resource Allocation Policy
- Resource Re-Allocation Policy
- Charging for Reserved Resources

Controlling Priority Reservation Creation

In addition to standing and administrative reservations, Moab can also create priority reservations. These reservations are used to allow the benefits of out-of-order execution (such as is available with backfill) without the side effect of job starvation. Starvation can occur in any system where the potential exists for a job to be overlooked by the scheduler for an indefinite period. In the case of backfill, small jobs may continue to run on available resources as they become available while a large job sits in the queue, never able to find enough nodes available simultaneously on which to run.

To avoid such situations, priority reservations are created for high priority jobs that cannot run immediately. When making these reservations, the scheduler determines the earliest time the job could start and then reserves these resources for use by this job at that future time.

Priority Reservation Creation Policy

Organizations have the ability to control how priority reservations are created and maintained. It is possible that one job can be at the top of the priority queue for a time and then get bypassed by another job submitted later. The parameter RESERVATIONPOLICY allows a site to determine how existing reservations should be handled when new reservations are made.

Value	Description

HIGHEST	All jobs that have ever received a priority reservation up to the RESERVATIONDEPTH number will maintain that reservation until they run, even if other jobs later bypass them in priority value.
	For example, if there are four jobs with priorities of 8, 10,12, and 20 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.
	With *HIGHEST*, Moab may appear to exceed the RESERVATIONDEPTH if it has already scheduled the maximum number of priority reservations and then users submit jobs with higher priority than those already given a priority reservation. Moab keeps all of the previously-created priority reservations and creates new ones for jobs with higher priority (again up to the quantity specified with RESERVATIONDEPTH). This means that, if your RESERVATIONDEPTH is set to 3, Moab can potentially schedule up to 3 new priority reservations each scheduling iteration, as long as new higher-priority jobs are continually submitted. This behavior ensures that the highest-priority jobs receive attention while the former highest-priority jobs do not lose their priority reservation.

CURRENTHIGHEST	Only the current top <RESERVATIONDEPTH> priority jobs receive reservations. Under this policy, all job reservations are destroyed each iteration when the queue is re-prioritized. The top jobs in the queue are then given new reservations.
NEVER	No priority reservations are made.

Priority Reservation Depth

By default, only the highest priority job receives a priority reservation. However, this behavior is configurable via the RESERVATIONDEPTH policy. Moab's default behavior of only reserving the highest priority job allows backfill to be used in a form known as liberal backfill. Liberal backfill tends to maximize system utilization and minimize overall average job turnaround time. However, it does lead to the potential of some lower priority jobs being indirectly delayed and may lead to greater variance in job turnaround time. The RESERVATIONDEPTH parameter can be set to a very large value, essentially enabling what is called conservative backfill where every job that cannot run is given a reservation. Most sites prefer the liberal backfill approach associated with the default RESERVATIONDEPTH of 1 or else select a slightly higher value. It is important to note that to prevent starvation in conjunction with reservations, monotonically increasing priority factors such as queue time or job XFactor should be enabled. See the Prioritization Overview for more information on priority factors.

Another important consequence of backfill and reservation depth is how they affect job priority. In Moab, all jobs are prioritized. Backfill allows jobs to be run out of order and thus, to some extent, job priority to be ignored. This effect, known as priority dilution, can cause many site policies implemented via Moab prioritization policies to be ineffective. Setting the RESERVATIONDEPTH parameter to a higher value

gives job priority more teeth at the cost of slightly lower system utilization. This lower utilization results from the constraints of these additional reservations, decreasing the scheduler's freedom and its ability to find additional optimizing schedules. Anecdotal evidence indicates that these utilization losses are fairly minor, rarely exceeding 8%.

It is difficult a priori to know the right setting for the **RESERVATIONDEPTH** parameter. Surveys indicate that the vast majority of sites use the default value of 1. Sites that do modify this value typically set it somewhere in the range of 2 to 10. The following guidelines may be useful in determining if and how to adjust this parameter:

Reasons to Increase RESERVATIONDEPTH

- The estimated job start time information provided by the showstart command is heavily used and the accuracy needs to be increased.
- Priority dilution prevents certain key mission objectives from being fulfilled.
- Users are more interested in knowing when their job will run than in having it run sooner.

Reasons to Decrease RESERVATIONDEPTH

- Scheduling efficiency and job throughput need to be increased.

Assigning Per-QoS Reservation Creation Rules

QoS based reservation depths can be enabled via the RESERVATIONQOSLIST parameter. This parameter allows varying reservation depths to be associated with different sets of job QoSes. For example, the following configuration creates two reservation depth groupings:

```
RESERVATIONDEPTH[0]   8
RESERVATIONQOSLIST[0] highprio,interactive,debug
RESERVATIONDEPTH[1]   2
RESERVATIONQOSLIST[1] batch
```

This example causes that the top 8 jobs belonging to the aggregate group of highprio, interactive, and debug QoS jobs will receive priority reservations. Additionally, the top two batch QoS jobs will also receive priority reservations. Use of this feature allows sites to maintain high throughput for important jobs by guaranteeing that a significant proportion of these jobs progress toward starting through use of the priority reservation.

By default, the following parameters are set inside Moab:

```
RESERVATIONDEPTH[DEFAULT]   1
RESERVATIONQOSLIST[DEFAULT] ALL
```

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

Managing Resource Failures

Moab allows organizations to control how to best respond to a number of real-world issues. Occasionally when a reservation becomes active and a job attempts to start, various resource manager race conditions or corrupt state situations will prevent the job from starting. By default, Moab assumes the resource manager is corrupt, releases the reservation, and attempts to re-create the reservation after a short timeout. However, in the interval between the reservation release and the re-creation timeout, other priority reservations may allocate the newly available resources, reserving them before the

original reservation gets an opportunity to reallocate them. Thus, when the original job reservation is re-established, its original resource may be unavailable and the resulting new reservation may be delayed several hours from the earlier start time. The parameter RESERVATIONRETRYTIME allows a site that is experiencing frequent resource manager race conditions and/or corruption situations to tell Moab to hold on to the reserved resource for a period of time in an attempt to allow the resource manager to correct its state.

Resource Allocation Policy

By default, when a standing or administrative reservation is created, Moab allocates nodes in accordance with the specified task count, node expression, node constraints, and the MINRESOURCE node allocation policy.

Charging for Reserved Resources

Either Moab HPC Suite 7.0 - Enterprise Edition or Moab Cloud Suite 7.0 are required for support of charging and allocation management capabilities.

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 7-3: 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 7-4: 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.

Related topics

- Reservation Overview
- Backfill

7.1.5 Configuring and Managing Reservations

- Reservation Attributes
  - Start/End Time
  - Access Control List (ACL)
  - Selecting Resources
  - Flags
- Configuring and Managing Standing Reservations
  - Standing Reservation Attributes
  - Standing Reservation Overview
  - Specifying Reservation Resources
  - Enforcing Policies Via Multiple Reservations
  - Affinity
  - ACL Modifiers
  - Reservation Ownership
  - Partitions
  - Resource Allocation Behavior
  - Rolling Reservations
  - Modifying Resources with Standing Reservations
- Managing Administrative Reservations

Reservation Attributes

All reservations possess a time frame of activity, an access control list (ACL), and a list of resources to be reserved. Additionally, reservations may also possess a number of extension attributes including epilog/prolog specification, reservation ownership and accountability attributes, and special flags that modify the reservation's behavior.

Start/End Time

All reservations possess a start and an end time that define the reservation's active time. During this active time, the resources within the reservation may only be used as specified by the reservation access control list (ACL). This active time may be specified as either a start/end pair or a start/duration pair. Reservations exist and are visible from the time they are created until the active time ends at which point they are automatically removed.

Access Control List (ACL)

For a reservation to be useful, it must be able to limit who or what can access the resources it has reserved. This is handled by way of an ACL. With reservations, ACLs can be based on credentials, resources requested, or performance metrics. In particular, with a standing reservation, the attributes

USERLIST, GROUPLIST, ACCOUNTLIST, CLASSLIST, QOSLIST, JOBATTRLIST, PROCLIMIT, MAXTIME, or TIMELIMIT may be specified. (See Affinity and Modifiers.)

Reservation access can be adjusted based on a job's requested node features by mapping node feature requests to job attributes as in the following example:

```
NODECFG[DEFAULT]  FEATURES+=ia64
NODETOJOBATTRMAP  ia64,ia32
SRCFG[pgs]        JOBATTRLIST=ia32
```

```
> mrsvctl -c -a jattr=gpfs\! -h "r:13-500"
```

Selecting Resources

When specifying which resources to reserve, the administrator has a number of options. These options allow control over how many resources are reserved and where they are reserved. The following reservation attributes allow the administrator to define resources.

Task Description

Moab uses the task concept extensively for its job and reservation management. A task is simply an atomic collection of resources, such as processors, memory, or local disk, which must be found on the same node. For example, if a task requires 4 processors and 2 GB of memory, the scheduler must find all processors AND memory on the same node; it cannot allocate 3 processors and 1 GB on one node and 1 processor and 1 GB of memory on another node to satisfy this task. Tasks constrain how the scheduler must collect resources for use in a standing reservation; however, they do not constrain the way in which the scheduler makes these cumulative resources available to jobs. A job can use the resources covered by an accessible reservation in whatever way it needs. If reservation X allocates 6 tasks with 2 processors and 512 MB of memory each, it could support job Y which requires 10 tasks of 1 processor and 128 MB of memory or job Z which requires 2 tasks of 4 processors and 1 GB of memory each. The task constraints used to acquire a reservation's resources are transparent to a job requesting use of these resources.

Example 7-5:

```
SRCFG[test] RESOURCES=PROCS:2,MEM:1024
```

Task Count

Using the task description, the **TASKCOUNT** attribute defines how many tasks must be allocated to satisfy the reservation request. To create a reservation, a task count and/or a host list must be specified.

Example 7-6:

```
SRCFG[test] TASKCOUNT=256
```

Host List

A host list constrains the set of resources available to a reservation. If no task count is specified, the reservation attempts to reserve one task on each of the listed resources. If a task count is specified that requests fewer resources than listed in the host list, the scheduler reserves only the number of tasks

from the host list specified by the task count attribute. If a task count is specified that requests more resources than listed in the host list, the scheduler reserves the host list nodes first and then seeks additional resources outside of this list.

Example 7-7:

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

Node Features

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

Example 7-8:

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

Partition

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

Example 7-9:

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

Flags

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

Flag Name	Description
ACLOVERLAP	**Deprecated (this is now a default flag).** In addition to free or idle nodes, a reservation may also reserve resources that possess credentials that meet the reservation's ACL. To change this behavior, set the **NOACLOVERLAP** on page 390 flag.
ADVRESJOBDESTROY	All jobs that have an ADVRES matching this reservation are canceled when the reservation is destroyed.
ALLOWJOBOVERLAP	A job is allowed to start in a reservation that may end before the job completes. When the reservation ends before the job completes, the job will not be canceled but will continue to run.
BYNAME	Reservation only allows access to jobs that meet reservation ACLs and explicitly request the resources of this reservation using the job ADVRES flag. (See Job to Reservation Binding.)

Flag Name	Description
DEDICATEDRESOURCE (aka EXCLUSIVE)	Reservation only placed on resources that are not reserved by any other reservation including job, system, and user reservations (except when combined with **IGNJOBRSV***). The order that SRCFG reservations are listed in the configuration is important when using *DEDICATEDRESOURCE*, because reservations made afterwards can steal resources later. During configuration, list *DEDICATEDRESOURCE* reservations last to guarantee exclusiveness.
EVACVMS	Reservation will automatically evacuate virtual machines from the reservation nodelist. The same action can be accomplished by using reservation profiles. For more information, see **Optimizing Maintenance Reservations** on page 380.
IGNIDLEJOBS*	Reservation can be placed on top of idle job reservations. This flag is meant to be used in conjunction with *DEDICATEDRESOURCE*.
IGNJOBRSV*	Ignores existing job reservations, allowing the reservation to be forced onto available resources even if it conflicts with existing job reservations. User and system reservation conflicts are still valid. It functions the same as IGNIDLEJOBS plus allows a reservation to be placed on top of an existing running job's reservation. This flag is meant to be used in conjunction with *DEDICATEDRESOURCE*.
IGNRSV*	Request ignores existing resource reservations allowing the reservation to be forced onto available resources even if this conflicts with other reservations. It functions the same as IGNJOBRSV plus allows the reservation to be placed on top of the system reservations. This flag is meant to be used in conjunction with *DEDICATEDRESOURCE*.
IGNSTATE*	Reservation ignores node state when assigning nodes. It functions the same as IGNRSV plus allows the reservation to be placed on nodes that are not currently available. Also ignores resource availability on nodes.

Flag Name	Description
NOACLOVERLAP	All resources must be free or idle, with no existing reservations. Moab will not allocate in-use resources even if they match the reservation's ACL. `mrsvctl -c -t 12 -E -F noacloverlap -a user==john` *Moab looks for resources that are exclusive (free). Without the flag, Moab would look for resources that are exclusive or that are already running `john`'s jobs.* This flag is meant to be used in conjunction with *DEDICATEDRESOURCE*.
NOCHARGE	By default, Moab charges the allocation manager for unused cycles in a standing reservation. Setting the *NOCHARGE* flag prevents Moab from charging the allocation manager for standing reservations.
NOVMMIGRATION	If set on a reservation, this prevents VMs from being migrated away from the reservation. If there are multiple reservations on the hypervisor and at least one reservation does not have the *NOVMIGRATION* flag, then VMs will be migrated.
OWNERPREEMPT	Jobs by the reservation owner are allowed to preempt non-owner jobs using reservation resources.
OWNERPREEMPTIGNOREMINTIME	Allows the *OWNERPREEMPT* flag to "trump" the **PREEMPTMINTIME** setting for jobs already running on a reservation when the owner of the reservation submits a job. For example: without the *OWNERPREEMPTIGNOREMINTIME* flag set, a job submitted by the owner of a reservation will not preempt non-owner jobs already running on the reservation until the **PREEMPTMINTIME** setting (if set) for those jobs is passed. With the *OWNERPREEMPTIGNOREMINTIME* flag set, a job submitted by the owner of a reservation immediately preempts non-owner jobs already running on the reservation, regardless of whether **PREEMPTMINTIME** is set for the non-owner jobs.
REQFULL	Reservation is only created when all resources can be allocated.
SINGLEUSE	Reservation is automatically removed after completion of the first job to use the reserved resources.

Flag Name	Description
SPACEFLEX	**Deprecated (this is now a default flag).** Reservation is allowed to adjust resources allocated over time in an attempt to optimize resource utilization.

* *IGNIDLEJOBS*, *IGNJOBRSV*, *IGNRSV*, and *IGNSTATE* flags are built on one another and form a hierarchy. *IGNJOBRSV* performs the function of *IGNIDLEJOBS* plus its own functions. *IGNRSV* performs the function of *IGNJOBSRV* and *IGNIDLEJOBS* plus its own functions. *IGNSTATE* performs the function of *IGNRSV*, *IGNJOBRSV*, and *IGNIDLEJOBS* plus its own functions. While you can use combinations of these flags, it is not necessary. If you set one flag, you do not need to set other flags that fall beneath it in the hierarchy.

Most flags can be associated with a reservation via the mrsvctl -c -F command or the SRCFG parameter.

Configuring Standing Reservations

Standing reservations allow resources to be dedicated for particular uses. This dedication can be configured to be permanent or periodic, recurring at a regular time of day and/or time of week. There is extensive applicability of standing reservations for everything from daily dedicated job runs to improved use of resources on weekends. By default, standing reservations can overlap other reservations. Unless you set an ignore-type flag (*ACLOVERLAP*, *DEDICATEDRESOURCE*, *IGNIDLEJOBS*, or *IGNJOBRSV*), they are automatically given the *IGNRSV* flag. All standing reservation attributes are specified via the SRCFG parameter using the attributes listed in the table below.

Standing Reservation Attributes

ACCESS	
Format	*DEDICATED* or *SHARED*
Default	---
Description	If set to *SHARED*, allows a standing reservation to use resources already allocated to other non-job reservations. Otherwise, these other reservations block resource access.
Example	`SRCFG[test] ACCESS=SHARED` *Standing reservation test may access resources allocated to existing standing and administrative reservations.* The order that SRCFG reservations are listed in the configuration are important when using *DEDICATED*, because reservations made afterwards can steal resources later. During configuration, list *DEDICATED* reservations last to guarantee exclusiveness.

ACCOUNTLIST

Format	List of valid, comma delimited account names (see ACL Modifiers).
Default	---
Description	Specifies that jobs with the associated accounts may use the resources contained within this reservation.
Example	`SRCFG[test] ACCOUNTLIST=ops,staff` *Jobs using the account ops or staff are granted access to the resources in standing reservation test.*

CHARGEACCOUNT

Format	Any valid accountname.
Default	---
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 math.

CHARGEUSER

Format	Any valid username.
Default	---

CHARGEUSER

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.

COMMENT

Format

<STRING>

> If the string contains whitespace, it should be enclosed in single (') or double quotes (").

Default

Description

Specifies a descriptive message associated with the standing reservation and all child reservations.

COMMENT	
Example	`SRCFG[test] COMMENT='rsv for network testing'` *Moab annotates the standing reservation test and all child reservations with the specified message. These messages show up within Moab client commands, Moab web tools, and graphical administrator tools.*

DAYS	
Format	One or more of the following (comma-delimited): • *Mon* • *Tue* • *Wed* • *Thu* • *Fri* • *Sat* • *Sun* • *[ALL]*
Default	*[ALL]*
Description	Specifies which days of the week the standing reservation is active.
Example	`SRCFG[test] DAYS=Mon,Tue,Wed,Thu,Fri` *Standing reservation test is active Monday through Friday.*

DEPTH	
Format	<INTEGER>
Default	*2*

DEPTH	
Description	Specifies the depth of standing reservations to be created (one per period). To satisfy the **DEPTH**, Moab creates new reservations at the beginning of the specified **PERIOD** on page 400. If your reservation ends at the same time that a new **PERIOD** begins, the number of reservations may not match the requested **DEPTH**. To prevent or resolve this issue, set the **ENDTIME** on page 395 a couple minutes before the beginning of the next **PERIOD**. For example, set the **ENDTIME** to *23:58* instead of *00:00*.
Example	`SRCFG[test] PERIOD=DAY DEPTH=6` *Specifies that six reservations will be created for standing reservation test.*

DISABLE	
Format	<BOOLEAN>
Default	*FALSE*
Description	Specifies that the standing reservation should no longer spawn child reservations.
Example	`SRCFG[test] PERIOD=DAY DEPTH=7 DISABLE=TRUE` *Specifies that reservations are created for standing reservation test for today and the next six days.*

ENDTIME	
Format	[[[DD:]HH:]MM:]SS
Default	*24:00:00*
Description	Specifies the time of day the standing reservation period ends (end of day or end of week depending on **PERIOD**).

Example	`SRCFG[test] STARTTIME=8:00:00`
	`SRCFG[test] ENDTIME=17:00:00`
	`SRCFG[test] PERIOD=DAY`
	Standing reservation test is active from 8:00 AM until 5:00 PM.

FLAGS	
Format	Comma-delimited list of zero or more flags listed in the reservation flags overview.
Default	---
Description	Specifies special reservation attributes. See Managing Reservations - Flags for details.
Example	`SRCFG[test] FLAGS=BYNAME,DEDICATEDRESOURCE` *Jobs may only access the resources within this reservation if they explicitly request the reservation by name. Further, the reservation is created to not overlap with other reservations.*

GROUPLIST	
Format	One or more comma-delimited group names.
Default	*[ALL]*
Description	Specifies the groups allowed access to this standing reservation (see ACL Modifiers).

Example	`SRCFG[test] GROUPLIST=staff,ops,special`
	`SRCFG[test] CLASSLIST=interactive`
	Moab allows jobs with the listed group IDs or which request the job class interactive to use the resources covered by the standing reservation.

HOSTLIST	
Format	One or more comma delimited host names or *host expressions* or the string "class:<classname>".
Default	---

HOSTLIST

Description

Specifies the set of hosts that the scheduler can search for resources to satisfy the reservation. If specified using the "class:X" format, Moab only selects hosts that support the specified class. If **TASKCOUNT** is also specified, only **TASKCOUNT** tasks are reserved. Otherwise, all matching hosts are reserved.

The **HOSTLIST** attribute is treated as host regular expression so `foo10` will map to `foo10`, `foo101`, `foo1006`, and so forth. To request an exact host match, the expression can be bounded by the carat and dollar symbol expression markers as in `^foo10$`.

Example

```
SRCFG[test] HOSTLIST=node001,node002,node003
SRCFG[test] RESOURCES=PROCS:2;MEM:512
SRCFG[test] TASKCOUNT=2
```

Moab reserves a total of two tasks with 2 processors and 512 MB each, using resources located on node001, node002, and/or node003.

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

The reservation will consume all nodes that have "node01" somewhere in their names and all nodes that have both "node1" and either a "3," "4," or "5" in their names.

```
SRCFG[test] HOSTLIST=r:node[1-6]
```

The reservation will consume all nodes with names that begin with "node" and end with any number 1 through 6. In other words, it will reserve node1, node2, node3, node4, node5, and node6.

JOBATTRLIST

Format

Comma-delimited list of one or more of the following job attributes:

- *PREEMPTEE*
- *INTERACTIVE*
- any generic attribute configured through NODECFG.

Default

JOBATTRLIST	
Description	Specifies job attributes that grant a job access to the reservation. Values can be specified with a "!="assignment to only allow jobs NOT requesting a certain feature inside the reservation. To enable/disable reservation access based on requested node features, use the parameter NODETOJOBATTRMAP.
Example	`SRCFG[test] JOBATTRLIST=PREEMPTEE` *Preemptible jobs can access the resources reserved within this reservation.*

MAXJOB	
Format	<INTEGER>
Default	---
Description	Specifies the maximum number of jobs that can run in the reservation.
Example	`SRCFG[test] MAXJOB=1` *Only one job will be allowed to run in this reservation.*

MAXTIME	
Format	[[[DD:]HH:]MM:]SS[+]
Default	---
Description	Specifies the maximum time for jobs allowable. Can be used with Affinity to attract jobs with same **MAXTIME**.
Example	`SRCFG[test] MAXTIME=1:00:00+` *Jobs with a time of 1:00:00 are attracted to this reservation.*

NODEFEATURES	
Format	Comma-delimited list of node features.
Default	---
Description	Specifies the required node features for nodes that are part of the standing reservation.
Example	`SRCFG[test] NODEFEATURES=wide,fddi` *All nodes allocated to the standing reservation must have both the wide and fddi node attributes.*

OWNER	
Format	<CREDTYPE>:<CREDID> Where *<CREDTYPE>* is one of *USER*, *GROUP*, *ACCT*, *QoS*, *CLASS* or *CLUSTER* and *<CREDTYPE>* is a valid credential id of that type.
Default	---
Description	Specifies the owner of the reservation. Setting ownership for a reservation grants the user management privileges, including the power to release it. Setting a *USER* as the *OWNER* of a reservation gives that user privileges to query and release the reservation. For sandbox reservations, sandboxes are applied to a specific peer only if *OWNER* is set to *CLUSTER:<PEERNAME>*.
Example	`SRCFG[test] OWNER=ACCT:jupiter` *User jupiter owns the reservation and may be granted special privileges associated with that ownership.*

PARTITION	
Format	Valid partition name.
Default	*[ALL]*

PARTITION	
Description	Specifies the partition in which to create the standing reservation.
Example	`SRCFG[test] PARTITION=OLD` *The standing reservation will only select resources from partition OLD.*

PERIOD	
Format	One of *DAY*, *WEEK*, or *INFINITY*.
Default	*DAY*
Description	Specifies the `period` of the standing reservation.
Example	`SRCFG[test] PERIOD=WEEK` Each standing reservation covers a one week period.

PROCLIMIT	
Format	<QUALIFIER><INTEGER> <*QUALIFIER*> may be one of the following <, <=, ==, >=, >
Default	---
Description	Specifies the processor limit for jobs requesting access to this standing reservation.
Example	`SRCFG[test] PROCLIMIT<=4` *Jobs requesting 4 or fewer processors are allowed to run.*

PSLIMIT	
Format	<QUALIFIER><INTEGER> <*QUALIFIER*> may be one of the following <, <=, ==, >=, >

PSLIMIT	
Default	---
Description	Specifies the processor-second limit for jobs requesting access to this standing reservation.
Example	`SRCFG[test] PSLIMIT<=40000` *Jobs requesting 40000 or fewer processor-seconds are allowed to run.*

QOSLIST	
Format	Zero or more valid, comma-delimited QoS names.
Default	---
Description	Specifies that jobs with the listed QoS names can access the reserved resources.
Example	`SRCFG[test] QOSLIST=hi,low,special` *Moab allows jobs using the listed QoSes access to the reserved resources.*

REQUIREDTPN	
Format	<QUALIFIER><INTEGER> <*QUALIFIER*> may be one of the following <, <=, ==, >=, >
Default	---
Description	Restricts access to reservations based on the job's TPN (tasks per node).
Example	`SRCFG[test] REQUIREDTPN==4` *Jobs with tpn=4 or ppn=4 would be allowed within the reservation, but any other TPN value would not. (For more information, see* **TPN (Exact Tasks Per Node)** on page 404*.)*

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. Neither credlock nor advres are compatible on the jobs submitted for this reservation.

Example	`SRCFG[ajax] ROLLBACKOFFSET=24:00:00 TASKCOUNT=32`
	`SRCFG[ajax] PERIOD=INFINITY ACCOUNTLIST=ajax`
	The standing reservation guarantees access to up to 32 processors within 24 hours to jobs from the ajax account.
	Adding an asterisk to the **ROLLBACKOFFSET** value pins rollback reservation start times when an idle reservation is created in the rollback reservation. For example:
	`SRCFG[staff] ROLLBACKOFFSET=18:00:00* PERIOD=INFINITY`

RSVACCESSLIST	
Format	<RESERVATION>[,...]
Default	---
Description	A list of reservations to which the specified reservation has access.
Example	`SRCFG[test] RSVACCESSLIST=rsv1,rsv2,rsv3`

RSVGROUP	
Format	<STRING>
Default	---
Description	See section Reservation Group for a detailed description.
Example	`SRCFG[test] RSVGROUP=rsvgrp1` `SRCFG[ajax] RSVGROUP=rsvgrp1`

STARTTIME	
Format	[[[DD:]HH:]MM:]SS
Default	*00:00:00:00* (midnight)
Description	Specifies the time of day/week the standing reservation becomes active. Whether this indicates a time of day or time of week depends on the setting of the **PERIOD** attribute. If specified within a reservation profile, a value of 0 indicates the reservation should start at the earliest opportunity.

Example	`SRCFG[test] STARTTIME=08:00:00`
	`SRCFG[test] ENDTIME=17:00:00`
	`SRCFG[test] PERIOD=DAY`
	The standing reservation will be active from 8:00 a.m. until 5:00 p.m. each day.

TASKCOUNT

Format	<INTEGER>
Default	*0* (unlimited tasks)
Description	Specifies how many tasks should be reserved for the reservation.

Example	`SRCFG[test] RESOURCES=PROCS:1;MEM:256`
	`SRCFG[test] TASKCOUNT=16`
	Standing reservation test reserves 16 tasks worth of resources; in this case, 16 processors and 4 GB of real memory.

TIMELIMIT

Format	[[[DD:]HH:]MM:]SS
Default	*-1* (no time based access)
Description	Specifies the maximum allowed overlap between the standing reservation and a job requesting resource access.
Example	`SRCFG[test] TIMELIMIT=1:00:00` *Moab allows jobs to access up to one hour of resources in the standing reservation.*

TPN (Exact Tasks Per Node)

Format	<INTEGER>
Default	*0* (no TPN constraint)
Description	Specifies the exact number of tasks per node that must be available on eligible nodes.

Example	`SRCFG[2] TPN=4`
	`SRCFG[2] RESOURCES=PROCS:2;MEM:256`
	Moab must locate four tasks on each node that is to be part of the reservation. That is, each node included in standing reservation 2 must have 8 processors and 1 GB of memory available.

TRIGGER	
Format	See **Creating a trigger** on page 658 for syntax.
Default	N/A
Description	Specifies event triggers to be launched by the scheduler under the scheduler's ID. These triggers can be used to conditionally cancel reservations, modify resources, or launch various actions at specified event offsets. See **About object triggers** on page 655 for more information.
Example	`SRCFG[fast] TRIGGER=EType=start,Offset=5:00:00,AType=exec,Action="/usr/local/domail.pl"` *Moab launches the `domail.pl` script 5 hours after any fast reservation starts.*

USERLIST	
Format	Comma-delimited list of users.
Default	---
Description	Specifies which users have access to the resources reserved by this reservation (see ACL Modifiers).
Example	`SRCFG[test] USERLIST=bob,joe,mary` *Users bob, joe and mary can all access the resources reserved within this reservation.*

Standing Reservation Overview

A standing reservation is similar to a normal administrative reservation in that it also places an access control list on a specified set of resources. Resources are specified on a per-task basis and currently include processors, local disk, real memory, and swap. The access control list supported for standing reservations includes users, groups, accounts, job classes, and QoS levels. Standing reservations can be configured to be permanent or periodic on a daily or weekly basis and can accept a daily or weekly start and end time. Regardless of whether permanent or recurring on a daily or weekly basis, standing reservations are enforced using a series of reservations, extending a number of periods into the future as controlled by the **DEPTH** attribute of the SRCFG parameter.

The following examples demonstrate possible configurations specified with the **SRCFG** parameter.

Example 7-10: Basic Business Hour Standing Reservation

```
SRCFG[interactive] TASKCOUNT=6 RESOURCES=PROCS:1,MEM:512
SRCFG[interactive] PERIOD=DAY DAYS=MON,TUE,WED,THU,FRI
```

```
SRCFG[interactive] STARTTIME=9:00:00 ENDTIME=17:00:00
SRCFG[interactive] CLASSLIST=interactive
```

When using the SRCFG parameter, attribute lists must be delimited using the comma (,), pipe (|), or colon (:) characters; they cannot be space delimited. For example, to specify a multi-class ACL, specify:

```
SRCFG[test] CLASSLIST=classA,classB
```

Only one **STARTTIME** and one **ENDTIME** value can be specified per reservation. If varied start and end times are desired throughout the week, complementary standing reservations should be created. For example, to establish a reservation from 8:00 p.m. until 6:00 a.m. the next day during business days, two reservations should be created-one from 8:00 p.m. until midnight, and the other from midnight until 6:00 a.m. Jobs can run across reservation boundaries allowing these two reservations to function as a single reservation that spans the night. The following example demonstrates how to span a reservation across 2 days on the same nodes:

```
SRCFG[Sun] PERIOD=WEEK
SRCFG[Sun] STARTTIME=00:20:00:00 ENDTIME=01:00:00:00
SRCFG[Sun] HOSTLIST=node01,node02,node03

SRCFG[Mon] PERIOD=WEEK
SRCFG[Mon] STARTTIME=01:00:00:00 ENDTIME=01:06:00:00
SRCFG[Sun] HOSTLIST=node01,node02,node03
```

The preceding example fully specifies a reservation including the quantity of resources requested using the **TASKCOUNT** and **RESOURCES** attributes. In all cases, resources are allocated to a reservation in units called tasks where a task is a collection of resources that must be allocated together on a single node. The **TASKCOUNT** attribute specifies the number of these tasks that should be reserved by the reservation. In conjunction with this attribute, the **RESOURCES** attribute defines the reservation task by indicating what resources must be included in each task. In this case, the scheduler must locate and reserve 1 processor and 512 MB of memory together on the same node for each task requested.

As mentioned previously, a standing reservation reserves resources over a given time frame. The **PERIOD** attribute may be set to a value of *DAY*, *WEEK*, or *INFINITY* to indicate the period over which this reservation should recur. If not specified, a standing reservation recurs on a daily basis. If a standing reservation is configured to recur daily, the attribute **DAYS** may be specified to indicate which days of the week the reservation should exist. This attribute takes a comma-delimited list of days where each day is specified as the first three letters of the day in all capital letters: *MON* or *FRI*. The preceding example specifies that this reservation is periodic on a daily basis and should only exist on business days.

The time of day during which the requested tasks are to be reserved is specified using the **STARTTIME** and **ENDTIME** attributes. These attributes are specified in standard military time HH:MM:SS format and both **STARTTIME** and **ENDTIME** specification is optional defaulting to midnight at the beginning and end of the day respectively. In the preceding example, resources are reserved from 9:00 a.m. until 5:00 p.m. on business days.

The final aspect of any reservation is the access control list indicating who or what can use the reserved resources. In the preceding example, the **CLASSLIST** attribute is used to indicate that jobs requesting the class "interactive" should be allowed to use this reservation.

Specifying Reservation Resources

In most cases, only a small subset of standing reservation attributes must be specified in any given case. For example, by default, **RESOURCES** is set to `PROCS=-1` which indicates that each task should reserve all of the processors on the node on which it is located. This, in essence, creates a one task equals one node mapping. In many cases, particularly on uniprocessor systems, this default behavior may be easiest to work with. However, in SMP environments, the **RESOURCES** attribute provides a powerful means of specifying an exact, multi-dimensional resource set.

An examination of the parameters documentation shows that the default value of **PERIOD** is *DAYS*. Thus, specifying this parameter in the preceding above was unnecessary. It was used only to introduce this parameter and indicate that other options exist beyond daily standing reservations.

Example 7-11: Host Constrained Standing Reservation

Although the first example did specify a quantity of resources to reserve, it did not specify where the needed tasks were to be located. If this information is not specified, Moab attempts to locate the needed resources anywhere it can find them. The Example 1 reservation essentially discovers hosts where the needed resources can be found. If the **SPACEFLEX** reservation flag is set, then the reservation continues to float to the best hosts over the life of the reservation. Otherwise, it will be locked to the initial set of allocated hosts.

If a site wanted to constrain a reservation to a subset of available resources, this could be accomplished using the **HOSTLIST** attribute. The **HOSTLIST** attribute is specified as a comma-separated list of host names 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 host list.*

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 host list provides more resources than what is required by the reservation as specified via **TASKCOUNT**, the scheduler will simply select the needed resources within the set of hosts listed.

Enforcing Policies Via Multiple Reservations

Single reservations enable multiple capabilities. Combinations of reservations can further extend a site's capabilities to impose specific policies.

Example 7-12: Reservation Stacking

If **HOSTLIST** is specified but **TASKCOUNT** is not, the scheduler will pack as many tasks as possible onto all of the listed hosts. For example, assume the site added a second standing reservation named *debug* to its configuration that reserved resources for use by certain members of its staff using the following configuration:

```
SRCFG[interactive] DAYS=MON,TUE,WED,THU,FRI
SRCFG[interactive] PERIOD=DAY
SRCFG[interactive] STARTTIME=10:00:00 ENDTIME=15:00:00
SRCFG[interactive] RESOURCES=PROCS:2,MEM:256
SRCFG[interactive] HOSTLIST=node001,node002,node005,node020
SRCFG[interactive] TASKCOUNT=6
SRCFG[interactive] CLASSLIST=interactive
SRCFG[debug]       HOSTLIST=node001,node002,node003,node004
SRCFG[debug]       USERLIST=helpdesk
SRCFG[debug]       GROUPLIST=operations,sysadmin
SRCFG[debug]       PERIOD=INFINITY
```

The new standing reservation is quite simple. Since **RESOURCES** is not specified, it will allocate all processors on each host that is allocated. Since **TASKCOUNT** is not specified, it will allocate every host listed in **HOSTLIST**. Since **PERIOD** is set to *INFINITY*, the reservation is always in force and there is no need to specify **STARTTIME**, **ENDTIME**, or **DAYS**.

The standing reservation has two access parameters set using the attributes USERLIST and **GROUPLIST**. This configuration indicates that the reservation can be accessed if any one of the access lists specified is satisfied by the job. In essence, reservation access is logically ORed 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 7-13: Multiple ACL Types

In most cases, access lists within a reservation are logically ORed 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 ANDed or ORed to the cumulative set of all other access lists as specified by the required ACL marker. Consider the following example configuration:

```
SRCFG[special] TASKCOUNT=32
SRCFG[special] PERIOD=WEEK
SRCFG[special] STARTTIME=1:08:00:00
SRCFG[special] ENDTIME=5:17:00:00
SRCFG[special] NODEFEATURES=largememory
SRCFG[special] TIMELIMIT=1:00:00*
SRCFG[special] QOSLIST=high,low,special-
SRCFG[special] ACCCOUNTLIST=!projectX,!projectY
```

The above configuration requests 32 tasks which translate to 32 nodes. The **PERIOD** attribute makes this reservation periodic on a weekly basis while the attributes **STARTTIME** and **ENDTIME** specify the week offsets when this reservation is to start and end. (Note that the specification format has changed to DD:HH:MM:SS.) In this case, the reservation starts on Monday at 8:00 a.m. and runs until Friday at 5:00 p.m. The reservation is enforced as a series of weekly reservations that only cover the specified time frame. The **NODEFEATURES** attribute indicates that each of the reserved nodes must have the node feature "largememory" configured.

As described earlier, **TIMELIMIT** indicates that jobs using this reservation can only use it for one hour. This means the job and the reservation can only overlap for one hour. Clearly jobs requiring an hour or less of wallclock time meet this constraint. However, a four-hour job that starts on Monday at 5:00 a.m. or a 12-hour job that starts on Friday at 4:00 p.m. also satisfies this constraint. Also, note the **TIMELIMIT** required ACL marker, *; it is set indicating that jobs must not only meet the **TIMELIMIT** access constraint but must also meet one or more of the other access constraints. In this example, the job can use this reservation if it can use the access specified via **QOSLIST** or **ACCOUNTLIST**; that is, it is assigned a QoS of *high*, *low*, or *special* , or the submitter of the job has an account that satisfies the *!projectX* and *!projectY* criteria. See the QoS Overview for more info about QoS configuration and usage.

Affinity

Reservation ACLs allow or deny access to reserved resources but they may be configured to also impact a job's affinity for a particular reservation. By default, jobs gravitate toward reservations through a mechanism known as positive affinity. This mechanism allows jobs to run on the most constrained resources leaving other, unreserved resources free for use by other jobs that may not be able to access the reserved resources. Normally this is a desired behavior. However, sometimes, it is desirable to reserve resources for use only as a last resort-using the reserved resources only when there are no other resources available. This last resort behavior is known as negative affinity. Note the '-' (hyphen or negative sign) following the *special* in the **QOSLIST** values. This special mark indicates that QoS *special* should be granted access to this reservation but should be assigned negative affinity. Thus, the **QOSLIST** attribute specifies that QoS *high* and *low* should be granted access with positive affinity (use the reservation first where possible) and QoS *special* granted access with negative affinity (use the reservation only when no other resources are available).

Affinity status is granted on a per access object basis rather than a per access list basis and always defaults to positive affinity. In addition to negative affinity, neutral affinity can also be specified using the equal sign (=) as in `QOSLIST[0] normal= high debug= low-`.

When a job matches multiple ACLs for a reservation, the final node affinity for the node, job, and reservation combination is based on the last matching ACL entry found in the configuration file.

For example, given the following reservation ACLs, a job matching both will receive a negative affinity:

```
SRCFG[res1] USERLIST=joe+ MAXTIME<=4:00:00-
```

With the following reservation ACLs, a job matching both will receive a positive affinity:

```
SRCFG[res1] MAXTIME<=4:00:00- USERLIST=joe+
```

ACL Modifiers

ACL modifiers allow a site to change the default behavior of ACL processing. By default, a reservation can be accessed if one or more of its ACLs can be met by the requestor. This behavior can be changed using the "deny" or "required" ACL modifier, as in the following tables:

Not	
Symbol:	! (exclamation point)
Description	If attribute is met, the requestor is denied access regardless of any other satisfied ACLs.
Example	`SRCFG[test] GROUPLIST=staff USERLIST=!steve` *Allow access to all staff members other than steve.*

Required	
Symbol:	* (asterisk)
Description	All required ACLs must be satisfied for requestor access to be granted.
Example	`SRCFG[test] QOSLIST=*high MAXTIME=*2:00:00` *Only jobs in QoS high that request less than 2 hours of walltime are granted access.*

XOR	
Symbol:	^ (carat)
Description	All attributes of the type specified other than the ones listed in the ACL satisfy the ACL.
Example	`SRCFG[test] QOSLIST=^high` *All jobs other than those requesting QoS high are granted access.*

CredLock	
Symbol:	& (ampersand)
Description	Matching jobs will be required to run on the resources reserved by this reservation. You can use this modifier on accounts, classes, groups, qualities of service, and users.
Example	`SRCFG[test] USERLIST=&john` *All of user john's jobs must run in this reservation.*

HPEnable (hard policy enable)	
Symbol:	~ (tilde)
Description	ACLs marked with this modifier are ignored during soft policy scheduling and are only considered for hard policy scheduling once all eligible soft policy jobs start.
Example	`SRCFG[johnspace] USERLIST=john CLASSLIST=~debug` *All of user john's jobs are allowed to run in the reservation at any time. Debug jobs are also allowed to run in this reservation but are only considered after all of John's jobs are given an opportunity to start. User john's jobs are considered before debug jobs regardless of job priority.* If HPEnable and **Not** markers are used in conjunction, then specified credentials are *blocked-out* of the reservation during soft-policy scheduling.

Note the **ACCOUNTLIST** values in Example 7-13 are preceded with an exclamation point, or NOT symbol. This indicates that all jobs with accounts other than *projectX* and *projectY* meet the account ACL. Note that if a !<X> value (!projectX) appears in an ACL line, that ACL is satisfied by any object not explicitly listed by a NOT entry. Also, if an object matches a NOT entry, the associated job is excluded from the reservation even if it meets other ACL requirements. For example, a QoS 3 job requesting account *projectX* is denied access to the reservation even though the job QoS matches the QoS ACL.

Example 7-14: Binding Users to Reservations at Reservation Creation

```
# create a 4 node reservation for john and bind all of john's jobs to that reservation
> mrsvctl -c -a user=&john -t 4
```

Reservation Ownership

Reservation ownership allows a site to control who owns the reserved resources during the reservation time frame. Depending on needs, this ownership may be identical to, a subset of, or completely distinct

from the reservation ACL. By default, reservation ownership implies resource accountability and resources not consumed by jobs are accounted against the reservation owner. In addition, ownership can also be associated with special privileges within the reservation.

Ownership is specified using the **OWNER** attribute in the format <CREDTYPE>:<CREDID>, as in **OWNER**=*USER:john*. To enable *john*'s jobs to preempt other jobs using resources within the reservation, the **SRCFG** attribute **FLAG** should be set to OWNERPREEMPT. In the example below, the *jupiter* project chooses to share resources with the *saturn* project but only when it does not currently need them.

Example 7-15: Limited Shared Access

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

Partitions

A reservation can be used in conjunction with a partition. Configuring a standing reservation on a partition allows constraints to be (indirectly) applied to a partition.

Example 7-16: Time Constraints by Partition

The following example places a 3-day wall-clock limit on two partitions and a 64 processor-hour limit on jobs running on partition *small*.

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

Resource Allocation Behavior

As mentioned, standing reservations can operate in one of two modes, floating, or non-floating (essentially node-locked). A floating reservation is created when the flag **SPACEFLEX** is specified. If a reservation is non-floating, the scheduler allocates all resources specified by the **HOSTLIST** parameter regardless of node state, job load, or even the presence of other standing reservations. Moab interprets the request for a non-floating reservation as, "I want a reservation on these exact nodes, no matter what!"

If a reservation is configured to be floating, the scheduler takes a more relaxed stand, searching through all possible nodes to find resources meeting standing reservation constraints. Only Idle, Running, or Busy nodes are considered and further, only considered if no reservation conflict is detected. The reservation attribute **ACCESS** modifies this behavior slightly and allows the reservation to allocate resources even if reservation conflicts exist.

If a **TASKCOUNT** is specified with or without a **HOSTEXPRESSION**, Moab will, by default, only consider "up" nodes for allocation. To change this behavior, the reservation flag IGNSTATE can be specified as in the following example:

```
SRCFG[nettest] GROUPLIST=sysadm
SRCFG[nettest] FLAGS=IGNSTATE
SRCFG[nettest] HOSTLIST=node1[3-8]
SRCFG[nettest] STARTTIME=9:00:00
SRCFG[nettest] ENDTIME=17:00:00
```

Access to existing reservations can be controlled using the reservation flag IGNRSV.

Other standing reservation attributes not covered here include **PARTITION** and **CHARGEACCOUNT**. These parameters are described in some detail in the parameters documentation.

Example 7-17: Using Reservations to Guarantee Turnover

In some cases, it is desirable to make certain a portion of a cluster's resources are available within a specific time frame. The following example creates a floating reservation belonging to the *jupiter* account that guarantees 16 tasks for use by jobs requesting up to one hour.

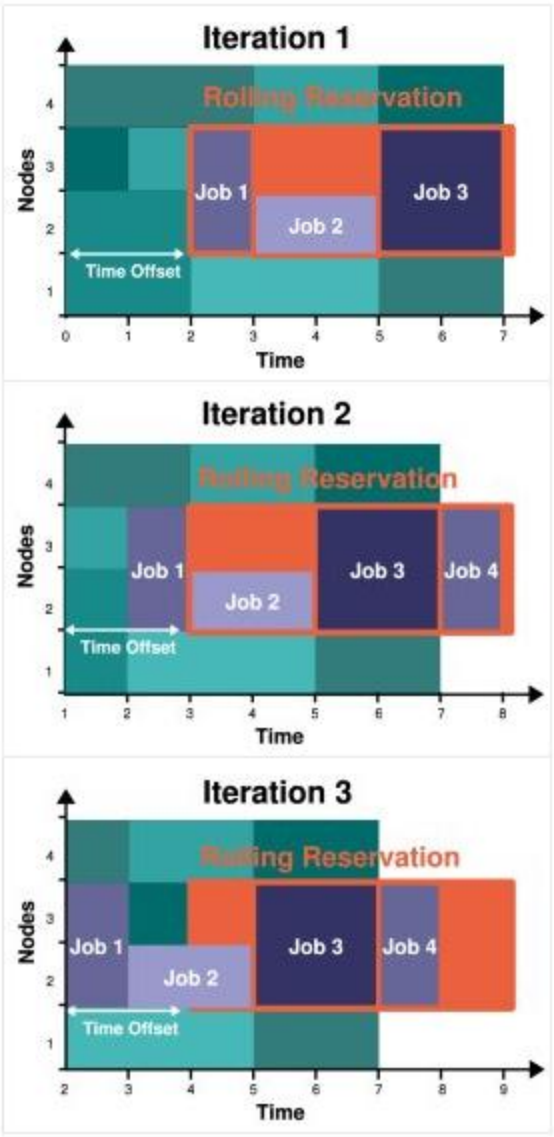
```
SRCFG[shortpool] OWNER=ACCT:jupiter
SRCFG[shortpool] FLAGS=SPACEFLEX
SRCFG[shortpool] MAXTIME=1:00:00
SRCFG[shortpool] TASKCOUNT=16
SRCFG[shortpool] STARTTIME=9:00:00
SRCFG[shortpool] ENDTIME=17:00:00
SRCFG[shortpool] DAYS=Mon,Tue,Wed,Thu,Fri
```

This reservation enables a capability similar to what was known in early Maui releases as "shortpool." The reservation covers every weekday from 9:00 a.m. to 5:00 p.m., reserving 16 tasks and allowing jobs to overlap the reservation for up to one hour. The *SPACEFLEX* flag indicates that the reservation may be dynamically modified--over time to re-locate to more optimal resources. In the case of a reservation with the **MAXTIME** ACL, this would include migrating to resources that are in use but that free up within the **MAXTIME** time frame. Additionally, because the **MAXTIME** ACL defaults to positive affinity, any jobs that fit the ACL attempt to use available reserved resources first before looking elsewhere.

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.

Image 7-2: Rolling reservation over 3 iterations



Example 7-18: Rollback Reservations

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

Adding an asterisk to the **ROLLBACKOFFSET** value pins rollback reservation start times when an idle reservation is created in the rollback reservation. For example: `SRCFG[staff] ROLLBACKOFFSET=18:00:00* PERIOD=INFINITY`.

Modifying Resources with Standing Reservations

Moab can customize compute resources associated with a reservation during the life of the reservation. This can be done generally using the TRIGGER attribute, or it can be done for operating systems using

the shortcut attribute OS. If set, Moab dynamically reprovisions allocated reservation nodes to the requested operating system as shown in the following example:

```
SRCFG[provision] PERIOD=DAY DAY=MON,WED,FRI STARTTIME=7:00:00 ENDTIME=10:00:00
SRCFG[provision] OS=rhel4  # provision nodes to use redhat during reservation, restore
when done
```

Managing Administrative Reservations

A default reservation with no ACL is termed an *administrative* reservation, but is occasionally referred to as a *system* reservation. It blocks access to all jobs because it possesses an empty access control list. It is often useful when performing administrative tasks but cannot be used for enforcing resource usage policies.

Administrative reservations are created and managed using the mrsvctl command. With this command, all aspects of reservation time frame, resource selection, and access control can be dynamically modified. The mdiag -r command can be used to view configuration, state, allocated resource information as well as identify any potential problems with the reservation. The following table briefly summarizes commands used for common actions. More detailed information is available in the command summaries.

Action	Command
create reservation	mrsvctl -c <RSV_DESCRIPTION>
list reservations	mrsvctl -l
release reservation	mrsvctl -r <RSVID>
modify reservation	mrsvctl -m <ATTR>=<VAL> <RSVID>
query reservation configuration	mdiag -r <RSVID>
display reservation hostlist	mrsvctl -q resources <RSVID>

Related topics

- SRCFG (configure standing reservations)
- RSVPROFILE (create reservation profiles)

7.1.6 Personal Reservations

- Enabling Personal Reservation Management
- Reservation Accountability and Defaults
  - Reservation Allocation and Charging
  - Setting Reservation Default Attributes
- Reservation Limits
- Reservation and Job Binding
  - Constraining a job to only run in a particular reservation
  - Constraining a Reservation to Only Accept Certain Jobs

By default, advance reservations are only available to scheduler administrators. While administrators may create and manage reservations to provide resource access to end-users, end-users cannot create, modify, or destroy these reservations. Moab extends the ability to manage reservations to end-users and provides control facilities to keep these features manageable. Reservations created by end-users are called personal reservations or user reservations.

Enabling Personal Reservation Management

User, or personal reservations can be enabled on a per QoS basis by setting the ENABLEUSERRSV flag as in the following example:

```
QOSCFG[titan]    QFLAGS=ENABLEUSERRSV # allow 'titan' QoS jobs to create user
reservations
USERCFG[DEFAULT] QDEF=titan           # allow all users to access 'titan' QoS
...
```

If set, end-users are allowed to create, modify, cancel, and query reservations they own. As with jobs, users may associate a personal reservation with any QoS or account to which they have access. This is accomplished by specifying per reservation accountable credentials as in the following example:

```
> mrsvctl -c -S AQOS=titan -h node01 -d 1:00:00 -s 1:30:00
Note:  reservation test.126 created
```

As in the preceding example, a non-administrator user who wants to create a reservation must *ALWAYS* specify an accountable QoS with the mrsvctl -S flag. This specified QoS must have the **ENABLEUSERRSV** flag. By default, a personal reservation is created with an ACL of only the user who created it.

Example 7-19: Allow All Users in Engineering Group to Create Personal Reservations

```
QOSCFG[rsv]     QFLAGS=ENABLEUSERRSV # allow 'rsv' QoS jobs to create user
reservations
GROUPCFG[sales] QDEF=rsv             # allow all users in group sales to access 'rsv'
QoS
...
```

Example 7-20: Allow Specific Users to Create Personal Reservations

```
# special QoS has higher job priority and ability to create user reservations
QOSCFG[special] QFLAGS=ENABLEUSERRSV
QOSCFG[special] PRIORITY=1000
# allow betty and steve to use the special QoS
USERCFG[betty]  QDEF=special
USERCFG[steve]  QLIST=fast,special,basic QDEF=rsv
...
```

Reservation Accountability

Personal reservations must be configured with a set of accountable credentials. These credentials (user, group, account, and so forth) indicate who is responsible for the resources dedicated by the reservation. If resources are dedicated by a reservation but not consumed by a job, these resources can be charged against the specified accountable credentials. Administrators are allowed to create reservations and specify any accountable credentials for that reservation. While end-users can also be allowed to create and otherwise modify personal reservations, they are only allowed to create reservations with accountable credentials to which they have access. Further, while administrators may manage any reservation, end-users may only control reservations they own.

Like jobs, reservation accountable credentials specify which credentials are charged for reservation usage and what policies are enforced as far as usage limits and allocation management is concerned. (See the mrsvctl command documentation for more information on setting personal reservation credentials.) While similar to jobs, personal reservations do have a separate set of usage limits and different allocation charging policies.

Setting Reservation Default Attributes

Organizations can use reservation profiles to set default attributes for personal reservations. These attributes can include reservation aspects such as management policies, charging credentials, ACLs, host constraints, and time frame settings.

Reservation Limits

Allowing end-users the ability to create advance reservations can lead to potentially unfair and unproductive resource usage. This results from the fact that by default, there is nothing to prevent a user from reserving all resources in a given system or reserving resources during time slots that would greatly impede the scheduler's ability to schedule jobs efficiently. Because of this, it is highly advised that sites initially place either usage or allocation based constraints on the use of personal reservations. This can be achieved using Moab Accounting Manager (see the Moab Accounting Manager Administrator Guide).

Reservation and Job Binding

Moab allows job-to-reservation binding to be configured at an administrator or end-user level. This binding constrains how job to reservation mapping is allowed.

Constraining a job to only run in a particular reservation

Jobs may be bound to a particular reservation at submit time (using the RM extension ADVRES) or dynamically using the mjobctl command. (See Job to Reservation Mapping.) In either case, once bound to

a reservation, a job may only run in that reservation even if other resources may be found outside of that reservation. The mjobctl command may also be used to dynamically release a job from reservation binding.

Example 7-21: Bind job to reservation

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

Example 7-22: Release job from reservation binding

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

Constraining a Reservation to Only Accept Certain Jobs

Binding a job to a reservation is independent of binding a reservation to a job. For example, a reservation may be created for user "steve." User "steve" may then submit a number of jobs including one that is bound to that reservation using the **ADVRES** attribute. However, this binding simply forces that one job to use the reservation, it does not prevent the reservation from accepting other jobs submitted by user "steve." To prevent these other jobs from using the reserved resources, reservation to job binding must occur. This binding is accomplished by specifying either general job binding or specific job binding.

General job binding is the most flexible form of binding. Using the BYNAME attribute, a reservation may be created that only accepts jobs specifically bound to it.

Specific job binding is more constraining. This form of binding causes the reservation to only accept specific jobs, regardless of other job attributes and is set using the **JOB** reservation ACL.

Example 7-23: Configure a reservation to accept only jobs that are bound to it

```
> mrsvctl -m flags+=byname grid.3
```

Example 7-24: Remove general reservation to job binding

```
> mrsvctl -m flags-=byname grid.3
```

Example 7-25: Configure a reservation to accept a specific job

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

Example 7-26: Remove a specific reservation to job binding

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

7.2 Partitions

- Partition Overview
- Defining Partitions
- Managing Partition Access

- Requesting Partitions
- Per-Partition Settings
- Miscellaneous Partition Issues

Partition Overview

Partitions are a logical construct that divide available resources. Any single resource (compute node) may only belong to a single partition. Often, natural hardware or resource manager bounds delimit partitions such as in the case of disjoint networks and diverse processor configurations within a cluster. For example, a cluster may consist of 256 nodes containing four 64 port switches. This cluster may receive excellent interprocess communication speeds for parallel job tasks located within the same switch but sub-stellar performance for tasks that span switches. To handle this, the site may choose to create four partitions, allowing jobs to run within any of the four partitions but not span them.

While partitions do have value, it is important to note that within Moab, the standing reservation facility provides significantly improved flexibility and should be used in the vast majority of politically motivated cases where partitions may be required under other resource management systems. Standing reservations provide time flexibility, improved access control features, and more extended resource specification options. Also, another Moab facility called Node Sets allows intelligent aggregation of resources to improve per job node allocation decisions. In cases where system partitioning is considered for such reasons, node sets may be able to provide a better solution.

Still, one key advantage of partitions over standing reservations and node sets is the ability to specify partition specific policies, limits, priorities, and scheduling algorithms although this feature is rarely required. An example of this need may be a cluster consisting of 48 nodes owned by the Astronomy Department and 16 nodes owned by the Mathematics Department. Each department may be willing to allow sharing of resources but wants to specify how their partition will be used. As mentioned, many of Moab's scheduling policies may be specified on a per partition basis allowing each department to control the scheduling goals within their partition.

The partition associated with each node should be specified as indicated in the Node Location section. With this done, partition access lists may be specified on a per job or per QoS basis to constrain which resources a job may have access to. (See the QoS Overview for more information.) By default, QoSes and jobs allow global partition access. Note that by default, a job may only use resources within a single partition.

If no partition is specified, Moab creates one partition per resource manager into which all resources corresponding to that resource manager are placed. (This partition is given the same name as the resource manager.)

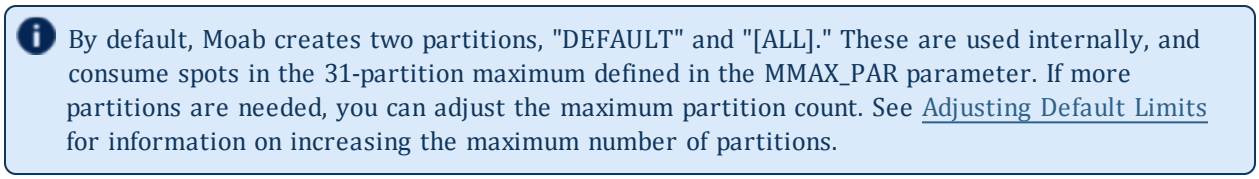
A partition may not span multiple resource managers. In addition to these resource manager partitions, a pseudo-partition named " [ALL]" is created that contains the aggregate resources of all partitions.

While the resource manager partitions are real partitions containing resources not explicitly assigned to other partitions, the " [ALL]" partition is only a convenience object and is not a real partition; thus it cannot be requested by jobs or included in configuration ACLs.

Defining Partitions

Node to partition mappings can be established directly using the NODECFG parameter or indirectly using the FEATUREPARTITIONHEADER parameter. If using direct mapping, this is accomplished as shown in the example that follows.

```
NODECFG[node001]   PARTITION=astronomy
NODECFG[node002]   PARTITION=astronomy
...
NODECFG[node049]   PARTITION=math
...
```



Managing Partition Access

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

Credential Based Access

Determining who can use which partition is specified using the *CFG parameters (USERCFG, GROUPCFG, ACCOUNTCFG, QOSCFG, CLASSCFG, and SYSCFG). These parameters allow you to select a partition access list on a credential or system wide basis using the **PLIST** attribute. By default, the access associated with any given job is the logical OR of all partition access lists assigned to the job's credentials.

For example, assume a site with two partitions, *general*, and *test*. The site management would like everybody to use the *general* partition by default. However, one user, Steve, needs to perform the majority of his work on the test partition. Two special groups, staff and management will also need access to use the test partition from time to time but will perform most of their work in the general partition. The following example configuration enables the needed user and group access and defaults for this site:

```
SYSCFG[base]     PLIST=general:test
USERCFG[DEFAULT] PLIST=general
USERCFG[steve]   PLIST=general:test
GROUPCFG[staff]  PLIST=general:test
GROUPCFG[mgmt]   PLIST=general:test
```

While using a logical OR approach allows sites to add access to certain jobs, some sites prefer to work the other way around. In these cases, access is granted by default and certain credentials are then restricted from accessing various partitions. To use this model, a system partition list must be specified as in the following example:

```
SYSCFG[base]    PLIST=general,test&
USERCFG[demo]   PLIST=test&
GROUPCFG[staff] PLIST=general&
```

In the preceding example, note the ampersand (&). This character, which can be located anywhere in the **PLIST** line, indicates that the specified partition list should be logically 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.

Per Job Resource Limits

Access to partitions can be constrained based on the resources requested on a per job basis with limits on both minimum and maximum resources requested. All limits are specified using PARCFG. See Usage Limits for more information on the available limits.

```
PARCFG[amd]     MAX.PROC=16
PARCFG[pIII]    MAX.WCLIMIT=12:00:00 MIN.PROC=4
PARCFG[aix]     MIN.NODE=12
```

Requesting Partitions

Users may request to use any partition they have access to on a per job basis. This is accomplished using the resource manager extensions since most native batch systems do not support the partition concept. For example, on a TORQUE system, a job submitted by a member of the group *staff* could request that the job run in the *test* partition by adding the line `-l partition=test` to the qsub command line. See the resource manager extension overview for more information on configuring and using resource manager extensions.

Per-Partition Settings

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

Setting	Description
DEFAULTNODEFEATURES	Specifies a default feature on a group of node within a partition and applies only to nodes in that partition.
JOBNODEMATCHPOLICY	Specifies the JOBNODEMATCHPOLICY to be applied to jobs that run in the specified partition.
NODEACCESSPOLICY	Specifies the NODEACCESSPOLICY to be applied to jobs that run in the specified partition.
NODEALLOCATIONPOLICY	Specifies the NODEALLOCATIONPOLICY to be applied to jobs that run in the specified partition.
USETTC	Specifies whether TTC specified at submission should be used and displayed by the scheduler.

Setting	Description
VMCREATEDURATION	Specifies the maximum amount of time VM creation can take before Moab considers it a failure (in [HH[:MM[:SS]). If no value is set, there is no maximum limit.
VMDELETEDURATION	Specifies the maximum amount of time VM deletion can take before Moab considers it a failure (in [HH[:MM[:SS]). If no value is set, there is no maximum limit.
VMMIGRATEDURATION	Specifies the maximum amount of time VM migration can take before Moab considers it a failure (in [HH[:MM[:SS]). If no value is set, there is no maximum limit.

Miscellaneous Partition Issues

A brief caution: Use of partitions has been quite limited in recent years as other, more effective approaches are selected for site scheduling policies. Consequently, some aspects of partitions have received only minor testing. Still, note that partitions are fully supported and any problem found will be rectified.

Related topics

- Standing Reservations
- Node Sets
- FEATUREPARTITIONHEADER parameter
- PARCFG parameter

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 QoSes available to each user and job.
- Enable special charging rates based on requested or delivered QoS levels.
- Enable limits on the extent of use for each defined QoS.
- Monitor current and historical usage for each defined QoS.

It contains the following sub-sections:

- QoS Overview
- QoS Enabled Privileges
    - Special Prioritization
    - Service Access and Constraints
    - Usage Limits and Overrides
    - Service Access Thresholds
    - Preemption Management
- Managing QoS Access
- Requesting QoS Services at Job Submission
- Restricting Access to Special Attributes

QoS Overview

Moab's QoS facility allows a site to give special treatment to various classes of jobs, users, groups, and so forth. Each QoS object can be thought of as a container of special privileges ranging from fairness policy exemptions, to special job prioritization, to special functionality access. Each QoS object also has an extensive access list of users, groups, and accounts that can access these privileges.

Sites can configure various QoSes 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 QoSes. A given job will have a default QoS and may have access to several additional QoSes. 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 MEMBERULIST attribute.

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

QoS Enabled Privileges

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

- **Special Prioritization** on page 425
- **Service Access and Constraints** on page 425
- **Usage Limits and Overrides** on page 428
- **Service Access Thresholds** on page 428
- **Preemption Management** on page 429

All privileges are managed via the QOSCFG parameter.

Special Prioritization

Attribute name	Description
FSTARGET	Specifies QoS fairshare target.
FSWEIGHT	Sets QoS fairshare weight offset affecting a job's fairshare priority component.
PRIORITY	Assigns priority to all jobs requesting particular QoS.
QTTARGET	Sets QoS queuetime target affecting a job's target priority component and QoS delivered.
QTWEIGHT	Sets QoS queuetime weight offset affecting a job's service priority component.
XFTARGET	Sets QoS XFactor target affecting a job's target priority component and QoS delivered.
XFWEIGHT	Sets QoS XFactor weight offset affecting a job's service priority component.

Example 7-27:

```
# assign priority for all QoS geo jobs

QOSCFG[geo]  PRIORITY=10000
```

Service Access and Constraints

The QoS facility can be used to enable special services and to disable default services. These services are enabled/disabled by setting the QoS **QFLAGS** attribute.

Flag Name	Description
DEADLINE	Job may request an absolute or relative completion deadline and Moab will reserve resources to meet that deadline. (An alternative priority based deadline behavior is discussed in the PRIORITY FACTORS section.)
DEDICATED	Moab dedicates all resources of an allocated node to the job meaning that the job will not share a node's compute resources with any other job.
ENABLEUSERRSV	Allow user or personal reservations to be created and managed.
IGNALL	Scheduler ignores all resource usage policies for jobs associated with this QoS.

Flag Name	Description
JOBPRIOACCRUALPOLICY	Specifies how Moab should track the dynamic aspects of a job's priority. The two valid values are *ACCRUE* and *RESET*. • *ACCRUE* indicates that the job will accrue queuetime based priority from the time it is submitted unless it violates any of the policies not specified in JOBPRIOEXCEPTIONS. • *RESET* indicates that it will accrue priority from the time it is submitted unless it violates any of the JOBPRIOEXCEPTIONS. However, with *RESET*, if the job does violate JOBPRIOEXCEPTIONS then its queuetime based priority will be reset to 0. **JOBPRIOACCRUALPOLICY** is a global parameter, but can be configured to work only in QOSCFG: `QOSCFG[arrays]  JOBPRIOACCRUALPOLICY=ACCRUE` The following old **JOBPRIOACCRUALPOLICY** values have been deprecated and should be adjusted to the following values: • QUEUEPOLICY = ACCRUE and JOBPRIOEXCEPTIONS SOFTPOLICY,HARDPOLICY • QUEUEPOLICYRESET = RESET and JOBPRIOEXCEPTIONS SOFTPOLICY,HARDPOLICY • ALWAYS = ACCRUE and JOBPRIOEXCEPTIONS ALL • FULLPOLICY = ACCRUE and JOBPRIOEXCEPTIONS NONE • FULLPOLICYRESET = RESET and JOBPRIOEXCEPTIONS NONE
JOBPRIOEXCEPTIONS	Specifies exceptions for calculating a job's dynamic priority (QUEUETIME, XFACTOR, TARGETQUEUETIME). Valid values are a comma delimited list of any of the following: *DEFER*, *DEPENDS*, *SOFTPOLICY*, *HARDPOLICY*, *IDLEPOLICY*, *USERHOLD*, *BATCHHOLD*, and *SYSTEMHOLD* (*ALL* or *NONE* can also be specified on their own). Normally, when a job violates a policy, is placed on hold, or has an unsatisfied dependency, it will not accrue priority. Exceptions can be configured to allow a job to accrue priority in spite of any of these violations. With *DEPENDS* a job will increase in priority even if there exists an unsatisfied dependency. With *SOFTPOLICY*, *HARDPOLICY*, or *IDLEPOLICY* a job can accrue priority despite violating a specific limit. With *DEFER*, *USERHOLD*, *BATCHHOLD*, or *SYSTEMHOLD* a job can accrue priority despite being on hold. **JOBPRIOEXCEPTIONS** is a global parameter, but can be configured to work only in QOSCFG: `QOSCFG[arrays]  JOBPRIOEXCEPTIONS=IDLEPOLICY`
NOBF	Job is not considered for backfill.

Flag Name	Description
NORESERVATION	Job should never reserve resources regardless of priority.
NTR	Job is prioritized as next to run (NTR) and backfill is disabled to prevent other jobs from jumping in front of ones with the NTR flag. It is important to note that jobs marked with this flag should not be blocked. If they are, Moab will stop scheduling because if a job is marked with this flag, no other jobs will be run until the flagged NTR (Next to Run) job starts. Consider using the **PRIORITY** attribute of the **QOSCFG[<QOSID>]** on page 894 parameter instead, when possible. Or, as you may encounter a scheduling delay for NTR-flagged jobs to start, consider using the **RESERVATIONDEPTH[X]** and **RESERVATIONQOSLIST[X]** parameters to provide better scheduling flow. See **Reservation Policies** on page 382 (especially the section on Assigning Per-QoS Reservation Creation Rules) for more information.
PREEMPTCONFIG	User jobs may specify options to alter how preemption impacts the job such as min-preempttime.
PREEMPTEE	Job may be preempted by higher priority **PREEMPTOR** jobs.
PREEMPTFSV	Job may be preempted by higher priority **PREEMPTOR** jobs if it exceeds its fairshare target when started.
PREEMPTOR	Job may preempt lower priority **PREEMPTEE** jobs.
PREEMPTSPV	Job may be preempted by higher priority **PREEMPTOR** jobs if it currently violates a soft usage policy limit.
PROVISION	If the job cannot locate available resources with the needed OS or software, the scheduler may provision a number of nodes to meet the needed OS or software requirements.
RESERVEALWAYS	Job should create resource reservation regardless of job priority.
RUNNOW	Boosts a job's system priority and makes the job a preemptor. RUNNOW overrides resource restrictions such as MAXJOB or MAXPROC.
TRIGGER	The job is able to directly specify triggers.
USERESERVED[:<RSVID>]	Job may only use resources within accessible reservations. If *<RSVID>* is specified, job may only use resources within the specified reservation.

Example 7-28: For lowprio QoS job, disable backfill and make job preemptible

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

Example 7-29: Bind all jobs to chemistry reservation

```
QOSCFG[chem-b]  QFLAGS=USERESERVED:chemistry
```

Other QoS Attributes

In addition to the flags, there are attributes that alter service access.

Attribute name	Description
SYSPRIO	Sets the system priority on jobs associated with this QoS.

Example 7-30: All jobs submitted under a QoS sample receive a system priority of 1

```
QOSCFG[sample]  SYSPRIO=1
```

Per QoS Required Reservations

If desired, jobs associated with a particular QoS can be locked into a reservation or reservation group using the **REQRID** attribute. For example, to force jobs using QoS *jasper* to only use the resources within the *failsafe* standing reservation, use the following:

```
QOSCFG[jasper] REQRID=failsafe
...
```

Usage Limits and Overrides

All credentials, including QoS, allow specification of job usage limits as described in the Basic Fairness Policies overview. In such cases, jobs are constrained by the most limiting of all applicable policies. With QoSes, an override limit may also be specified and with this limit, jobs are constrained by the override, regardless of other limits specified. The following attributes can override the throttling policies from other credentials:

OMAXJOB, **OMAXNODE**, **OMAXPE**, **OMAXPROC**, **OMAXPS**, **OMAXJPROC**, **OMAXJPS**, **OMAXJWC**, and **OMAXJNODE**.

(See Usage Limits/Throttling Policies Override Limits.)

Example 7-31:

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

Service Access Thresholds

Jobs can be granted access to services such as preemption and reservation creation, and they can be granted access to resource reservations. However, with QoS thresholds, this access can be made

conditional on the current queuetime and XFactor metrics of an idle job. The following table lists the available QoS service thresholds:

Threshold attribute	Description
PREEMPTQTTHRESHOLD	A job with this QoS becomes a preemptor if the specified queuetime threshold is reached.
PREEMPTXFTHRESHOLD	A job with this QoS becomes a preemptor if the specified XFactor threshold is reached.
RSVQTTHRESHOLD	A job with this QoS can create a job reservation to guarantee resource access if the specified queuetime threshold is reached.
RSVXFTHRESHOLD	A job with this QoS can create a job reservation to guarantee resource access if the specified XFactor threshold is reached.
ACLQTTHRESHOLD	A job with this QoS can access reservations with a corresponding QoS ACL only if the specified queuetime threshold is reached.
ACLXFTHRESHOLD	A job with this QoS can access reservations with a corresponding QoS ACL only if the specified XFactor threshold is reached.
TRIGGERQTTHRESHOLD	If a job with this QoS fails to run before this threshold is reached, any failure triggers associated with this QoS will fire.

Preemption Management

Job preemption facilities can be controlled on a per-QoS basis using the PREEMPTEE and PREEMPTOR flags. Jobs that are preemptible can optionally be constrained to only be preempted in a particular manner by specifying the QoS **PREEMPTPOLICY** attribute as in the following example:

```
QOSCFG[special] QFLAGS=PREEMPTEE PREEMPTPOLICY=CHECKPOINT
```

For preemption to be effective, a job must be marked as a preemptee and must be enabled for the requested preemption type. For example, if the PREEMPTPOLICY is set to suspend, a potential target job must be both a preemptee and marked with the job flag **SUSPENDABLE**. (See suspension for more information.) If the target job is not suspendable, it will be either requeued or canceled. Likewise, if the **PREEMPTPOLICY** is set to *requeue*, the job will be requeued if it is marked restartable. Otherwise, it will be canceled.

The minimum time a job must run before being considered eligible for preemption can also be configured on a per-QoS basis using the **PREEMPTMINTIME** attribute, which is analogous to the JOBPREEMPTMINACTIVETIME. Conversely, **PREEMPTMAXTIME** sets a threshold for which a job is no longer eligible for preemption; see JOBPREEMPTMAXACTIVETIME for analogous details.

The **PREEMPTEES** attribute allows you to specify which QoSes that a job in a specific QoS is allowed to preempt. The **PREEMPTEES** list is a comma-delimited list of QoS IDs. When a **PREEMPTEES** attribute is

specified, a job using that QoS can only preempt jobs using QoSes listed in the **PREEMPTEES** list. In turn, those QoSes must be flagged as *PREEMPTEE* as in the following example:

```
QOSCFG[a] QFLAGS=PREEMPTOR PREEMPTEES=b,c
QOSCFG[b] QFLAGS=PREEMPTEE
QOSCFG[c] QFLAGS=PREEMPTEE
```

In the example, jobs in the a QoS can only preempt jobs in the b and c QoSes.

Managing QoS Access

Specifying Credential Based QoS Access

You can define the privileges allowed within a QoS by using the QOSCFG parameter; however, in most cases access to the QoS is enabled via credential specific *CFG parameters, specifically the USERCFG, GROUPCFG, ACCOUNTCFG, and CLASSCFG parameters, which allow defining QoS access lists and QoS defaults. Specify credential specific QoS access by using the **QLIST** and/or **QDEF** attributes of the associated credential parameter.

QoS Access via Logical OR

To enable QoS access, the **QLIST** and/or **QDEF** attributes of the appropriate user, group, account, or class/queue should be specified as in the following example:

```
# user john's jobs can access QoS geo, chem, or staff with geo as default
USERCFG[john]     QDEF=geo   QLIST=geo,chem,staff
# group system jobs can access the development QoS
GROUPCFG[systems] QDEF=development
# class batch jobs can access the normal QoS
CLASSCFG[batch]   QDEF=normal
```

By default, jobs may request a QoS if access to that QoS is allowed by any of the job's credentials. (In the previous example, a job from user *john* submitted to the class *batch* could request QoSes *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:

```
# 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 QoSes regardless of other
credentials
CLASSCFG[debug]   QLIST=development,lowpri&
```

Specifying QoS Based Access

QoS access may also be specified from within the QoS object using the QoS **MEMBERULIST** attribute as in the following example:

```
# define QoS premiere and grant access to users steve and john
QOSCFG[premiere]  PRIORITY=1000  QFLAGS=PREEMPTOR  MEMBERULIST=steve,john
```

By default, if a job requests a QoS that it cannot access, Moab places a hold on that job. The QOSREJECTPOLICY can be used to modify this behavior.

Requesting QoS Services at Job Submission

By default, jobs inherit a default QoS based on the user, group, class, and account associated with the job. If a job has access to multiple QoS levels, the submitter can explicitly request a particular QoS using the QoS resource manager extension as in the following example:

```
> msub -l nodes=1,walltime=100,qos=special3 job.cmd
```

Restricting Access to Special Attributes

By default, Moab allows all users access to special attributes such as node access policy. By enabling the QoS facility **SPECATTRS**, the access to these policies can be restricted. For example, to enable the facility, in the moab.cfg file, specify `QOSCFG[DEFAULT] SPECATTRS=`. Then, to allow access to the special attributes, indicate which special attributes a specific QoS may access.

```
QOSCFG[DEFAULT] SPECATTRS=
QOSCFG[high]  SPECATTRS=NACCESSPOLICY
```

Related topics

- Credential Overview
- Allocation Management Overview
- Rollback Reservations
- Job Deadlines

8.0 Optimizing Scheduling Behavior – Backfill and Node Sets

- **Optimization Overview** on page 433
- **Backfill** on page 434
- **Node Set Overview** on page 439

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:

Level	Description
Workload Ordering	Prioritizing workload and utilizing backfill
Intelligent Resource Allocation	Selecting those resources that best meet the job's needs or best enable future jobs to run (see node allocation)
Maximizing Intra-Job Efficiency	Selecting the type of nodes, collection of nodes, and proximity of nodes required to maximize job performance by minimizing both job compute and inter-process communication time (see node sets and node allocation)
Job Pree-mption	Preempting jobs to allow the most important jobs to receive the best response time (see pree-mption)

Level	Description
Utilizing Flexible Policies	Using policies that minimize blocking and resource fragmentation while enforcing needed constraints (see soft throttling policies and reservations)

8.2 Backfill

- Backfill Overview
- Backfill Algorithms
- Configuring Backfill

Backfill Overview

Backfill is a scheduling optimization that allows a scheduler to make better use of available resources by running jobs out of order. When Moab schedules, it prioritizes the jobs in the queue according to a number of factors and then orders the jobs into a highest priority first (or priority FIFO) sorted list. It starts the jobs one by one stepping through the priority list until it reaches a job it cannot start. Because all jobs and reservations possess a start time and a wallclock limit, Moab can determine the completion time of all jobs in the queue. Consequently, Moab can also determine the earliest the needed resources will become available for the highest priority job to start.

Backfill operates based on this earliest job start information. Because Moab knows the earliest the highest priority job can start, and which resources it will need at that time, it can also determine which jobs can be started without delaying this job. Enabling backfill allows the scheduler to start other, lower-priority jobs so long as they do not delay the highest priority job. If backfill is enabled, Moab protects the highest priority job's start time by creating a job reservation to reserve the needed resources at the appropriate time. Moab then can start any job that will not interfere with this reservation.

Backfill offers significant scheduler performance improvement. In a typical large system, enabling backfill increases system utilization by about 20% and improves turnaround time by an even greater amount. Because of the way it works, essentially filling in holes in node space, backfill tends to favor smaller and shorter running jobs more than larger and longer running ones. It is common to see over 90% of these small and short jobs backfilled. Consequently, sites will see marked improvement in the level of service delivered to the small, short jobs and moderate to little improvement for the larger, long ones.

With most algorithms and policies, there is a trade-off. Backfill is not an exception but the negative effects are minor. Because backfill locates jobs to run from throughout the idle job queue, it tends to diminish the influence of the job prioritization a site has chosen and thus may negate some desired workload steering attempts through this prioritization. Although by default the start time of the highest priority job is protected by a reservation, there is nothing to prevent the third priority job from starting early and possibly delaying the start of the second priority job. This issue is addressed along with its trade-offs later in this section.

Another problem is a little more subtle. Consider the following scenario involving a two-processor cluster. Job A has a four-hour wallclock limit and requires one processor. It started one hour ago (time zero) and will reach its wallclock limit in three more hours. Job B is the highest priority idle job and requires two processors for one hour. Job C is the next highest priority job and requires one processor for two hours. Moab examines the jobs and correctly determines that job A must finish in three hours and thus, the earliest job B can start is in three hours. Moab also determines that job C can start and finish in less than this amount of time. Consequently, Moab starts job C on the idle processor at time one. One hour later (time two), job A completes early. Apparently, the user overestimated the amount of time job A would need by a few hours. Since job B is now the highest priority job, it should be able to run. However, job C, a lower priority job was started an hour ago and the resources needed for job B are not available. Moab re-evaluates job B's reservation and determines that it can slide forward an hour. At time three, job B starts.

In review, backfill provided positive benefits. Job A successfully ran to completion. Job C was started immediately. Job B was able to start one hour sooner than its original target time, although, had backfill not been enabled, job B would have been able to run two hours earlier.

The scenario just described occurs quite frequently because user estimates for job duration are generally inaccurate. Job wallclock estimate accuracy, or wallclock accuracy, is defined as the ratio of wall time required to actually run the job divided by the wall time requested for the job. Wallclock accuracy varies from site to site but the site average is rarely better than 50%. Because the quality of the walltime estimate provided by the user is so low, job reservations for high priority jobs are often later than they need to be.

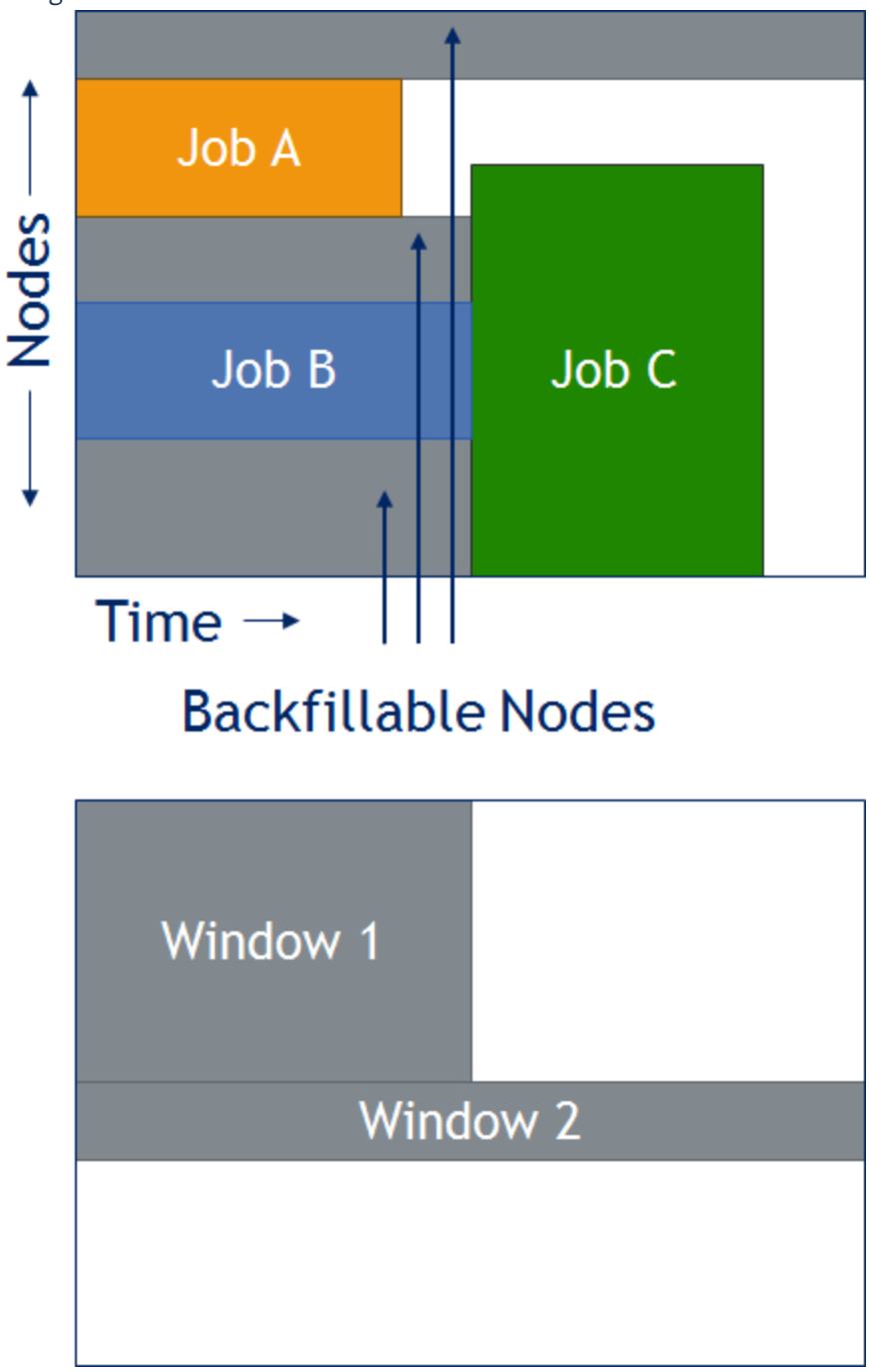
Although there do exist some minor drawbacks with backfill, its net performance impact on a site's workload is very positive. While a few of the highest priority jobs may get temporarily delayed, their position as highest priority was most likely accelerated by the fact that jobs in front of them were able to start earlier due to backfill. Studies have shown that only a very small number of jobs are truly delayed and when they are, it is only by a fraction of their total queue time. At the same time, many jobs are started significantly earlier than would have occurred without backfill.

Backfill Algorithms

The algorithm behind Moab backfill scheduling is straightforward, although there are a number of issues and parameters that should be highlighted. First of all, Moab makes two backfill scheduling passes. For each pass, Moab selects a list of jobs that are eligible for backfill. On the first pass, only those jobs that meet the constraints of the soft fairness throttling policies are considered and scheduled. The second pass expands this list of jobs to include those that meet the hard (less constrained) fairness throttling policies.

The second important concept regarding Moab backfill is the concept of backfill windows. The figure below shows a simple batch environment containing two running jobs and a reservation for a third job. The present time is represented by the leftmost end of the box with the future moving to the right. The light gray boxes represent currently idle nodes that are eligible for backfill. For this example, let's assume that the space represented covers 8 nodes and a 2 hour time frame. To determine backfill windows, Moab analyzes the idle nodes essentially looking for largest node-time rectangles. It determines that there are two backfill windows. The first window, Window 1, consists of 4 nodes that are available for only one hour (because some of the nodes are blocked by the reservation for Job 3). The second window contains only one node but has no time limit because this node is not blocked by the reservation for Job 3. It is important to note that these backfill windows overlap.

Image 8-1: Backfillable nodes create backfill windows 1 and 2



Once the backfill windows have been determined, Moab begins to traverse them. The current behavior is to traverse these windows widest window first (most nodes to fewest nodes). As each backfill window is evaluated, Moab applies the backfill algorithm specified by the BACKFILLPOLICY parameter.

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

1. The list of feasible backfill jobs is filtered, selecting only those that will actually fit in the current backfill window.
2. The first job is started.
3. While backfill jobs and idle resources remain, repeat step 1.

If *NONE* is set, the backfill policy is disabled.

Other backfill policies behave in a generally similar manner. The parameters documentation provides further details.

Liberal versus Conservative Backfill

By default, Moab reserves only the highest priority job resulting in a liberal and aggressive backfill. This reservation guarantees that backfilled jobs will not delay the highest priority job, although they may delay other jobs. The parameter RESERVATIONDEPTH controls how conservative or liberal the backfill policy is. This parameter controls how deep down the queue priority reservations will be made. While increasing this parameter guarantees that priority jobs will not be bypassed, it reduces the freedom of the scheduler to backfill resulting in somewhat lower system utilization. The significance of the trade-offs should be evaluated on a site by site basis.

Configuring Backfill

Backfill Policies

Backfill is enabled in Moab by specifying the BACKFILLPOLICY parameter. By default, backfill is enabled in Moab using the *FIRSTFIT* algorithm. However, this parameter can also be set to *NONE* (disabled).

The number of reservations that protect the resources required by priority jobs can be controlled using RESERVATIONDEPTH. This depth can be distributed across job QoS levels using RESERVATIONQOSLIST.

Backfill Chunking

In a batch environment saturated with serial jobs, serial jobs will, over time, dominate the resources available for backfill at the expense of other jobs. This is due to the time-dimension fragmentation associated with running serial jobs. For example, given an environment with an abundance of serial jobs, if a multi-processor job completes freeing processors, one of three things will happen:

1. The freed resources are allocated to another job requiring the same number of processors.
2. Additional jobs may complete at the same time allowing a larger job to allocate the aggregate resources.
3. The freed resources are allocated to one or more smaller jobs.

In environments where the scheduling iteration is much higher than the average time between completing jobs, case 3 occurs far more often than case 2, leading to smaller and smaller jobs populating the system over time.

To address this issue, the scheduler incorporates the concept of chunking. Chunking allows the scheduler to favor case 2 maintaining a more controlled balance between large and small jobs. The idea of chunking involves establishing a time-based threshold during which resources available for backfill are aggregated. This threshold is set using the parameter BFCHUNKDURATION. When resources are freed, they are made available only to jobs of a certain size (set using the parameter BFCHUNKSIZE) or larger.

These resources remain protected from smaller jobs until either additional resources are freed up and a larger job can use the aggregate resources, or until the **BFCHUNKDURATION** threshold time expires.

Backfill chunking is only activated when a job of size **BFCHUNKSIZE** or larger is blocked in backfill due to lack of resources.

It is important to note that the optimal settings for these parameters is very site-specific and will depend on the workload (including the average job turnaround time, job size, and mix of large to small jobs), cluster resources, and other scheduling environmental factors. Setting too restrictive values needlessly reduces utilization while settings that are too relaxed do not allowed the desired aggregation to occur.

Backfill chunking is only enabled in conjunction with the *FIRSTFIT* backfill policy.

Virtual Wallclock Time Scaling

In most environments, users submit jobs with rough estimations of the wallclock times. Within the HPC industry, a job typically runs for 40% of its specified wallclock time. Virtual Wallclock Time Scaling takes advantage of this fact to implement a form of optimistic backfilling. Jobs that are eligible for backfilling and not restricted by other policies are virtually scaled by the BFVIRTUALWALLTIMESCALINGFACTOR (assuming that the jobs finish before this new virtual wallclock limit). The scaled jobs are then compared to backfill windows to see if there is space and time for them to be scheduled. The scaled jobs are only scheduled if there is no possibility that it will conflict with a standing or administrator reservation. Conflicts with such reservations occur if the virtual wallclock time overlaps a reservation, or if the original non-virtual wallclock time overlaps a standing or administrator reservation. Jobs that can fit into an available backfill window without having their walltime scaled are backfilled "as-is" (meaning, without virtually scaling the original walltime).

Virtual Wallclock Time Scaling is only enabled when the BFVIRTUALWALLTIMESCALINGFACTOR parameter is defined.

If a virtually-scaled job fits into a window, and is backfilled, it will run until completion or until it comes within one scheduling iteration (RMPOLLINTERVAL defines the exact time of an iteration) of the virtual wallclock time expiration. In the latter case the job's wallclock time is restored to its original time and Moab checks and resolves conflicts caused by this "expansion." Conflicts may occur when the backfilled job is restored to its full duration resulting in reservation overlap. The BFVIRTUALWALLTIMECONFLICTPOLICY parameter controls how Moab handles these conflicts.

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

Related topics

- BACKFILLDEPTH Parameter
- BACKFILLPOLICY Parameter
- BFMINVIRTUALWALLTIME
- Reservation Policy Overview

8.3 Node Set Overview

- Node Set Usage Overview
- Node Set Configuration
  - Node Set Policy
  - Node Set Attribute
  - Node Set Constraint Handling
  - Node Set List
  - Node Set Tolerance
  - Node Set Priority
  - NODESETPLUS
  - Nested Node Sets
- Requesting Node Sets for Job Submission
- Configuring Node Sets for Classes

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 `ANYOF:FEATURE:bigmem,fastos` to request nodes with the `bigmem` or `fastos` feature. Alternatively, if a simple feature-homogeneous node set is desired, `ONEOF:FEATURE` may be specified. On the other hand, a job may request a feature based node set with the configuration `ONEOF:FEATURE:bigmem,fastos`, in which case Moab will first attempt to locate adequate nodes where all nodes contain the `bigmem` feature. If such a set cannot be found, Moab will look for sets of nodes containing the other specified features. In highly heterogeneous clusters, the use of node sets improves job throughput by 10 to 15%.

Node sets can be requested on a system wide or per job basis. System wide configuration is accomplished via the NODESET* parameters while per job specification occurs via the resource manager extensions. 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.

The GLOBAL node is included in all feature node sets.

Node Set Configuration

Global node sets are defined using the NODESETPOLICY, NODESETATTRIBUTE, NODESETLIST, and NODESETISOPTIONAL 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:

```
NODESETPOLICY     ONEOF
NODESETATTRIBUTE  FEATURE
NODESETISOPTIONAL TRUE
NODESETLIST       switchA,switchB,switchC,switchD
...
```

Node Set Policy

In the preceding example, the NODESETPOLICY parameter is set to the policy *ONEOF* and tells Moab to allocate nodes within a single attribute set. Other nodeset policies are listed in the following table:

Policy	Description
ANYOF	Select resources from all sets contained in node set list. The job could span multiple node sets.
FIRSTOF	Select resources from first set to match specified constraints.
ONEOF	Select a single set that contains adequate resources to support job.

Node Set Attribute

The example's NODESETATTRIBUTE parameter is set to *FEATURE* specifying that the node sets are to be constructed along node feature boundaries.

Node Set Constraint Handling

The next parameter, NODESETISOPTIONAL, indicates that Moab should not delay the start time of a job if the desired node set is not available but adequate idle resources exist outside of the set. Setting this parameter to *TRUE* basically tells Moab to attempt to use a node set if it is available, but if not, run the job as soon as possible anyway.

Setting NODESETISOPTIONAL to *FALSE* will force the job to always run in a complete nodeset regardless of any start delay this imposes.

Node Set List

Finally, the NODESETLIST value of `switchA switchB...` tells Moab to only use node sets based on the listed feature values. This is necessary since sites will often use node features for many purposes and the resulting node sets would be of little use for switch proximity if they were generated based on irrelevant node features indicating things such as processor speed or node architecture.

To add nodes to the NODESETLIST, you must configure features on your nodes using the NODECFG **FEATURES** on page 475 attribute.

```
NODECFG[node01] FEATURES+=switchA
NODECFG[node02] FEATURES+=switchA
NODECFG[node03] FEATURES+=switchB
```

Nodes node01 and node02 contain the switchA feature, and node node03 contains the switchB feature.

Node Set Priority

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

Priority Type	Description	Details
AFFINITY	Avoid a resource set with negative affinity.	Choosing this type causes Moab to select a node set with no negative affinity nodes (nodes that have a reservation that with negative affinity). If all node sets have negative affinity, then Moab will select the first matching node set.
BESTFIT	Select the smallest resource set possible.	Choosing this type causes Moab, when selecting a node set, to eliminate sets that do not have all the required resources. From the remaining sets, Moab chooses the set with the least amount of resources. This priority type most closely matches the job requirements in order to waste the least amount of resources. This type minimizes fragmentation of larger resource sets.
MINLOSS	Select the resource set that results in the minimal wasted resources assuming no internal job load balancing is available. (Assumes parallel jobs only run as fast as the slowest allocated node.)	Choosing this type works only when using the following configuration: NODESETATTRIBUTE *FEATURE* In a *SHAREDMEM* environment (See **Moab-NUMA Integration Guide** on page 1008 for more information.), Moab will select the node set based on NUMA properties (the smallest feasible node set).

Priority Type	Description	Details
WORSTFIT	Select the largest resource set possible.	This type causes Moab, when choosing a node set, to eliminate sets that do not have all the required resources. From the remaining sets, Moab chooses the set with the greatest amount of resources. This type minimizes fragmentation of smaller resource sets, but increases fragmentation of larger resource sets.

NODESETPLUS

Moab supports additional node set behavior by specifying the NODESETPLUS parameter. Possible values when specifying this parameter are *SPANEVENLY* and *DELAY*.

Neither *SPANEVENLY* nor *DELAY* will work with multi-req jobs or preemption.

SPANEVENLY

Moab attempts to fit all jobs within one node set, or it spans any number of node sets evenly. When a job specifies a NODESETDELAY, Moab attempts to contain the job within a single node set; if unable to do so, it spans node sets evenly, unless doing so would delay the job beyond the requested **NODESETDELAY**.

DELAY

Moab attempts to 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 **NODESETDELAY**.

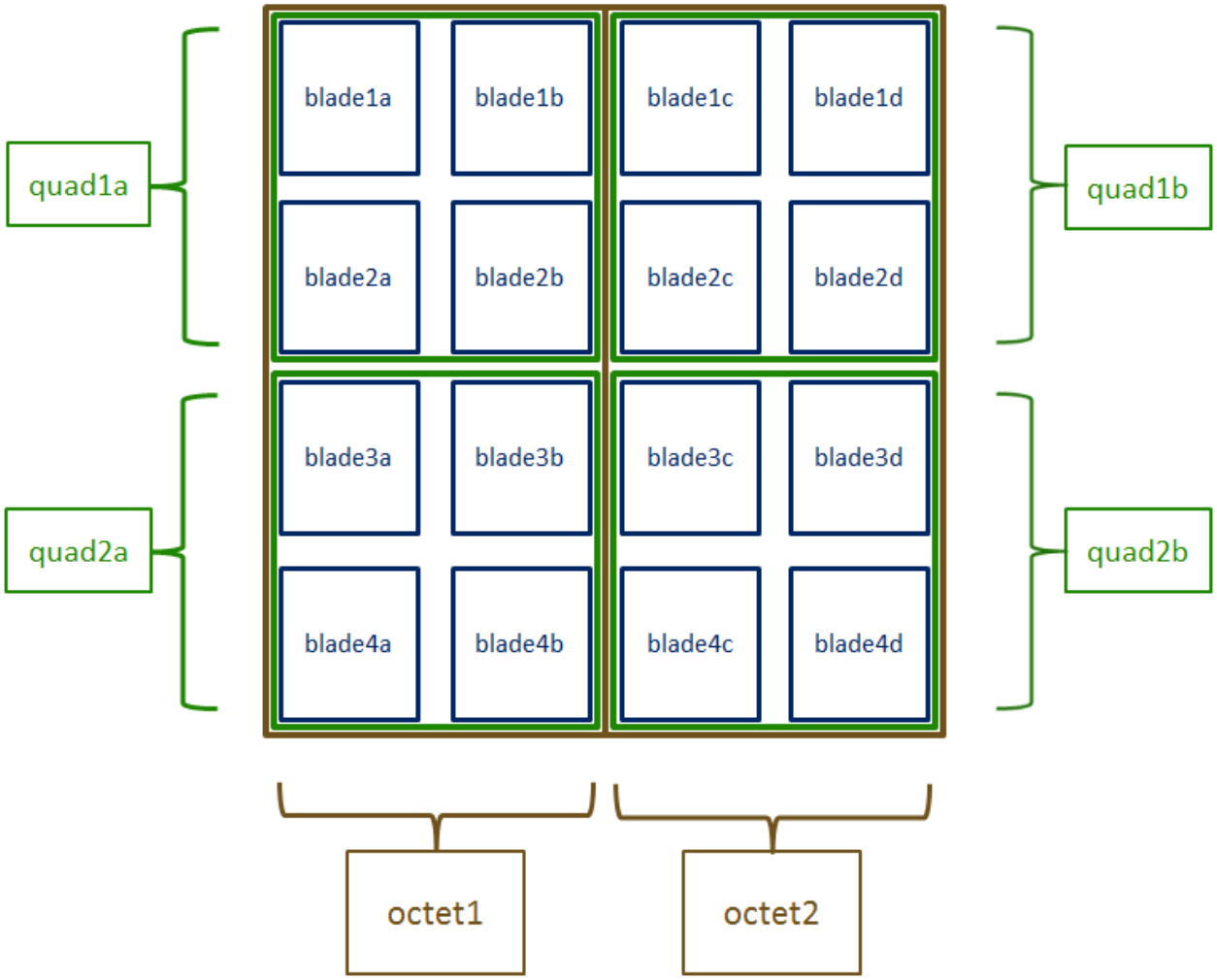
Nested Node Sets

Moab attempts to fit jobs on node sets in the order they are specified in the **NODESETLIST**. You can create nested node sets by listing your node sets in a specific order. Here is an example of a "smallest to largest" nested node set:

```
NODESETPOLICY ONEOF
NODESETATTRIBUTE FEATURE
NODESETISOPTIONAL FALSE
NODESETLIST blade1a,blade1b,blade2a,blade2b,blade3a,
blade3b,blade4a,blade4b,quad1a,quad1b,quad2a,
quad2b,octet1,octet2,sixteen
```

The accompanying cluster would look like this:

Image 8-2: Octet, quad, and blade node sets on a cluster



In this example, Moab tries to fit the job on the nodes in the blade sets first. If that doesn't work, it moves up to the nodes in the quad sets (a set of four blade sets). If the quads are insufficient, it tries the nodes in the octet sets (a set of four quad node sets).

Requesting Node Sets for Job Submission

On a per job basis, each user can specify the equivalent of all parameters except **NODESETDELAY**. As mentioned previously, this is accomplished using the resource manager extensions.

Configuring Node Sets for Classes

Classes can be configured with a default node set. In the configuration file, specify **DEFAULT.NODESET** with the following syntax: `DEFAULT.NODESET=<SETTYPE>:<SETATTR>[:<SETLIST>[,<SETLIST>]...]`. For example, in a heterogeneous cluster with two different types of processors, the following configuration confines jobs assigned to the `amd` class to run on either `ATHLON` or `OPTERON` processors:

```
CLASSCFG[amd] DEFAULT.NODESET=ONEOF:FEATURE:ATHLON,OPTERON
```

```
...
```

Related topics

- Resource Manager Extensions
- CLASSCFG
- Partition Overview

9.0 Evaluating System Performance - Statistics, Profiling, Testing, and Simulation

- **Moab Performance Evaluation Overview** on page 445
- **Accounting: Job and System Statistics** on page 445
- **Testing New Versions and Configurations** on page 447
- **Answering What If? Questions with the Simulator** on page 448

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.

- Accounting Overview
- Real-Time Statistics
- FairShare Usage Statistics

Accounting Overview

Moab provides accounting data correlated to most major objects used within the cluster scheduling environment. These records provide job and reservation accounting, resource accounting, and credential-based accounting.

Job and Reservation Accounting

As each job or reservation completes, Moab creates a complete persistent trace record containing information about who ran, the time frame of all significant events, and what resources were allocated. In addition, actual execution environment, failure reports, requested service levels, and other pieces of key information are also recorded. A complete description of each accounting data field can be found within section Workload Traces.

Resource Accounting

The load on any given node is available historically allowing identification of not only its usage at any point in time, but the actual jobs which were running on it. Moab Cluster Manager can show load information (assuming load is configured as a generic metric), but not the individual jobs that were running on a node at some point in the past. For aggregated, historical statistics covering node usage and availability, the showstats command may be run with the `-n` flag.

Credential Accounting

Current and historical usage for users, groups, account, QoSes, and classes are determined in a manner similar to that available for evaluating nodes. For aggregated, historical statistics covering credential usage and availability, the showstats command may be run with the corresponding credential flag.

If needed, detailed credential accounting can also be enabled globally or on a credential by credential basis. With detailed credential accounting enabled, real-time information regarding per-credential usage over time can be displayed. To enable detailed per credential accounting, the **ENABLEPROFILING** attribute must be specified for credentials that are to be monitored. For example, to track detailed credentials, the following should be used:

```
USERCFG[DEFAULT]      ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]       ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]     ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]     ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]   ENABLEPROFILING=TRUE
```

Credential level profiling operates by maintaining a number of time-based statistical records for each credential. The parameters PROFILECOUNT and PROFILEDURATION control the number and duration of the statistical records.

Real-Time Statistics

Moab provides real-time statistical information about how the machine is running from a scheduling point of view. The showstats command is actually a suite of commands providing detailed information on

an overall scheduling basis as well as a per user, group, account and node basis. This command gets its information from in memory statistics that are loaded at scheduler start time from the scheduler checkpoint file. (See Checkpoint/Restart for more information.) This checkpoint file is updated periodically and when the scheduler is shut down allowing statistics to be collected over an extended time frame. At any time, real-time statistics can be reset using the mschedctl -f command.

In addition to the showstats command, the showstats -f command also obtains its information from the in memory statistics and checkpoint file. This command displays a processor-time based matrix of scheduling performance for a wide variety of metrics. Information such as backfill effectiveness or average job queue time can be determined on a job size/duration basis.

FairShare Usage Statistics

Regardless of whether fairshare is enabled, detailed credential based fairshare statistics are maintained. Like job traces, these statistics are stored in the directory pointed to by the STATDIR parameter. Fairshare stats are maintained in a separate statistics file using the format `FS.<EPOCHTIME>` (`FS.982713600`, for example) with one file created per fairshare window. (See the Fairshare Overview for more information.) These files are also flat text and record credential based usage statistics. Information from these files can be seen via the mdiag -f command.

Related topics

- Simulation Overview
- Generic Consumable Resources
- Object Variables
- Generic Event Counters

9.3 Testing New Versions and Configurations

- MONITOR Mode
- INTERACTIVE Mode

MONITOR Mode

Moab supports a scheduling mode called *MONITOR*. In this mode, the scheduler initializes, contacts the resource manager and other peer services, and conducts scheduling cycles exactly as it would if running in *NORMAL* or production mode. Job are prioritized, reservations created, policies and limits enforced, and administrator and end-user commands enabled. The key difference is that although live resource management information is loaded, *MONITOR* mode disables Moab's ability to start, preempt, cancel, or otherwise modify jobs or resources. Moab continues to attempt to schedule exactly as it would in *NORMAL* mode but its ability to actually impact the system is disabled. Using this mode, a site can quickly verify correct resource manager configuration and scheduler operation. This mode can also be used to validate new policies and constraints. In fact, Moab can be run in *MONITOR* mode on a production system while another scheduler or even another version of Moab is running on the same system. This unique ability can allow new versions and configurations to be fully tested without any exposure to potential failures and with no cluster downtime.

To run Moab in *MONITOR* mode, simply set the **MODE** attribute of the **SCHEDCFG** parameter to *MONITOR* and start Moab. Normal scheduler commands can be used to evaluate configuration and performance. Diagnostic commands can be used to look for any potential issues. Further, the Moab log file can be used to determine which jobs Moab attempted to start, and which resources Moab attempted to allocate.

If another instance of Moab is running in production and a site administrator wants to evaluate an alternate configuration or new version, this is easily done but care should be taken to avoid conflicts with the primary scheduler. Potential conflicts include statistics files, logs, checkpoint files, and user interface ports. One of the easiest ways to avoid these conflicts is to create a new test directory with its own log and stats subdirectories. The new `moab.cfg` file can be created from scratch or based on the existing *moab.cfg* file already in use. In either case, make certain that the **PORT** attribute of the **SCHEDCFG** parameter differs from that used by the production scheduler by at least two ports. If testing with the production binary executable, the MOABHOMEDIR environment variable should be set to point to the new test directory to prevent Moab from loading the production `moab.cfg` file.

INTERACTIVE Mode

INTERACTIVE mode allows for evaluation of new versions and configurations in a manner different from *MONITOR* mode. Instead of disabling all resource and job control functions, Moab sends the desired change request to the screen and asks for permission to complete it. For example, before starting a job, Moab may print something like the following to the screen:

```
Command:  start job 1139.ncsa.edu on node list test013,test017,test018,test021
Accept:  (y/n) [default: n]?
```

The administrator must specifically accept each command request after verifying it correctly meets desired site policies. Moab then executes the specified command. This mode is highly useful in validating scheduler behavior and can be used until configuration is appropriately tuned and all parties are comfortable with the scheduler's performance. In most cases, sites will want to set the scheduling mode to *NORMAL* after verifying correct behavior.

Related topics

- Testing New Releases and Policies
- 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 General Job Administration

- **Job Holds** on page 449
- **Job Priority Management** on page 451
- **Suspend/Resume Handling** on page 451
- **Checkpoint/Restart Facilities** on page 452
- **Job Dependencies** on page 452
- **Job Defaults and Per Job Limits** on page 454
- **General Job Policies** on page 455
- **Using a Local Queue** on page 457
- **Job Deadlines** on page 460
- **Job Arrays** on page 462

10.1 Job Holds

Holds and Deferred Jobs

Moab supports job holds applied by users (user holds), administrators (system holds), and resource managers (batch holds). There is also a temporary hold known as a job defer.

User Holds

User holds are very straightforward. Many, if not most, resource managers provide interfaces by which users can place a hold on their own job that tells the scheduler not to run the job while the hold is in place. Users may use this capability because the job's data is not yet ready, or they want to be present when the job runs to monitor results. Such user holds are created by, and under the control of a non-privileged user and may be removed at any time by that user. As would be expected, users can only place holds on their jobs. Jobs with a user hold in place will have a Moab state of `Hold` or `UserHold` depending on the resource manager being used.

System Holds

The system hold is put in place by a system administrator either manually or by way of an automated tool. As with all holds, the job is not allowed to run so long as this hold is in place. A batch administrator can place and release system holds on any job regardless of job ownership. However, unlike a user hold, normal users cannot release a system hold even on their own jobs. System holds are often used during system maintenance and to prevent particular jobs from running in accordance with current system

needs. Jobs with a system hold in place will have a Moab state of `Hold` or `SystemHold` depending on the resource manager being used.

Batch Holds

Batch holds are placed on a job by the scheduler itself when it determines that a job cannot run. The reasons for this vary but can be displayed by issuing the checkjob<*JOBID*> command. Possible reasons are included in the following list:

- No Resources — The job requests resources of a type or amount that do not exist on the system.
- System Limits — The job is larger or longer than what is allowed by the specified system policies.
- Bank Failure — The allocations bank is experiencing failures.
- No Allocations — The job requests use of an account that is out of allocations and no fallback account has been specified.
- RM Reject — The resource manager refuses to start the job.
- RM Failure — The resource manager is experiencing failures.
- Policy Violation — The job violates certain throttling policies preventing it from running now and in the future.
- No QOS Access — The job does not have access to the QoS level it requests.

Jobs which are placed in a batch hold will show up within Moab in the state `BatchHold`.

Job Defer

In most cases, a job violating these policies is not placed into a batch hold immediately; rather, it is deferred. The parameter DEFERTIME indicates how long it is deferred. At this time, it is allowed back into the idle queue and again considered for scheduling. If it again is unable to run at that time or at any time in the future, it is again deferred for the timeframe specified by DEFERTIME. A job is released and deferred up to DEFERCOUNT times at which point the scheduler places a batch hold on the job and waits for a system administrator to determine the correct course of action. Deferred jobs have a Moab state of `Deferred`. As with jobs in the BatchHold state, the reason the job was deferred can be determined by use of the checkjob command.

At any time, a job can be released from any hold or deferred state using the releasehold command. The Moab logs should provide detailed information about the cause of any batch hold or job deferral.

Under Moab, the reason a job is deferred or placed in a batch hold is stored in memory but is not checkpointed. Thus this information is available only until Moab is recycled at which point the checkjob command no longer displays this reason information.

Related topics

- DEFERSTARTCOUNT - number of job start failures allowed before job is deferred

10.2 Job Priority Management

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

- Priority Configuration - see Job Prioritization
- Manual Intervention with setspri

10.3 Suspend/Resume Handling

When supported by the resource manager, Moab can suspend and resume jobs. A user can suspend his/her own jobs, but only an administrator can resume them. By default, a job is suspended for one minute before it can resume. You can modify this default time using the MINADMINSTIME parameter.

A job must be marked as `suspendable` for Moab to suspend and resume it. To do so, either submit the job with the *suspendable* flag attached to it or configure a credential to pass the flag to its associated jobs. These methods are demonstrated in the examples below:

```
msub -l flags=suspendable
```

```
GROUPCFG[default] JOBFLAGS=SUSPENDABLE
```

Once the job is suspendable, Moab allows you to suspend jobs using the two following methods: (1) manually on the command line and (2) automatically in the `moab.cfg` file.

To manually suspend jobs, use the mjobctl command as demonstrated in the following examplee:

```
> mjobctl -s job05
```

Moab suspends `job05`, preventing it from running immediately in the job queue.

If you are an administrator and want to resume a job, use the mjobctl command as demonstrated in the following example:

```
> mjobctl -r job05
```

Moab removes `job05` from a suspended state and allows it to run.

You can also configure the Moab preemption policy to suspend and resume jobs automatically by setting the PREEMPTPOLICY parameter to *SUSPEND*. A sample Moab configuration looks like this:

```
PREEMPTPOLICY SUSPEND
...
USERCFG[tom] JOBFLAGS=SUSPENDABLE
```

Moab suspends jobs submitted by user *tom* if necessary to make resources available for jobs with higher priority.

If your resource manager has a native interface, you must configure JOBSUSPENDURL to suspend and resume jobs.

For more information about suspending and resuming jobs in Moab, see the following sections:

- manual preemption with the mjobctl command
- Job preemption

10.4 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.

Related topics

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

10.5 Job Dependencies

- Basic Job Dependency Support
  - Job Dependency Syntax

Basic Job Dependency Support

By default, basic single step job dependencies are supported through completed/failed step evaluation. Basic dependency support does not require special configuration and is activated by default. Dependent jobs are only supported through a resource manager and therefore submission methods depend upon the specific resource manager being used. For the TORQUE qsub command, the semantics listed in the section below can be used with the `-W x=depend=<STRING>` or `-W depend=<STRING>` flag; for the Moab msub command, the `-l depend=<STRING>` or `-W x=depend=<STRING>` flag. For other resource managers, consult the resource manager specific documentation.

Job Dependency Syntax

Dependency	Format	Description
after	`after:<job>[:<job>]...`	Job may start at any time after specified jobs have started execution.
afterany	`afterany:<job>[:<job>]...`	Job may start at any time after all specified jobs have completed regardless of completion status.
afterok	`afterok:<job>[:<job>]...`	Job may be start at any time after all specified jobs have successfully completed.
afternotok	`afternotok:<job>[:<job>]...`	Job may start at any time after all specified jobs have completed unsuccessfully.
before	`before:<job>[:<job>]...`	Job may start at any time before specified jobs have started execution.
beforeany	`beforeany:<job>[:<job>]...`	Job may start at any time before all specified jobs have completed regardless of completion status.
beforeok	`beforeok:<job>[:<job>]...`	Job may start at any time before all specified jobs have successfully completed.
beforenotok	`beforenotok:<job>[:<job>]...`	Job may start at any time before any specified jobs have completed unsuccessfully.
on	`on:<count>`	Job may start after *<count>* dependencies on other jobs have been satisfied.
synccount	`synccount:<count>`	Job is the first in a set of jobs to be executed at the same time. *<count>* is the number of additional jobs in the set, which can be up to 5. synccount is valid for single-request jobs with TORQUE as the resource manager.
syncwith	`syncwith:<job>`	Job is an additional member of a set of jobs to be executed at the same time. Moab supports up to 5 jobs. syncwith is valid for single-request jobs with TORQUE as the resource manager.

<job>=`{JOBNAME.jobname|jobid}`

When using JobName dependencies, prepend "JOBNAME." to avoid ambiguity.

The `before` dependencies do not work with jobs submitted with msub; they work only with qsub.

Any of the dependencies containing `before` must be used in conjunction with the `on` dependency. So, if job A must run before job B, job B must be submitted with *depend=on:1*, as well as job A having *depend=before:A*. This means job B cannot run until one dependency of another job on job B has been fulfilled. This prevents job B from running until job A can be successfully submitted.

Related topics

- Job Deadlines

10.6 Job Defaults and Per Job Limits

Job Defaults

Job defaults can be specified on a per queue basis. These defaults are specified using the CLASSCFG parameter. The following table shows the applicable attributes:

Attribute	Format	Example
DEFAULT.FEATURES	comma-delimited list of node features	`CLASSCFG[batch] DEFAULT.FEATURES=fast,io` *Jobs submitted to class batch will request nodes features fast and io.*
DEFAULT.WCLIMIT	`[[[DD:]HH:]MM:]SS`	`CLASSCFG[batch] DEFAULT.WCLIMIT=1:00:00` *Jobs submitted to class batch will request one hour of walltime by default.*

Per Job Maximum Limits

Job maximum limits can be specified on a per queue basis. These defaults are specified using the CLASSCFG parameter. The following table shows the applicable attributes:

Attribute	Format	Example
MAX.WCLIMIT	`[[[DD:]HH:]MM:]SS`	`CLASSCFG[batch] MAX.WCLIMIT=1:00:00` *Jobs submitted to class batch can request no more than one hour of walltime.*

Per Job Minimum Limits

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

Attribute	Format	Example
MIN.PROC	*<integer>*	`CLASSCFG[batch] MIN.PROC=10` *Jobs submitted to class batch can request no less than ten processors.*

Related topics

- Usage-based Limits

10.7 General Job Policies

- Multi-Node Support
- Multi-Req Support
- Job Size Policy
- Malleable Job Support
- Enabling Job User Proxy

There are a number of configurable policies that help control advanced job functions. These policies help determine allowable job sizes and structures.

Multi-Node Support

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

Multi-Req Support

Jobs can specify multiple types of resources for allocation. For example, a job could request 4 nodes with 256 MB of memory and 8 nodes with feature `fast` present.

Resources specified in a multi-req job are delimited with a plus sign (+).

Neither *SPANEVENLY* nor *DELAY* values of the NODESETPLUS parameter will work with multi-req jobs or preemption.

Example 10-1:

```
-l nodes=4:ppn=1+10:ppn=5+2:ppn=2
```

*This example requests 4 nodes with 1 proc each, 10 nodes with 5 procs each, and 2 nodes with 2 procs each. The total number of processors requested is (4*1) + (10*5) + (2*2), or 58 processors.*

Example 10-2:

```
-l nodes=15+1:ppn=4
```

The job submitted in this example requests a total of 16 nodes. 15 of these nodes have no specific requirements, but the remaining node must have 4 processors.

Example 10-3:

```
-l nodes=3:fast+1:io
```

The job requests a total of 4 nodes: 3 nodes with the fast feature and 1 node with the io feature.

Job Size Policy

Moab allows jobs to request resource ranges. Using this range information, the scheduler is able to maximize the amount of resources available to the job while minimizing the amount of time the job is blocked waiting for resources. The JOBSIZEPOLICY parameter can be used to set this behavior according to local site needs.

Job resource ranges may only be specified when using a local queue as described in the Using a Local Queue section.

Malleable Job Support

A job can specify whether it is able to use more processors or less processors and what effect, if any, that has on its wallclock time. For example, a job may run for 10 minutes on 1 processor, 5 minutes on 2 processors and 3 minutes on 3 processors. When a job is submitted with a task request list attached, Moab determines which task request fits best and molds the job based on its specifications. To submit a job with a task request list and allow Moab to mold it based on the current scheduler environment, use the TRL flag in the Resource Manager Extension.

Enabling Job User Proxy

By default, user proxying is disabled. To be enabled, it must be authorized using the **PROXYLIST** attribute of the USERCFG parameter. This parameter can be specified either as a comma-delimited list of users or as the keyword *validate*. If the keyword *validate* is specified, the RMCFG attribute **JOBVALIDATEURL** should be set and used to confirm that the job's owner can proxy to the job's execution user. An example script performing this check for ssh-based systems is provided in the `tools` directory.(See Job Validate Tool Overview.)

For some resource managers (RM), proxying must also be enabled at the RM level. The following example shows how ssh-based proxying can be accomplished in a Moab+TORQUE with SSH environment.

To validate proxy users, Moab must be running as root.

Example 10-4: SSH Proxy Settings

```
USERCFG[DEFAULT] PROXYLIST=validate
RMCFG[base]  TYPE=<resource manager>
JOBVALIDATEURL=exec://$HOME/tools/job.validate.sshproxy.pl
```

```
> qmgr -c 's s allow proxy user=true'
> su - testuser
> qsub -I -u testuser2
qsub: waiting for job 533.igt.org to start
qsub: job 533.igt.org ready
testuser2@igt:~$
```

In this example, the validate tool, '`job.validate.sshproxy.pl`', can verify proxying is allowed by becoming the submit user and determining if the submit user can achieve passwordless access to the specified execution user. However, site-specific tools can use any method to determine proxy access including a flat file look-up, database lookup, querying of an information service such as NIS or LDAP, or other local or remote tests. For example, if proxy validation is required but end-user accounts are not available on the management node running Moab, the job validate service could perform the validation test on a representative remote host such as a login host.

This feature supports qsub only.

The job validate tool is highly flexible allowing any combination of job attributes to be evaluated and tested using either local or remote validation tests. The validate tool allows not only pass/fail responses but also allows the job to be modified, or rejected in a custom manner depending on the site or the nature of the failure.

Related topics

- Usage Limits

10.8 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.

Local Queue Configuration

A local queue is configured just like a standard resource manager queue. It may have defaults, limits, resource mapping, and credential access constraints. The following table describes the most common settings:

Default queue	
Format	`RMCFG[internal] DEFAULTCLASS=<CLASSID>`
Description	The job class/queue assigned to the job if one is not explicitly requested by the submitter. All jobs submitted directly to Moab are initially received by the pseudo-resource manager *internal*. Therefore, default queue configuration may only be applied to it.
Example	`RMCFG[internal] DEFAULTCLASS=batch`

Class default resource requirements	
Format	`CLASSCFG[<CLASSID>] DEFAULT.FEATURES=<X> CLASSCFG[<CLASSID>] DEFAULT.MEM=<X> CLASSCFG[<CLASSID>] DEFAULT.NODE=<X> CLASSCFG[<CLASSID>] DEFAULT.NODESET=<X> CLASSCFG[<CLASSID>] DEFAULT.PROC=<X> CLASSCFG [<CLASSID>] DEFAULT.WCLIMIT=<X>`
Description	The settings assigned to the job if not explicitly set by the submitter. Default values are available for node features, per task memory, node count, nodeset configuration, processor count, and wallclock limit.

Example	`CLASSCFG[batch] DEFAULT.WCLIMIT=4 DEFAULT.FEATURES=matlab`
	or
	`CLASSCFG[batch] DEFAULT.WCLIMIT=4`
	`CLASSCFG[batch] DEFAULT.FEATURES=matlab`

Class maximum resource limits	
Format	`CLASSCFG[<CLASSID>] MAX.FEATURES=<X> CLASSCFG[<CLASSID>] MAX.NODE=<X> CLASSCFG[<CLASSID>] MAX.PROC=<X> CLASSCFG[<CLASSID>] MAX.WCLIMIT=<X>`

Class maximum resource limits	
Description	The maximum node features, node count, processor count, and wallclock limit allowed for a job submitted to the class/queue. If these limits are not satisfied, the job is not accepted and the submit request fails. **MAX.FEATURES** indicates that only the listed features may be requested by a job.

Example	`CLASSCFG[smalljob] MAX.PROC=4 MAX.FEATURES=slow,matlab`
	or
	`CLASSCFG[smalljob] MAX.PROC=4`
	`CLASSCFG[smalljob] MAX.FEATURES=slow,matlab`

Class minimum resource limits	
Format	`CLASSCFG[<CLASSID>] MIN.FEATURES=<X> CLASSCFG[<CLASSID>] MIN.NODE=<X>` `CLASSCFG[<CLASSID>] MIN.PROC=<X> CLASSCFG[<CLASSID>] MIN.WCLIMIT=<X>`
Description	The minimum node features, node count, processor count, and wallclock limit allowed for a job submitted to the class/queue. If these limits are not satisfied, the job is not accepted and the submit request fails. **MIN.FEATURES** indicates that only the listed features may be requested by a job.

Example	`CLASSCFG[bigjob] MIN.PROC=4 MIN.WCLIMIT=1:00:00`
	or
	`CLASSCFG[bigjob] MIN.PROC=4`
	`CLASSCFG[bigjob] MIN.WCLIMIT=1:00:00`

Class access	
Format	`CLASSCFG[<CLASSID>] REQUIREDUSERLIST=<USERID>[,<USERID>]...`
Description	The list of users who may submit jobs to the queue.
Example	`CLASSCFG[math] REQUIREDUSERLIST=john,steve`

Available resources	
Format	CLASSCFG[*<CLASSID>*] HOSTLIST=*<HOSTID>*[,*<HOSTID>*]...
Description	The list of nodes that jobs in the queue may use.
Example	`CLASSCFG[special] HOSTLIST=node001,node003,node13`

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.

10.9 Job Deadlines

- Deadline Overview
- **Setting Job Deadlines via QoS** on page 460
  - **Setting Job Deadlines at Job Submission** on page 461
  - **Submitting a Job to a QoS with a Preconfigured Deadline** on page 461
- Job Termination Date
- Conflict Policies

Deadline Overview

Job deadlines may be specified on a per job and per credential basis and are also supported using both absolute and QoS based specifications. A job requesting a deadline is first evaluated to determine if the deadline is acceptable. If so, Moab adds it to the list of deadline jobs and allocates resources to guarantee that all accepted deadline jobs are able to complete on or before their requested deadline. Once the scheduler confirms that all deadlines can be satisfied, it then optimizes resource allocation (in priority order) attempting to execute all jobs at the earliest possible time.

Setting Job Deadlines via QoS

Two types of job deadlines exist in Moab. The priority-based deadline linearly increases a job's priority as its deadline approaches (See **Deadline (DEADLINE) Subcomponent** on page 294 for more information). The QoS method allows you to set a job completion time on job submission if, and only if, it requests and is allowed to access a QoS with the *DEADLINE* QFLAG set. This method is more powerful than the priority method, because Moab will attempt to make a reservation for the job as soon as the job enters the queue in order to meet the deadline, essentially bumping it to the front of the queue.

When a job is submitted to a QoS with the *DEADLINE* flag set, the job's `-l deadline` attribute is honored. If such QoS access is not available, or if resources do not exist at job submission time to allow the deadline to be satisfied, the job's deadline request is ignored.

Two methods exist for setting deadlines with a QoS:

- Submitting a job to a deadline-enabled QoS and specifying a deadline using msub -l.
- Submitting a job to a deadline-enabled QoS with a QTTARGET specified.

Setting Job Deadlines at Job Submission

This method of setting a job deadline allows you to specify a job deadline as you submit the job. You can set the deadline as either an exact date and time or as an amount of time after job submission (i.e. three hours after submission).

To specify a deadline on job submission

1. In `moab.cfg`, reate a QoS with the *DEADLINE* flag enabled.

```
...
QOSCFG[special] QFLAGS=DEADLINE
```

Jobs requesting the QoS `special` may submit jobs with a deadline that Moab will honor.

2. Submit a job to the QoS and set a deadline. This can be either absolute or relative.

   a. For an absolute deadline, use the format `hh:mm:ss_mm/dd/yy`. The following configuration sets a deadline for a job to finish by 8 a.m. on March 15th, 2013.

```
msub -l qos=special deadline=08:00:00_03/15/13 job.sh
```

The job must finish running by 8 A.M. on March 15, 2013.

   b. For a relative deadline, or the completion deadline of the job relative to its submission time, use the time format `[[[DD:]HH:]MM:]SS`.

```
msub -l qos=special deadline=5:00:00 job.sh
```

The job's deadline is 5 hours after its submission.

Submitting a Job to a QoS with a Preconfigured Deadline

You may also set a relative job deadline by limiting the job's queue time. This method allows you to pre-configure the deadline rather than giving the power to specify a deadline to the user submitting the job. For jobs requesting these QoSes, Moab identifies and sets job deadlines to satisfy the corresponding response time targets.

To submit a job to a QoS with a preconfigured deadline

1. In `moab.cfg`, create a QoS with both the *DEADLINE* **QFLAG** and a response time target (**QTTARGET**). The **QTTARGET** is the maximum amount of time that Moab should allow the job to be idle in the queue.

```
...
QOSCFG[special2] QFLAGS=DEADLINE QTTARGET=1:00:00
```

Given this configuration, a job requesting QoS special2 must spend a maximum of one hour in the queue.

2. Submit a job requesting the `special2` quality of service.

```
msub -l qos=special2 walltime=2:00:00 job.sh
```

This two-hour job has a completion time deadline set to three hours after its submission (one hour of target queue time and two hours of run time).

Job Termination Date

In addition to job completion targets, jobs may also be submitted with a TERMTIME attribute. The scheduler attempts to complete the job prior to the termination date, but if it is unsuccessful, it will terminate (cancel) the job once the termination date is reached.

Conflict Policies

The specific policy can be configured using the DEADLINEPOLICY parameter. Moab does not have a default policy for this parameter.

Policy	Description
CANCEL	The job is canceled and the user is notified that the deadline could not be satisfied.
HOLD	The job has a batch hold placed on it indefinitely. The administrator can then decide what action to take.
RETRY	The job continually retries each iteration to meet its deadline; note that when used with `QTTARGET` the job's deadline continues to slide with relative time.
IGNORE	The job has its request ignored and is scheduled as normal.

Deadline scheduling may not function properly with per partition scheduling enabled. Check that PARALLOCATIONPOLICY is disabled to ensure DEADLINEPOLICY will work correctly.

Related topics

- QoS Facilities
- Job Submission Eligible Start Time constraints

10.10 Job Arrays

- Job Array Overview
- Enabling Job Arrays
- Sub-job Definitions

- Using Environment Variables to Specify Array Index Values
  - Control
  - Reporting
- Job Array Cancellation Policies
- Examples
  - Submitting Job Arrays

Job Array Overview

You can submit an array of jobs to Moab via the msub command. Array jobs are an easy way to submit many sub-jobs that perform the same work using the same script, but operate on different sets of data. Sub-jobs are the jobs created by an array job and are identified by the array job ID and an index; for example, if `235[1]` is an identifier, the number `235` is a job array ID, and `1` is the sub-job.

Sub-jobs of an array are executed in sub-job index order.

Moab job arrays are different from TORQUE job arrays.

Enabling Job Arrays

To enable job arrays, include the ENABLEJOBARRAYS parameter in the Moab configuration file (`moab.cfg`).

Sub-job Definitions

Like a normal job, an array job submits a job script, but it additionally has a start index (`sidx`) and an end index (`eidx`); array jobs also have increment (`incr`) values, which Moab uses to create sub-jobs, all executing the same script. The model for sub-job creation follows the formula of end index minus start index plus increment divided by the increment value: (`eidx - sidx + incr`) / `incr`.

To illustrate, suppose an array job has a start index of 1, an end index of 100, and an increment of 1. This is an array job that creates (100 - 1 + 1) / 1 = 100 sub-jobs with indexes of 1, 2, 3, ..., 100. An increment of 2 produces (100 - 1 + 2) / 2 = 50 sub-jobs with indexes of 1, 3, 5, ..., 99. An increment of 2 with a start index of 2 produces (100 - 2 + 2) / 2 = 50 sub-jobs with indexes of 2, 4, 6, ..., 100. Again, sub-jobs are jobs in their own right that have a slightly different job naming convention `jobID[subJobIndex]` (e.g. `mycluster.45[37]` or `45[37]`).

Using Environment Variables to Specify Array Index Values

The script can use an environment variable to obtain the array index value to form data file and/or directory names unique to an array job's particular sub-job. The following two environment variables are supplied so job scripts can recognize what index in the array they are in; use the msub command with the -V option to pass the environment parameters to the resource manager, or include the parameters in a job script; for example: `#PBS -V MOAB_JOBARRAYRANGE`.

Environment Parameter	Description

MOAB_ JOBARRAYINDEX	Used to create dataset file names, directory names, and so forth, when splitting up a single problem into multiple jobs.
	For example, a user may split up a problem into 20 separate jobs, each with its own input and output data files whose names contain the numbers 1-20.
	To illustrate, assume a user submits the 20 sub-jobs using two msub commands; one to submit the ten even-numbered jobs and one to submit the ten odd-numbered jobs.
	`msub -t job1.[1-20:2]`
	`msub -t job2.[2-20:2]`
	The `MOAB_JOBARRAYINDEX` environment variable value would populate each of the two job arrays' ten sub-jobs as `1`, `3`, `5`, `7`, `9`, `11`, `13`, `15`, `17` and `19` for the first array job's ten sub-jobs, and `2`, `4`, `6`, `8`, `10`, `12`, `14`, `16`, `18`, and `20` for the second array job's ten sub-jobs.

MOAB_ JOBARRAYRANGE	The count of jobs in the array.

Control

Users can control individual sub-jobs in the same manner as normal jobs. In addition, an array job represents its group of sub-jobs and any user or administrator commands performed on an array job apply to its sub-jobs; for example, the command canceljob *<arrayJobId>* cancels all sub-jobs that belong to the array job. For more information about job control, see the documentation for the mjobctl command.

Reporting

In the first example below, the parts unique to array subjobs are in red.

```
$ checkjob -v Moab.1[1]
job Moab.1[1]

AName: Moab
State: Running
Creds:  user:user1  group:usergroup1
WallTime:   00:00:17 of 8:20:00
SubmitTime: Thu Nov  4 11:50:03
(Time Queued  Total: 00:00:00  Eligible:   INFINITY)
StartTime: Thu Nov  4 11:50:03
Total Requested Tasks: 1
Req[0]  TaskCount: 1  Partition: base
Average Utilized Procs: 0.96
NodeCount:  1
Allocated Nodes:
[node010:1]

Job Group:        Moab.1
Parent Array ID:  Moab.1
Array Index:      1
Array Range:      10
SystemID:   Moab
SystemJID:  Moab.1[1]
Task Distribution: node010
IWD:            /home/user1
```

```
UMask:          0000
Executable:     /opt/moab/spool/moab.job.3CvNjl
StartCount:     1
Partition List: base
SrcRM:          internal  DstRM: base  DstRMJID: Moab.1[1]
Flags:          ARRAYJOB,GLOBALQUEUE
StartPriority:  1
PE:             1.00
Reservation 'Moab.1[1]' (-00:00:19 -> 8:19:41  Duration: 8:20:00)
```

If the array range is not provided, the output displays all the jobs in the array.

```
$ checkjob -v Moab.1
job Moab.1

AName: Moab
Job Array Info:
  Name: Moab.1
  1 : Moab.1[1] : Running
  2 : Moab.1[2] : Running
  3 : Moab.1[3] : Running
  4 : Moab.1[4] : Running
  5 : Moab.1[5] : Running
  6 : Moab.1[6] : Running
  7 : Moab.1[7] : Running
  8 : Moab.1[8] : Running
  9 : Moab.1[9] : Running
  10 : Moab.1[10] : Running
  11 : Moab.1[11] : Running
  12 : Moab.1[12] : Running
  13 : Moab.1[13] : Running
  14 : Moab.1[14] : Running
  15 : Moab.1[15] : Running
  16 : Moab.1[16] : Running
  17 : Moab.1[17] : Running
  18 : Moab.1[18] : Running
  19 : Moab.1[19] : Running
  20 : Moab.1[20] : Running
  Totals:
    Active:   20
    Idle:     0
    Complete: 0
```

You can also use showq. This displays the array master job with a count of how many sub-jobs are in each queue.

```
$ showq

active jobs------------------------
JOBID              USERNAME      STATE PROCS   REMAINING            STARTTIME

Moab.1(5)           aesplin    Running     5    00:52:41  Thu Jun 23 17:05:56
Moab.2(1)           aesplin    Running     1    00:53:41  Thu Jun 23 17:06:56

6 active jobs               6 of 6 processors in use by local jobs (100.00%)
1 of 1 nodes active      (100.00%)

eligible jobs----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME
```

```
Moab.2(4)           aesplin       Idle     4     1:00:00  Thu Jun 23 17:06:56

4 eligible jobs

blocked jobs------------------------
JOBID               USERNAME      STATE PROCS     WCLIMIT            QUEUETIME

Moab.2(1)           aesplin    Blocked     1     1:00:00  Thu Jun 23 17:06:56

1 blocked job

Total jobs:  11
```

`Moab.1` has five sub-jobs running. `Moab.2` has one sub-job running, four waiting to run, and one that is currently blocked.

Job Array Cancellation Policies

Job arrays can be canceled based on the success or failure of the first sub-job, the first success or failure of any sub-job, or if any sub-job exits with a specified exit code. The job array cancellation policies are:

Cancel Policy	Description	Exclusivity
CancelOnFirstFailure	Cancels the job array if the first sub-job (**JOBARRAYINDEX** = *1*) fails. `> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnFirstFailure`	Mutually exclusive
CancelOnFirstSuccess	Cancels the job array if the first sub-job (**JOBARRAYINDEX** = *1*) succeeds. `> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnFirstSuccess`	
CancelOnAnyFailure	Cancels the job array if any sub-job fails. `> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnAnyFailure`	
CancelOnAnySuccess	Cancels the job array if any sub-job succeeds. `> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnAnySuccess`	
CancelOnExitCode	Cancels the job array if any sub-job returns the specified exit code. `> msub -t myarray[1-1000%50] -l ...,flags=CancelOnExitCode:<error code list>` The syntax for the error code list are ranges specified with a dash and individual codes delimited by a plus (+) sign, such as: 1-4+9+15 Exit codes 1-387 are accepted.	

Up to two cancellation polices can be specified for an array and the two policies must be delimited by a colon (:). The two "first sub-job" policies are mutually exclusive, as are the three "any sub-job" policies. You can use either "first sub-job" policy with one of the "any sub-job" policies, as shown in this example:

```
> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnFirstFailure:CancelOnExitCode:3-7+11
```

Examples

Operations can be performed on individual jobs, a selection of jobs in a job array, or on the entire array.

Submitting Job Arrays

The syntax for submitting job arrays is: `msub -t [<jobname>]<indexlist>[%<limit>] arrayscript.sh`

The *<jobname>* and *<limit>* are optional. The jobname does not override the `jobID` Moab assigns to the array. When submitting an array with a jobname, Moab returns the `jobID`, which is the scheduler name followed by a unique ID.

For example, if the scheduler name in `moab.cfg` is *Moab* (`SCHEDCFG[Moab]`), submitting an array with a jobname responds like this:

```
> msub -t myarray[1-10] job.sh

Moab.6
```

To specify that only a certain number of sub-jobs in the array can run at a time, use the percent sign (`%`) delimiter. In this example, only five sub-jobs in the array can run at a time:

```
> msub -t myarray[1-1000]%5
```

To submit a specific set of array sub-jobs, use the comma delimiter in the array index list:

```
> msub -t myarray[1,2,3,4]
> msub -t myarray[1-5,7,10]
```

You can use the checkjob command on either the `jobID` or the `jobname` you specified.

```
> msub -t myarray[1-2] job.sh

Moab.10

$ checkjob -v myarray
  job Moab.10

AName: myarray
Job Array Info:
   Name: Moab.10
   1 : Moab.10[1] : Running
   2 : Moab.10[2] : Running

   Sub-jobs:           2
     Active:           2 ( 100.0% )
     Eligible:         0 ( 0.0% )
     Blocked:          0 ( 0.0% )
     Completed:        0 ( 0.0% )
```

```
State: Idle
Creds:  user:tuser1  group:tgroup1
WallTime:   00:00:00 of 99:23:59:59
SubmitTime: Thu Jun  2 16:37:17
   (Time Queued  Total: 00:00:33  Eligible: 00:00:00)

Total Requested Tasks: 1

Req[0]  TaskCount: 1  Partition: ALL
```

To submit a job with a step size, use a colon in the array range and specify how many jobs to step. In the example below, a step size of 2 is requested. The sub-jobs will be numbered according to the step size inside the index limit. The array master job name will be the same as explained above.

```
$ msub -t myarray[2-10:2] job.sh

job Moab.15

$ checkjob -v myarray #or you could use 'checkjob -v Moab.15'
job Moab.15

AName: myarray
Job Array Info:
   Name: Moab.15
   2 : Moab.15[2] : Running
   4 : Moab.15[4] : Running
   6 : Moab.15[6] : Running
   8 : Moab.15[8] : Running
   10 : Moab.15[10] : Running

   Sub-jobs:           5
     Active:           5 ( 100.0% )
     Eligible:         0 ( 0.0% )
     Blocked:          0 ( 0.0% )
     Completed:        0 ( 0.0% )

State: Idle
Creds:  user:tuser1  group:tgroup1
WallTime:   00:00:00 of 99:23:59:59
SubmitTime: Thu Jun  2 16:37:17
   (Time Queued  Total: 00:00:33  Eligible: 00:00:00)

Total Requested Tasks: 1

Req[0]  TaskCount: 1  Partition: ALL
```

Related topics

- Job Dependencies

11.0 General Node Administration

- **Node Location** on page 470
- **Node Attributes** on page 473
- **Node Specific Policies** on page 483
- **Managing Shared Cluster Resources (Floating Resources)** on page 484
- **Managing Node State** on page 488
- **Managing Consumable Generic Resources** on page 490
- **Enabling Generic Metrics** on page 492
- **Enabling Generic Events** on page 496

Overview

Moab has a very flexible and generalized definition of a node. This flexible definition, together with the fact that Moab must inter-operate with many resource managers of varying capacities, requires that Moab must possess a complete set of mechanisms for managing nodes that in some cases may be redundant with resource manager facilities.

Resource Manager Specified 'Opaque' Attributes

Many resource managers support the concept of opaque node attributes, allowing a site to assign arbitrary strings to a node. These strings are opaque in the sense that the resource manager passes them along to the scheduler without assigning any meaning to them. Nodes possessing these opaque attributes can then be requested by various jobs. Using certain Moab parameters, sites can assign a meaning within Moab to these opaque node attributes and extract specific node information. For example, setting the parameter FEATUREPROCSPEEDHEADER xps causes a node with the opaque string `xps950` to be assigned a processor speed of 950 MHz within Moab.

Scheduler Specified Default Node Attributes

Some default node attributes can be assigned on a rack or partition basis. In addition, many node attributes can be specified globally by configuring the *DEFAULT* node template using the NODECFG parameter (i.e., `NODECFG[DEFAULT] PROCSPEED=3200`). Unless explicitly specified otherwise, nodes inherit node attributes from the associated rack or partition or from the default node template. See the Partition Overview for more information.

Scheduler Specified Node Attributes

The **NODECFG** parameter also allows direct per-node specification of virtually all node attributes supported via other mechanisms and also provides a number of additional attributes not found elsewhere. For example, a site administrator may want to specify something like the following:

```
NODECFG[node031] MAXJOB=2 PROCSPEED=600 PARTITION=small
```

These approaches may be mixed and matched according to the site's local needs. Precedence for the approaches generally follows the order listed earlier in cases where conflicting node configuration information is specified through one or more mechanisms.

11.1 Node Location

Nodes can be assigned three types of location information based on partitions, racks, and queues.

- Partitions
- Racks
- Queues
  - TORQUE/OpenPBS Queue to Node Mapping
- Node Selection/Specification

Partitions

The first form of location assignment, the partition, allows nodes to be grouped according to physical resource constraints or policy needs. By default, jobs are not allowed to span more than one partition so partition boundaries are often valuable if an underlying network topology make certain resource allocations undesirable. Additionally, per-partition policies can be specified to grant control over how scheduling is handled on a partition by partition basis. See the Partition Overview for more information.

Racks

Rack-based location information is orthogonal to the partition based configuration and is mainly an organizational construct. In general rack based location usage, a node is assigned both a rack and a slot number. This approach has descended from the IBM SP2 organizational approach in which a rack can contain any number of slots but typically contains between 1 and 99. Using the rack and slot number combo, individual compute nodes can be grouped and displayed in a more ordered manner in certain Moab commands (i.e., showstate). Currently, rack information can only be specified directly by the system via the SDR interface on SP2/Loadleveler systems. In all other systems, this information must be specified using an information service or specified manually using the RACK, SLOT, and SIZE attributes of the NODECFG parameter.

Sites may arbitrarily assign nodes to racks and rack slots without impacting scheduling behavior. Neither rack numbers nor rack slot numbers need to be contiguous and their use is simply for convenience purposes in displaying and analyzing compute resources.

Example 11-1:

```
NODECFG[node024] RACK=1 SLOT=1
```

```
NODECFG[node025] RACK=1 SLOT=2
NODECFG[node026] RACK=2 SLOT=1 PARTITION=special
...
```

When specifying node and rack information, slot values must be in the range of 1 to 99, and racks must be in the range of 1 to 399.

Queues

Some resource managers allow queues (or classes) to be defined and then associated with a subset of available compute resources. With systems such as Loadleveler or PBSPro these queue to node mappings are automatically detected. On resource managers that do not provide this service, Moab provides alternative mechanisms for enabling this feature.

TORQUE/OpenPBS Queue to Node Mapping

Under TORQUE, queue to node mapping can be accomplished by using the qmgr command to set the queue acl_hosts parameter to the mapping host list 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.

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 host names. Each element will be interpreted as a regular expression.

```
SRCFG[basic]  HOSTLIST=cl37.icluster,ax45,ax46
...
```

Exact Lists

When Moab receives a list of nodes it will, by default, interpret each element as a regular expression. To disable this and have each element interpreted as a string node name, the `l:` can be used as in the following example:

```
> setres l:n00,n01,n02
```

Node Range

Node lists can be specified as one or more comma or whitespace delimited node ranges. Each node range can be based using either *<STARTINDEX>-<ENDINDEX>* or *<HEADER>*`[`*<STARTINDEX>-<ENDINDEX>*`]` format. To explicitly request a range, the node expression must be preceded with the string `r:` as in the following example:

```
> setres r:37-472,513,516-855
```

When you specify a *<HEADER>* for the range, note that it must only contain alphabetical characters. As always, the range must be numeric.

```
CLASSCFG[long] HOSTLIST=r:anc-b[37-472]
```

Only one expression is allowed with node ranges.

By default, Moab attempts to extract a node's node index assuming this information is built into the node's naming convention. If needed, this information can be explicitly specified in the Moab configuration file using NODECFG's **NODEINDEX** attribute, or it can be extracted from alternately formatted node IDs by specifying the NODEIDFORMAT parameter.

Node Regular Expression

Node lists may also be specified as one or more comma or whitespace delimited regular expressions. Each node regular expression must be specified in a format acceptable by the standard C regular expression libraries that allow support for wildcard and other special characters such as the following:

- `*` (asterisk)
- `.` (period)
- `[ ]` (left and right bracket)
- `^` (caret)
- `$` (dollar)

Node lists are by default interpreted as a regular expression but can also be explicitly requested with the string *x:* as in the following examples:

```
# select nodes cl30 thru cl55
SRCFG[basic]  HOSTLIST=x:cl[34],cl5[0-5]
...
```

```
# select nodes cl30 thru cl55
SRCFG[basic]  HOSTLIST=cl[34],cl5[0-5]
...
```

To control node selection search ordering, set the **OBJECTELIST** parameter to one of the following options: exact, range, regex, rangere, or rerange.

11.2 Node Attributes

- **Configurable Node Attributes** on page 473
- **Node Features/Node Properties** on page 482

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:

ACCESS on page 474
ARCH on page 474
CHARGERATE on page 474
COMMENT on page 474
ENABLEPROFILING on page 474
FEATURES on page 475
FLAGS on page 475
GRES on page 475
LOGLEVEL on page 476
MAXIOIN on page 476
MAXJOB on page 476
MAXJOBPERUSER on page 476
MAXPE on page 476
MAXPEPERJOB on page 476
MAXPROC on page 476
NETWORK on page 476
NODEINDEX on page 476
NODETYPE on page 477
OS on page 477
OSLIST on page 477
OVERCOMMIT on page 477
PARTITION on page 477
POWERPOLICY on page 477
PREEMPTMAXCPULOAD on page 478
PREEMPTMINMEMAVAIL on page 478
PREEMPTPOLICY on page 478
PRIORITY on page 478
PRIORITYF on page 479
PROCSPEED on page 479
PROVRM on page 479
RACK on page 479
RADISK on page 479
RCDISK on page 480
RCMEM on page 480
RCPROC on page 480
RCSWAP on page 481
SIZE on page 481
SLOT on page 481
SPEED on page 481
TRIGGER on page 481
VARIABLE on page 481
VMOCTHRESHOLD on page 482

Attribute	Description
ACCESS	Specifies the node access policy that can be one of *SHARED*, *SHAREDONLY*, *SINGLEJOB*, *SINGLETASK*, or *SINGLEUSER*. See Node Access Policies for more details. `NODECFG[node013] ACCESS=singlejob`
ARCH	Specifies the node's processor architecture. `NODECFG[node013] ARCH=opteron`

CHARGERATE	Allows a site to assign specific charging rates to the usage of particular resources. The **CHARGERATE** 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).
	This feature can only be used in conjunction with the AMCFG[] LOCALCOST flag which limits its use to cases where Moab calculates the full charge to be used by Moab Accounting Manager.
	`NODECFG[DEFAULT] CHARGERATE=1.0`
	`NODECFG[node003] CHARGERATE=1.5`
	`NODECFG[node022] CHARGERATE=2.5`

COMMENT	Allows an organization to annotate a node via the configuration file to indicate special information regarding this node to both users and administrators. The **COMMENT** value may be specified as a quote delimited string as shown in the example that follows. Comment information is visible using checknode, mdiag, Moab Cluster Manager, and Moab Access Portal. `NODECFG[node013] COMMENT="Login Node"`
ENABLEPROFILING	Allows an organization to track node state over time. This information is available using showstats -n. `NODECFG[DEFAULT] ENABLEPROFILING=TRUE`

Attribute	Description
FEATURES	Not all resource managers allow specification of opaque node features (also known as node properties). For these systems, the NODECFG parameter can be used to directly assign a list of node features to individual nodes. To set/overwrite a node's features, use `FEATURES=<X>`; to append node features, use `FEATURES+=<X>`. `NODECFG[node013] FEATURES+=gpfs,fastio` The total number of supported node features is limited as described in the Adjusting Default Limits section. If supported by the resource manager, the resource manager specific manner of requesting node features/properties within a job may be used. (Within TORQUE, use `qsub -l nodes=<NODECOUNT>:<NODEFEATURE>`.) However, if either not supported within the resource manager or if support is limited, the Moab feature resource manager extension may be used.
FLAGS	Specifies various flags that should be set on the given node. Node flags must be set using the mschedctl -m config command. Do not set node flags in the `moab.cfg` file. Flags set in `moab.cfg` may conflict with settings controlled automatically by resource managers, Moab Web Services, or Viewpoint. • *globalvars* - The node has variables that may be used by triggers. • *novmmigrations* - Excludes this hypervisor from VM auto-migrations. This means that VMs cannot automatically migrate to or from this hypervisor while this flag is set. `NODECFG[node1] FLAGS=NoVMMigrations` *To allow VMs to resume migrating, remove this flag using `mschedctl -m config 'NODECFG[node1] FLAGS-=NoVMMigrations'` or use a resource manager to unset the flag. Because both Moab and the RM report the novmmigration flag and the RM's setting always overrides the Moab setting, you cannot remove the flag via the Moab command when the RM is reporting it.*
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`

Attribute	Description
LOGLEVEL	Node specific loglevel allowing targeted 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`
	`...`

NETWORK	The ability to specify which networks are available to a given node is limited to only a few resource managers. Using the **NETWORK** attribute, administrators can establish this node to network connection directly through the scheduler. The NODECFG parameter allows this list to be specified in a comma-delimited list.
	`NODECFG[node024] NETWORK=GigE`
	`...`

NODEINDEX	The node's index. See Node Location for details.

Attribute	Description
NODETYPE	The **NODETYPE** attribute is most commonly used in conjunction with an allocation management system such as Moab Accounting Manager. 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.
OSLIST	This attribute specifies the list of operating systems the node can run. `NODECFG[compute002] OSLIST=linux,windows`
OVERCOMMIT	Specifies the high-water limit for over-allocation of processors or memory on a hypervisor. This setting is used to protect hypervisors from having too many VMs placed on them, regardless of the utilization level of those VMs. Possible attributes include `DISK`, `MEM`, `PROC`, and `SWAP`. Usage is `<attr>:<integer>`. `NODECFG[node012] OVERCOMMIT=PROC:2,MEM:4`
PARTITION	See Node Location for details.
POWERPOLICY	The **POWERPOLICY** can be set to *OnDemand* or *STATIC*. It defaults to *STATIC* if not set. If set to *STATIC*, Moab will never automatically change the power status of a node. If set to *OnDemand*, Moab will turn the machine off and on based on work-load and global settings. See Green Computing for further details.

Attribute	Description

PREEMPTMAXCPULOAD	If the node CPU load exceeds the specified value, any batch jobs running on the node are preempted using the preemption policy specified with the node's PREEMPTPOLICY attribute. If this attribute is not specified, the global default policy specified with PREEMPTPOLICY parameter is used. See Sharing Server Resources for further details.
	`NODECFG[node024] PRIORITY=-150 COMMENT="NFS Server Node"`
	`NODECFG[node024] PREEMPTPOLICY=CANCEL PREEMPTMAXCPULOAD=1.2`
	`...`

PREEMPTMINMEMAVAIL	If the available node memory drops below the specified value, any batch jobs running on the node are preempted using the preemption policy specified with the node's PREEMPTPOLICY attribute. If this attribute is not specified, the global default policy specified with PREEMPTPOLICY parameter is used. See Sharing Server Resources for further details.
	`NODECFG[node024] PRIORITY=-150 COMMENT="NFS Server Node"`
	`NODECFG[node024] PREEMPTPOLICY=CANCEL PREEMPTMINMEMAVAIL=256`
	`...`

PREEMPTPOLICY	If any node preemption policies are triggered (such as PREEMPTMAXCPULOAD or PREEMPTMINMEMAVAIL) any batch jobs running on the node are preempted using this preemption policy if specified. If not specified, the global default preemption policy specified with PREEMPTPOLICY parameter is used. See Sharing Server Resources for further details.
	`NODECFG[node024] PRIORITY=-150 COMMENT="NFS Server Node"`
	`NODECFG[node024] PREEMPTPOLICY=CANCEL PREEMPTMAXCPULOAD=1.2`
	`...`

PRIORITY	The **PRIORITY** attribute specifies the fixed node priority relative to other nodes. It is only used if NODEALLOCATIONPOLICY is set to *PRIORITY*. The default node priority is 0. A default cluster-wide node priority may be set by configuring the **PRIORITY** attribute of the *DEFAULT* node. See Priority Node Allocation for more details.
	`NODEALLOCATIONPOLICY  PRIORITY`
	`NODECFG[node024] PRIORITY=120`
	`...`

Attribute	Description

PRIORITYF	The **PRIORITYF** attribute specifies the function to use when calculating a node's allocation priority specific to a particular job. It is only used if NODEALLOCATIONPOLICY is set to *PRIORITY*. The default node priority function sets a node's priority exactly equal to the configured node priority. The priority function allows a site to indicate that various environmental considerations such as node load, reservation affinity, and ownership be taken into account as well using the following format:
	*<COEFFICIENT> * <ATTRIBUTE> [+ <COEFFICIENT> * <ATTRIBUTE>]...*
	<ATTRIBUTE> is an attribute from the table found in the Priority Node Allocation section.
	A default cluster-wide node priority function may be set by configuring the **PRIORITYF** attribute of the *DEFAULT* node. See Priority Node Allocation for more details.
	`NODEALLOCATIONPOLICY PRIORITY`
	`NODECFG[node024] PRIORITYF='APROCS + .01 * AMEM - 10 * JOBCOUNT'`
	`...`

PROCSPEED	Knowing a node's processor speed can help the scheduler improve intra-job efficiencies by allocating nodes of similar speeds together. This helps reduce losses due to poor internal job load balancing. Moab's Node Set scheduling policies allow a site to control processor speed based allocation behavior. Processor speed information is specified in MHz and can be indicated directly using NODECFG or through use of the FEATUREPROCSPEEDHEADER parameter.

PROVRM	Provisioning resource managers can be specified on a per node basis. This allows flexibility in mixed environments. If the node does not have a provisioning resource manager, the default provisioning resource manager will be used. The default is always the first one listed in `moab.cfg`.
	`RMCFG[prov] TYPE=NATIVE RESOURCETYPE=PROV`
	`RMCFG[prov] PROVDURATION=10:00`
	`RMCFG[prov] NODEMODIFYURL=exec://$HOME/tools/os.switch.pl`
	`...`
	`NODECFG[node024] PROVRM=prov`

RACK	The rack associated with the node's physical location. Valid values range from 1 to 400. See Node Location for details.
RADISK	Jobs can request a certain amount of disk space through the RM Extension String's **DDISK** parameter. When done this way, Moab can track the amount of disk space available for other jobs. To set the total amount of disk space available the **RADISK** parameter is used.

Attribute	Description
RCDISK	Jobs can request a certain amount of disk space (in MB) through the RM Extension String's **DDISK** parameter. When done this way, Moab can track the amount of disk space available for other jobs. The **RCDISK** attribute constrains the amount of disk reported by a resource manager while the **RADISK** attribute specifies the amount of disk available to jobs. If the resource manager does not report available disk, the **RADISK** attribute should be used.

RCMEM	Jobs can request a certain amount of real memory (RAM) in MB through the RM Extension String's **DMEM** parameter. When done this way, Moab can track the amount of memory available for other jobs. The **RCMEM** attribute constrains the amount of RAM reported by a resource manager while the **RAMEM** attribute specifies the amount of RAM available to jobs. If the resource manager does not report available disk, the **RAMEM** attribute should be used.
	Please note that memory reported by the resource manager will override the configured value unless a trailing caret (^) is used.
	`NODECFG[node024] RCMEM=2048`
	`...`
	If the resource manager does not report any memory, then Moab will assign node0242048 MB of memory.
	`NODECFG[node024] RCMEM=2048^`
	`...`
	Moab will assign 2048 MB of memory to node024 regardless of what the resource manager reports.

RCPROC	The **RCPROC** specifies the number of processors available on a compute node.
	`NODECFG[node024] RCPROC=8`
	`...`

Attribute	Description

RCSWAP	Jobs can request a certain amount of swap space in MB.
	RCSWAP works similarly to **RCMEM**. Setting **RCSWAP** on a node will set the swap but can be overridden by swap reported by the resource manager. If the trailing caret (^) is used, Moab will ignore the swap reported by the resource manager and use the configured amount.
	`NODECFG[node024] RCSWAP=2048`
	`...`
	If the resource manager does not report any memory, Moab will assign node0242048 MB of swap.
	`NODECFG[node024] RCSWAP=2048^`
	`...`
	Moab will assign 2048 MB of swap to node024 regardless of what the resource manager reports.

SIZE	The number of slots or size units consumed by the node. This value is used in graphically representing the cluster using showstate or Moab Cluster Manager. See Node Location for details. For display purposes, legal size values include *1*, *2*, *3*, *4*, *6*, *8*, *12*, and *16*.
	`NODECFG[node024] SIZE=2`
	`...`

SLOT	The first slot in the rack associated with the node's physical location. Valid values range from 1 to **MMAX_RACKSIZE** (default=*64*). See Node Location for details.
SPEED	Because today's processors have multiple cores and adjustable clock frequency, this feature has no meaning and will be deprecated.
TRIGGER	See Object Triggers for details.

VARIABLE	Variables associated with the given node, which can be used in job scheduling. See -l PREF.
	`NODECFG[node024] VARIABLE=var1`
	`...`

Attribute	Description
VMOCTHRESHOLD	Specifies the high-water threshold for utilization of resources on a server (i.e. processor and memory). This setting is used to protect hypervisors from becoming too highly utilized and thus negatively impacting the performance of VMs running on the hypervisor. Possible attributes include PROC and MEM. `NODECFG[node024] VMOCTHRESHOLD=PROC=2,MEM=2`

Node Features/Node Properties

A node feature (or node property) is an opaque string label that is associated with a compute node. Each compute node may have any number of node features assigned to it, and jobs may request allocation of nodes that have specific features assigned. Node features are labels and their association with a compute node is not conditional, meaning they cannot be consumed or exhausted.

Node features may be assigned by the resource manager, and this information may be imported by Moab or node features may be specified within Moab directly. As a convenience feature, certain node attributes can be specified via node features using the parameters listed in the following table:

PARAMETER	DESCRIPTION
FEATURENODETYPEHEADER	Set Node Type
FEATUREPARTITIONHEADER	Set Partition
FEATUREPROCSPEEDHEADER	Set Processor Speed
FEATURERACKHEADER	Set Rack
FEATURESLOTHEADER	Set Slot

Example 11-2:

```
FEATUREPARTITIONHEADER  par
FEATUREPROCSPEEDHEADER  cpu
```

Related topics

- 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

11.3 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] ...`.

Node Usage/Throttling Policies

MAXJOB

This policy constrains the number of total independent jobs a given node may run simultaneously. It can only be specified via the NODECFG parameter.

On Cray XT systems, use the NID (node id) instead of the node name. For more information, see Configuring the moab.cfg file.

MAXJOBPERUSER

Constrains the number of total independent jobs a given node may run simultaneously associated with any single user. It can only be specified via the NODECFG parameter.

MAXJOBPERGROUP

Constrains the number of total independent jobs a given node may run simultaneously associated with any single group. It can only be specified via the NODECFG parameter.

MAXLOAD

MAXLOAD constrains the CPU load the node will support as opposed to the number of jobs. This maximum load policy can also be applied system wide using the parameter NODEMAXLOAD.

MAXPE

This policy constrains the number of total dedicated processor-equivalents a given node may support simultaneously. It can only be specified via the **NODECFG** parameter.

MAXPROC

This policy constrains the number of total dedicated processors a given node may support simultaneously. It can only be specified via the **NODECFG** parameter.

MAXPROCPERUSER

This policy constrains the number of total processors a given node may have dedicated to any single user. It can only be specified via the **NODECFG** parameter.

MAXPROCPERGROUP

This policy constrains the number of total processors a given node may have dedicated to any single group. It can only be specified via the **NODECFG** parameter.

Node throttling policies are used strictly as constraints. If a node is defined as having a single processor or the NODEACCESSPOLICY is set to *SINGLETASK*, and a **MAXPROC** policy of 4 is specified, Moab will not run more than one task per node. A node's configured processors must be specified so that multiple jobs may run and then the **MAXJOB** policy will be effective. The number of configured processors per node is specified on a resource manager specific basis. PBS, for example, allows this to be adjusted by setting the number of virtual processors with the `np` parameter for each node in the PBS `nodes` file.

Example 11-3:

```
NODECFG[node024] MAXJOB=4 MAXJOBPERUSER=2
NODECFG[node025] MAXJOB=2
NODECFG[node026] MAXJOBPERUSER=1
NODECFG[DEFAULT] MAXLOAD=2.5
...
```

Node Access Policies

While most sites require only a single cluster wide node access policy (commonly set using NODEACCESSPOLICY), it is possible to specify this policy on a node by node basis using the **ACCESS** attributes of the NODECFG parameter. This attribute may be set to any of the valid node access policy values listed in the Node Access Policies section.

Example 11-4:

To set a global policy of *SINGLETASK* on all nodes except nodes 13 and 14, use the following:

```
# by default, enforce dedicated node access on all nodes
NODEACCESSPOLICY  SINGLETASK
# allow nodes 13 and 14 to be shared
NODECFG[node13]   ACCESS=SHARED
NODECFG[node14]   ACCESS=SHARED
```

Related topics

- mnodectl

11.4 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.

Shared Cluster Resource Overview

Shared cluster resources such as file systems, networks, and licenses can be managed through creating a pseudo-node. You can configure a pseudo-node via the NODECFG parameter much as a normal node would be but additional information is required to allow the scheduler to contact and synchronize state with the resource.

In the following example, a license manager is added as a cluster resource by defining the *GLOBAL* pseudo-node and specifying how the scheduler should query and modify its state.

```
NODECFG[GLOBAL] RMLIST=NATIVE
NODECFG[GLOBAL] QUERYCMD=/usr/local/bin/flquery.sh
NODECFG[GLOBAL] MODIFYCMD=/usr/local/bin/flmodify.sh
```

In some cases, pseudo-node resources may be very comparable to node-locked generic resources however there are a few fundamental differences which determine when one method of describing resources should be used over the other. The following table contrasts the two resource types.

Attribute	Pseudo-Node	Generic Resource
Node-Locked	No - Resources can be encapsulated as an independent node.	Yes - Must be associated with an existing compute node.
Requires exclusive batch system control over resource	No - Resources (such as file systems and licenses) may be consumed both inside and outside of batch system workload.	Yes - Resources must only be consumed by batch workload. Use outside of batch control results in loss of resource synchronization.
Allows scheduler level allocation of resources	Yes - If required, the scheduler can take external administrative action to allocate the resource to the job.	No - The scheduler can only maintain logical allocation information and cannot take any external action to allocate resources to the job.

Configuring Generic Consumable Floating Resources

Consumable floating resources are configured in the same way as node-locked generic resources with the exception of using the *GLOBAL* node instead of a particular node.

```
NODECFG[GLOBAL] GRES=tape:4,matlab:2
...
```

In this setup, four resources of type tape and 2 of type matlab are floating and available across all nodes.

Requesting Consumable Floating Resources

Floating resources are requested on a per task basis using native resource manager job submission methods or using the GRES resource manager extensions.

Configuring Cluster File Systems

Moab allows both the file space and bandwidth attributes or a cluster file system to be tracked, reserved, and scheduled. With this capability, a job or reservation may request a particular quantity of file space and a required amount of I/O bandwidth to this file system. While file system resources are managed as a cluster generic resource, they are specified using the **FS** attribute of the **NODECFG** parameter as in the following example:

```
NODECFG[GLOBAL] FS=PV1:10000@100,PV2:5000@100
...
```

In this example, PV1 defines a 10 GB file system with a maximum throughput of 100 MB/s while PV2 defines a 5 GB file system also possessing a maximum throughput of 100 MB/s.

A job may request cluster file system resources using the `fs` resource manager extension. For a TORQUE based system, the following could be used:

```
>qsub -l nodes=1,walltime=1:00:00 -W x=fs:10@50
```

Configuring Cluster Licenses

Jobs may request and reserve software licenses using native methods or using the GRES resource manager extension. If the cluster license manager does not support a query interface, license availability may be specified within Moab using the **GRES** attribute of the NODECFG parameter.

Example 11-5: Configure Moab to support four floating quickcalc and two floating matlab licenses.

```
NODECFG[GLOBAL] GRES=quickcalc:4,matlab:2
...
```

Example 11-6: Submit a TORQUE job requesting a node-locked or floating quickcalc license.

```
> qsub -l nodes=1,software=quickcalc,walltime=72000 testjob.cmd
```

Configuring Generic Resources as Features

Moab can be configured to treat generic resources as features in order to provide more control over server access. For instance, if a node is configured with a certain **GRES** and that **GRES** is turned off, jobs requesting the node will not run. To turn a GRES into a feature, set the **FEATUREGRES** attribute of GRESCFG to *TRUE* in the `moab.cfg` file.

```
GRESCFG[gres1] FEATUREGRES=TRUE
```

Moab now treats gres1 as a scheduler-wide feature rather than a normal generic resource.

Note that jobs are submitted normally using the same GRES syntax.

You can safely upgrade an existing cluster to use the feature while jobs are running.

Two methods exist for managing GRES features: via Moab commands and via the resource manager. Using Moab commands means that feature changes are not checkpointed; they do not remain in place

when Moab restarts. Using the resource manager causes changes to be reported by the RM, so any changes made before a Moab restart are still present after it.

These methods are mutually exclusive. Use one or the other, but do not mix methods.

Managing Feature GRES via Moab Commands

In the following example, *gres1* and *gres2* are configured in the `moab.cfg` file. *gres1* is not currently functioning correctly, so it is set to 0, turning the feature off. Values above 0 and non-specified values turn the feature on.

```
NODECFG[GLOBAL] GRES=gres1:0
NODECFG[GLOBAL] GRES=gres2:10000
GRESCFG[gres1] FEATUREGRES=TRUE
GRESCFG[gres2] FEATUREGRES=TRUE
```

Moab now treats gres1 and gres2 as features.

To verify that this is set up correctly, run mdiag -S -v. It returns the following:

```
> mdiag -S -v
...
  Scheduler FeatureGres: gres1:off,gres2:on
```

Once Moab has started, use mschedctl -m to modify whether the feature is turned on or off.

```
mschedctl -m sched featuregres:gres1=on

INFO: FeatureGRes 'gres1' turned on
```

You can verify that the feature turned on or off by once again running mdiag -S -v.

If Moab restarts, it will not checkpoint the state of these changed feature general resources. Instead, it will read the `moab.cfg` file to determine whether the feature GRES is on or off.

With feature GRES configured, jobs are submitted normally, requesting GRES type *gres1* and *gres2*. Moab ignores GRES counts and reads the feature simply as on or off.

```
> msub -l nodes=1,walltime=600,gres=gres1

1012
> checkjob 1012
job 1012

AName: STDIN
State: Running
.....
StartTime: Tue Jul 3 15:33:28
Feature GRes: gres1
Total Requested Tasks: 1
```

If you request a feature that is currently turned off, the state is not reported as `Running`, but as `Idle`. A message like the following returns:

```
BLOCK MSG: requested feature gres 'gres2' is off
```

Managing Feature GRES via the Resource Manager

You can automate the process of having a feature GRES turn on and off by setting up an external tool and configuring Moab to query the tool the same way that Moab queries a license manager. For example:

```
RMCFG[myRM] CLUSTERQUERYURL=file:///$HOME/tools/myRM.dat TYPE=NATIVE
RESOURCETYPE=LICENSE

GRESCFG[gres1] FEATUREGRES=TRUE
GRESCFG[gres2] FEATUREGRES=TRUE
```

> *LICENSE means that the RM does not contain any compute resources and that Moab should not attempt to use it to manage any jobs (start, cancel, submit, etc.).*

The `myRM.dat` file should contain something like the following:

```
GLOBAL state=Idle cres=gres1:0,gres2:10
```

External tools can easily update the file based on file system availability. Switching any of the feature GRES to 0 turns it off and switching it to a positive value turns it on. If you use this external mechanism, you do not need to use mschedctl -m to turn a feature GRES on or off. You also do not need to worry about whether Moab has checkpointed the information or not, since the information is provided by the RM and not by any external commands.

Related topics

- Managing Resources Directly with the Native Interface

11.5 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

Node State Definitions

State	Definition
Down	Node is either not reporting status, is reporting status but failures are detected, or is reporting status but has been marked down by an administrator.
Idle	Node is reporting status, currently is not executing any workload, and is ready to accept additional workload.

State	Definition
Busy	Node is reporting status, currently is executing workload, and cannot accept additional workload due to load.
Running	Node is reporting status, currently is executing workload, and can accept additional workload.
Drained	Node is reporting status, currently is not executing workload, and cannot accept additional workload due to administrative action.
Draining	Node is reporting status, currently is executing workload, and cannot accept additional workload due to administrative action.

Specifying Node States within Native Resource Managers

Native resource managers can report node state implicitly and explicitly, using **NODESTATE**, **LOAD**, and other attributes. See Managing Resources Directly with the Native Interface for more information.

Moab Based Node State Adjustment

Node state can be adjusted based on reported processor, memory, or other load factors. It can also be adjusted based on reports of one or more resource managers in a multi-resource manager configuration. Also, both generic events and generic metrics can be used to adjust node state.

- TORQUE health scripts (allow compute nodes to detect and report site specific failures).

Adjusting Scheduling Behavior Based on Reported Node State

Based on reported node state, Moab can support various policies to make better use of available resources.

Down State

- JOBACTIONONNODEFAILURE parameter (cancel/requeue jobs if allocated nodes fail).
- Triggers (take specified action if failure is detected).

Related topics

- Managing Resources Directly with the Native Interface
- License Management
- Adjusting Node Availability
- NODEMAXLOAD parameter

11.6 Managing Consumable Generic Resources

- Configuring Node-Locked Consumable Generic Resources
  - Requesting Consumable Generic Resources
- Managing Generic Resource Race Conditions

Each time a job is allocated to a compute node, it consumes one or more types of resources. Standard resources such as CPU, memory, disk, network adapter bandwidth, and swap are automatically tracked and consumed by Moab. However, in many cases, additional resources may be provided by nodes and consumed by jobs that must be tracked. The purpose of this tracking may include accounting, billing, or the prevention of resource over-subscription. Generic consumable resources may be used to manage software licenses, I/O usage, bandwidth, application connections, or any other aspect of the larger compute environment; they may be associated with compute nodes, networks, storage systems, or other real or virtual resources.

These additional resources can be managed within Moab by defining one or more generic resources. The first step in defining a generic resource involves naming the resource. Generic resource availability can then be associated with various compute nodes and generic resource usage requirements can be associated with jobs.

Differences between Node Features and Consumable Resources

A node feature (or node property) is an opaque string label that is associated with a compute node. Each compute node may have any number of node features assigned to it and jobs may request allocation of nodes that have specific features assigned. Node features are labels and their association with a compute node is not conditional, meaning they cannot be consumed or exhausted.

Configuring Node-locked Consumable Generic Resources

Consumable generic resources are supported within Moab using either direct configuration or resource manager auto-detect (as when using TORQUE and accelerator hardware). For direct configuration, node-locked consumable generic resources (or generic resources) are specified using the **NODECFG** parameter's **GRES** attribute. This attribute is specified using the format *<ATTR>*:*<COUNT>* as in the following example:

```
NODECFG[titan001] GRES=tape:4
NODECFG[login32]  GRES=matlab:2,prime:4
NODECFG[login33]  GRES=matlab:2
...
```

By default, Moab supports up to 128 independent generic resource types.

Requesting Consumable Generic Resources

Generic resources can be requested on a per task or per job basis using the **GRES** resource manager extension. If the generic resource is located on a compute node, requests are by default interpreted as a

per task request. If the generic resource is located on a shared, cluster-level resource (such as a network or storage system), then the request defaults to a per job interpretation.

Generic resources are specified per task, not per node. When you submit a job, each processor becomes a task. For example, a job asking for `nodes=3:ppn=4,gres=test:5` asks for 60 gres of type test ((3*4 processors)*5).

If using TORQUE, the GRES or software resource can be requested as in the following examples:

Example 11-7: Per Task Requests

```
NODECFG[compute001] GRES=dvd:2 SPEED=2200
NODECFG[compute002] GRES=dvd:2 SPEED=2200
NODECFG[compute003] GRES=dvd:2 SPEED=2200
NODECFG[compute004] GRES=dvd:2 SPEED=2200
NODECFG[compute005] SPEED=2200
NODECFG[compute006] SPEED=2200
NODECFG[compute007] SPEED=2200
NODECFG[compute008] SPEED=2200
```

```
# submit job which will allocate only from nodes 1 through 4 requesting one dvd per
task
> qsub -l nodes=2,walltime=100,gres=dvd job.cmd
```

In this example, Moab determines that compute nodes exist that possess the requested generic resource. A compute node is a node object that possesses processors on which compute jobs actually execute. License server, network, and storage resources are typically represented by non-compute nodes. Because compute nodes exist with the requested generic resource, Moab interprets this job as requesting two compute nodes each of which must also possess a DVD generic resource.

Example 11-8: Per Job Requests

```
NODECFG[network] PARTITION=shared GRES=bandwidth:2000000
```

```
# submit job which will allocate 2 nodes and 10000 units of network bandwidth
> qsub -l nodes=2,walltime=100,gres=bandwidth:10000 job.cmd
```

In this example, Moab determines that there exist no compute nodes that also possess the generic resource bandwidth so this job is translated into a multiple-requirement—multi-req—job. Moab creates a job that has a requirement for two compute nodes and a second requirement for 10000 bandwidth generic resources. Because this is a multi-req job, Moab knows that it can locate these needed resources separately.

Using Generic Resource Requests in Conjunction with other Constraints

Jobs can explicitly specify generic resource constraints. However, if a job also specifies a host list, the host list constraint overrides the generic resource constraint if the request is for per task allocation. In the Per Task Requests example, if the job also specified a host list, the *DVD* request is ignored.

Requesting Resources with No Generic Resources

In some cases, it is valuable to allocate nodes that currently have no generic resources available. This can be done using the special value *none* as in the following example:

```
> qsub -l nodes=2,walltime=100,gres=none job.cmd
```

In this case, the job only allocates compute nodes that have no generic resources associated with them.

Requesting Generic Resources Automatically within a Queue/Class

Generic resource constraints can be assigned to a queue or class and inherited by any jobs that do not have a **gres** request. This allows targeting of specific resources, automation of co-allocation requests, and other uses. To enable this, use the DEFAULT.GRES attribute of the CLASSCFG parameter as in the following example:

```
CLASSCFG[viz] DEFAULT.GRES=graphics:2
```

For each node requested by a *viz* job, also request two graphics cards.

Managing Generic Resource Race Conditions

A software license race condition "window of opportunity" opens when Moab checks a license server for sufficient available licenses and closes when the user's software actually checks out the software licenses. The time between these two events can be seconds to many minutes depending on overhead factors such as node OS provisioning, job startup, licensed software startup, and so forth.

During this window, another Moab-scheduled job or a user or job external to the cluster or cloud can obtain enough software licenses that by the time the job attempts to obtain its software licenses, there are an insufficient quantity of available licenses. In such cases a job will sit and wait for the license, and while it waits it occupies but does not use resources that another job could have used. Use the **STARTDELAY** parameter to prevent such a situation.

```
GRESCFG[<license>] STARTDELAY=<window_of_opportunity>
```

With the **STARTDELAY** parameter enabled (on a per generic resource basis) Moab blocks any idle jobs requesting the same generic resource from starting until the *<window_of_opportunity>* passes. The window is defined by the customer on a per generic resource basis.

Related topics

- GRESCFG parameter
- Generic Metrics
- Generic Events
- General Node Attributes
- Floating Generic Resources
- Per Class Assignment of Generic Resource Consumption
- mnodectl -m command to dynamically modify node resources
- Favoring Jobs Based On Generic Resource Requirements

11.7 Enabling Generic Metrics

- Configuring Generic Metrics
- Example Generic Metric Usage

Moab allows organizations to enable generic performance metrics. These metrics allow decisions to be made and reports to be generated based on site specific environmental factors. This increases Moab's awareness of what is occurring within a given cluster environment, and allows arbitrary information to be associated with resources and the workload within the cluster. Uses of these metrics are widespread and can cover anything from tracking node temperature, to memory faults, to application effectiveness.

- Execute triggers when specified thresholds are reached
- Modify node allocation affinity for specific jobs
- Initiate automated notifications when thresholds are reached
- Display current, average, maximum, and minimum metrics values in reports and charts within Moab Cluster Manager

Configuring Generic Metrics

A new generic metric is automatically created and tracked at the server level if it is reported by either a node or a job.

To associate a generic metric with a job or node, a native resource manager must be set up and the GMETRIC attribute must be specified. For example, to associate a generic metric of *temp* with each node in a TORQUE cluster, the following could be reported by a native resource manager:

```
# temperature output
node001 GMETRIC[temp]=113
node002 GMETRIC[temp]=107
node003 GMETRIC[temp]=83
node004 GMETRIC[temp]=85
...
```

Generic metrics are tracked as floating point values allowing virtually any number to be reported.

In the preceding example, the new metric, *temp*, can now be used to monitor system usage and performance or to allow the scheduler to take action should certain thresholds be reached. Some uses include the following:

- Executing triggers based on generic metric thresholds
- Adjust a node's availability for accepting additional workload
- Adjust a node's allocation priority
- Initiate administrator notification of current, minimum, maximum, or average generic metric values
- Use metrics to report resource and job performance
- Use metrics to report resource and job failures
- Using job profiles to allow Moab to learn which resources best run which applications
- Tracking effective application efficiency to identify resource brown 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 commands.

Example Generic Metric Usage

As an example, consider a cluster with two primary purposes for generic metrics. The first purpose is to track and adjust scheduling behavior based on node temperature to mitigate overheating nodes. The second purpose is to track and charge for utilization of a locally developed data staging service.

The first step in enabling a generic metric is to create probes to monitor and report this information. Depending on the environment, this information may be distributed or centralized. In the case of temperature monitoring, this information is often centralized by a hardware monitoring service and available via command line or an API. If monitoring a locally developed data staging service, this information may need to be collected from multiple remote nodes and aggregated to a central location. The following are popular freely available monitoring tools:

Tool	Link
BigBrother	http://www.bb4.org
Ganglia	http://ganglia.sourceforge.net
Monit	http://www.tildeslash.com/monit
Nagios	http://www.nagios.org

Once the needed probes are in place, a native resource manager interface must be created to report this information to Moab. Creating a native resource manager interface should be very simple, and in most cases a script similar to those found in the `$TOOLSDIR`(`$PREFIX/tools`) directory can be used as a template. For this example, we will assume centralized information and will use the RM script that follows:

```
#!/usr/bin/perl
# 'hwctl outputs information in format '<NODEID> <TEMP>'
open(TQUERY,"/usr/sbin/hwctl -q temp |");
while (<TQUERY>)
  {
  my $nodeid,$temp = split /\w+/;
```

```
  $dstage=GetDSUsage($nodeid);
  print "$nodeid GMETRIC[temp]=$temp GMETRIC[dstage]=$dstage
";
  }
```

With the script complete, the next step is to integrate this information into Moab. This is accomplished with the following configuration line:

```
RMCFG[local] TYPE=NATIVE CLUSTERQUERYURL=file://$TOOLSDIR/node.query.local.pl
...
```

Moab can now be recycled and temperature and data staging usage information will be integrated into Moab compute node reports.

If the checknode command is run, output similar to the following is reported:

```
> checknode cluster013
...
Generic Metrics:  temp=113.2,dstage=23748
...
```

Moab Cluster Manager reports full current and historical generic metric information in its visual cluster overview screen.

The next step in configuring Moab is to inform Moab to take certain actions based on the new information it is tracking. For this example, there are two purposes. The first purpose is to get jobs to avoid hot nodes when possible. This is accomplished using the **GMETRIC** attribute of the Node Allocation Priority function as in the following example:

```
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT] PRIORITYF=PRIORITY-10*GMETRIC[temp]
...
```

This simple priority function reduces the priority of the hottest nodes making such less likely to be allocated. See Node Allocation Priority Factors for a complete list of available priority factors.

The example cluster is also interested in notifying administrators if the temperature of a given node ever exceeds a critical threshold. This is accomplished using a trigger. The following line will send email to administrators any time the temperature of a node exceeds 120 degrees.

```
NODECFG[DEFAULT] TRIGGER=atype=mail,etype=threshold,threshold=gmetric[temp]
>120,action='warning: node $OID temp high'
...
```

Related topics

- Simulation Overview
- Generic Consumable Resources
- Object Variables
- Generic Event Counters

11.8 Enabling Generic Events

- Configuring Generic Events
  - Action Types
  - Named Events
  - Generic Metric (GMetric) Events
- Reporting Generic Events
  - Using Generic Events for VM Detection
- Generic Events Attributes
- 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

Configuring Generic Events

Generic events are defined in the `moab.cfg` file and have several different configuration options. The only required option is **action**.

The full list of configurable options for generic events is in the following table:

Attribute	Description
ACTION	Comma-delimited list of actions to be processed when a new event is received.
ECOUNT	Number of events that must occur before launching action. Action will be launched each *<ECOUNT>* event if rearm is set.
REARM	Minimum time between events specified in `[[[DD:]HH:]MM:]SS` format.
SEVERITY	An arbitrary severity level from 1 through 4, inclusive. SEVERITY appears in the output of `mdiag -n -v -v --xml`. The severity level will not be used for any other purpose.

Action Types

The impact of the event is controlled using the **ACTION** attribute of the GEVENTCFG parameter. The **ACTION** attribute is comma-delimited and may include any combination of the actions in the following table:

Value	Description
DISABLE [:<OTYPE>:<OID>]	Marks event object (or specified object) down until event report is cleared.
EXECUTE	Executes a script at the provided path. The value of *EXECUTE* is not contained in quotation marks. Arguments are allowed at the end of the path and are separated by question marks (?). Trigger variables (such as *$OID*) are allowed.
NOTIFY	Notifies administrators of the event occurrence.

Value	Description
OBJECTXMLSTDIN	If the *EXECUTE* action type is also specified, this flag passes an XML description of the firing gevent to the script.
OFF	Powers off node or resource.
ON	Powers on node or resource.
PREEMPT [:<POLICY>]	Preempts workload associated with object (valid for node, job, reservation, partition, resource manager, user, group, account, class, QoS, and cluster objects).
RECORD	Records events to the event log. The record action causes a line to be added to the event log regardless of whether or not RECORDEVENTLIST includes GEVENT.
RESERVE [:<DURATION>]	Reserves node for specified duration (default: 24 hours).
RESET	Resets object (valid for nodes - causes reboot).
SIGNAL[:<SIGNO>]	Sends signal to associated jobs or services (valid for node, job, reservation, partition, resource manager, user, group, account, class, QoS, and cluster objects).

This is an example of using `objectxmlstdin` with a gevent:

```
<gevent name="bob" statuscode="0" time="1320334763">Testing</gevent>
```

Named Events

In general, generic events are named, with the exception of those based on generic metrics. Names are used primarily to differentiate between different events and do not have any intrinsic meaning to Moab. It is suggested that the administrator choose names that denote specific meanings within the organization.

Example 11-9:

```
# Note: cpu failures require admin attention, create maintenance reservation
GEVENTCFG[cpufail] action=notify,record,disable,reserve rearm=01:00:00# Note: power
failures are transient, minimize future use
GEVENTCFG[powerfail] action=notify,record, rearm=00:05:00
# Note: fs full can be automatically fixed
GEVENTCFG[fsfull] action=notify,execute:/home/jason/MyPython/cleartmp.py?$OID?nodefix
# Note: memory errors can cause invalid job results, clear node immediately
GEVENTCFG[badmem] action=notify,record,preempt,disable,reserve
```

Generic Metric (GMetric) Events

GMetric events are generic events based on generic metrics. They are used for executing an action when a generic metric passes a defined threshold. Unlike named events, GMetric events are not named and use

the following format:
GEVENTCFG[GMETRIC<COMPARISON>VALUE] ACTION=...

Example 11-10:

```
GEVENTCFG[cputemp>150] action=off
```

This form of generic events uses the GMetric name, as returned by a **GMETRIC** attribute in a native Resource Manager interface.

Only one generic event may be specified for any given generic metric.

Valid comparative operators are shows in the following table:

Type	Comparison	Notes
>	greater than	Numeric values only
> =	greater than or equal to	Numeric values only
= =	equal to	Numeric values only
<	less than	Numeric values only
< =	less than or equal to	Numeric values only
< >	not equal	Numeric values only

Reporting Generic Events

Unlike generic metrics, generic events can be optionally configured at the global level to adjust rearm policies, and other behaviors. In all cases, this is accomplished using the GEVENTCFG parameter.

To report an event associated with a job or node, use the native Resource Manager interface or the mjobctl or mnodectl commands. You can report generic events on the scheduler with the mschedctl command.

If using the native Resource Manager interface, use the GEVENT attribute as in the following example:

```
node001 GEVENT[hitemp]='temperature exceeds 150 degrees'
node017 GEVENT[fullfs]='/var/tmp is full'
```

The time at which the event occurred can be passed to Moab to prevent multiple processing of the same event. This is accomplished by specifying the event type in the format *<GEVENTID>*[:*<EVENTTIME>*] as in what follows:

```
node001 GEVENT[hitemp:1130325993]='temperature exceeds 150 degrees'
node017 GEVENT[fullfs:1130325142]='/var/tmp is full'
```

Using Generic Events for VM Detection

To enable Moab to detect a virtual machine (VM) reported by a generic event, do the following:

1. Set up your resource manager to detect virtual machine creation and to submit a generic event to Moab.
2. Configure `moab.cfg` to recognize a generic event.

```
GEVENTCFG[NewVM] ACTION=execute:/opt/moab/AddVM.py,OBJECTXMLSTDIN
```

3. Report the event.

```
> mschedctl -c gevent -n NewVM -m "VM=newVMName"
```

With the ObjectXMLStdin action set, Moab sends an XML description of the generic event to the script, so the message passes through.

The following sample Perl script submits a VMTracking job for the new VM:

```
#!/usr/bin/perl

# in moab.cfg: GEVENTCFG[NewVM] ACTION=execute:$TOOLSDIR/newvm_event.pl,OBJECTXMLSTDIN
# trigger gevent with: mschedctl -c gevent -n NewVM -m "VM=TestVM1"
# input to this script: <gevent name="NewVM" statuscode="0"
time="1318500261">VM=TestVM1</gevent>

use strict;

my $vmidVarName = "preVMID";
my $vmTemplate = "existingVM";
my $vmOwner = "operator";

$ENV{MOABHOMEDIR} = '/opt/moab';

my $xml = join "", <STDIN>;
my ($vmid) = ($xml =~ m/VM=([^\<]+)\</);
if ( defined $vmid )
{
       my $cmd = qq| $ENV{MOABHOMEDIR}/bin/mvmctl -q $vmid --xml |;
       my $vmxml = `$cmd`;
       my ($hv, $os, $proc, $disk, $mem) = (undef, undef, undef, undef, undef);
       ($hv) = ($vmxml =~ m/CONTAINERNODE="([^"]+)"/);
       ($os) = ($vmxml =~ m/OS="([^"]+)"/);
       ($proc) = ($vmxml =~ m/RCPROC="([^"]+)"/);
       ($mem) = ($vmxml =~ m/RCMEM="([^"]+)"/);
       ($disk) = ($vmxml =~ m/RCDISK="([^"]+)"/);
       die "Error parsing VM XML. Invalid VMID $vmid or $hv || $os || $proc || $mem ||
$disk?
```

```
"
                if ( ! defined $hv || !defined $os || !defined $proc || !defined $mem || !defined
$disk );

        $cmd = qq| $ENV{MOABHOMEDIR}/bin/msub -l
hostlist=$hv,os=$os,nodes=1:ppn=$proc,mem=$mem,file=$disk,template=$vmTemplate,VAR=$vm
idVarName=$vmid --proxy=$vmOwner /dev/null |;
        my $msubout = `$cmd`;
        die "Error executing msub. Output is:
$msubout
" if ( $? );
} else {
        die "Error parsing VMID from GEVENT message
";
}
```

Generic Events Attributes

Each node will record the following about reported generic events:

- status - is event active
- message - human readable message associated with event
- count - number of event incidences reported since statistics were cleared
- time - time of most recent event

Each event can be individually cleared, annotated, or deleted by cluster administrators using a mnodectl command.

Generic events are only available in Moab 4.5.0 and later.

Manually Creating Generic Events

Generic events may be manually created on a physical node or VM.

To add GEVENT `event` with message "`hello`" to `node02`, do the following:

```
> mnodectl -m gevent=event:"hello" node02
```

To add GEVENT `event` with message "`hello`" to `myvm`, do the following:

```
> mvmctl -m gevent=event:"hello" myvm
```

Related topics

- Simulation Overview
- Generic Consumable Resources
- Object Variables
- Generic Event Counters

12.0 Resource Managers and Interfaces

- **Resource Manager Overview** on page 504
- **Resource Manager Configuration** on page 507
- **Resource Manager Extensions** on page 537
- **Adding New Resource Manager Interfaces** on page 565
- **Managing Resources Directly with the Native Interface** on page 566
- **Utilizing Multiple Resource Managers** on page 576
- **License Management** on page 578
- **Resource Provisioning** on page 580
- **Resource Manager Translation** on page 584

Moab provides a powerful resource management interface that enables significant flexibility in how resources and workloads are managed. Highlights of this interface are listed in what follows:

Highlight	Description
Support for Multiple Standard Resource Manager Interface Protocols	Manage cluster resources and workloads via PBS, Loadleveler, SGE, LSF, or BProc based resource managers.
Support for Generic Resource Manager Interfaces	Manage cluster resources securely via locally developed or open source projects using simple flat text interfaces or XML over HTTP.
Support for Multiple Simultaneous Resource Managers	Integrate resource and workload streams from multiple independent sources reporting disjoint sets of resources.

Highlight	Description
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.)

12.1 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.

- Scheduler/Resource Manager Interactions
  - Resource Manager Commands
  - Resource Manager Flow
- Resource Manager Specific Details (Limitations/Special Features)
- Synchronizing Conflicting Information
- Evaluating Resource Manager Availability and Performance

Scheduler/Resource Manager Interactions

Moab interacts with all resource managers using a common set of commands and objects. Each resource manager interfaces, obtains, and translates Moab concepts regarding workload and resources into native resource manager objects, attributes, and commands.

Information on creating a new scheduler resource manager interface can be found in the Adding New Resource Manager Interfaces section.

Resource Manager Commands

For many environments, Moab interaction with the resource manager is limited to the following objects and functions:

Object	Function	Details
Job	Query	Collect detailed state, requirement, and utilization information about jobs
	Modify	Change job state and/or attributes
	Start	Execute a job on a specified set of resources
	Cancel	Cancel an existing job
	Preempt/Resume	Suspend, resume, checkpoint, restart, or requeue a job
Node	Query	Collect detailed state, configuration, and utilization information about compute resources
	Modify	Change node state and/or attributes
Queue	Query	Collect detailed policy and configuration information from the resource manager

Using these functions, Moab is able to fully manage workload, resources, and cluster policies. More detailed information about resource manager specific capabilities and limitations for each of these functions can be found in the individual resource manager overviews. (LL, PBS, LSF, SGE, BProc, or WIKI).

Beyond these base functions, other commands exist to support advanced features such as provisioning and cluster level resource management.

Resource Manager Flow

In general, Moab interacts with resource managers in a sequence of steps each scheduling iteration. These steps are outlined in what follows:

1. load global resource information
2. load node specific information (optional)

3. load job information
4. load queue/policy information (optional)
5. cancel/preempt/modify jobs according to cluster policies
6. start jobs in accordance with available resources and policy constraints
7. handle user commands

Typically, each step completes before the next step is started. However, with current systems, size and complexity mandate a more advanced parallel approach providing benefits in the areas of reliability, concurrency, and responsiveness.

Resource Manager Specific Details (Limitations/Special Features)

- TORQUE
  - TORQUE Homepage
- SLURM/Wiki
  - SLURM Integration Guide
  - Wiki Overview

Synchronizing Conflicting Information

Moab does not trust resource manager information. Node, job, and policy information is reloaded on each iteration and discrepancies are detected. Synchronization issues and allocation conflicts are logged and handled where possible. To assist sites in minimizing stale information and conflicts, a number of policies and parameters are available.

- Node State Synchronization Policies (see **NODESYNCTIME** on page 882)
- Stale Data Purging (see **JOBPURGETIME** on page 860)
- Thread Management (preventing resource manager failures from affecting scheduler operation)
- Resource Manager Poll Interval (see **RMPOLLINTERVAL** on page 907)
- Node Query Refresh Rate (see **NODEPOLLFREQUENCY** on page 879)

Evaluating Resource Manager Availability and Performance

Each resource manager is individually tracked and evaluated by Moab. Using the **mdiag -R** on page 125 command, a site can determine how a resource manager is configured, how heavily it is loaded, what failures, if any, have occurred in the recent past, and how responsive it is to requests.

Related topics

- Resource Manager Configuration
- Resource Manager Extensions

12.2 Resource Manager Configuration

- Defining and Configuring Resource Manager Interfaces
  - Resource Manager Attributes
- Resource Manager Configuration Details
  - Resource Manager Types
  - Resource Manager Name
  - Resource Manager Location
  - Resource Manager Flags
  - Other Attributes
- Scheduler/Resource Manager Interactions

Defining and Configuring Resource Manager Interfaces

Moab resource manager interfaces are defined using the **RMCFG** on page 906 parameter. This parameter allows specification of key aspects of the interface. In most cases, only the **TYPE** attribute needs to be specified and Moab determines the needed defaults required to activate and use the selected interface. In the following example, an interface to a Loadleveler resource manager is defined.

```
RMCFG[orion] TYPE=LL...
```

Note that the resource manager is given a label of *orion*. This label can be any arbitrary site-selected string and is for local usage only. For sites with multiple active resource managers, the labels can be used to distinguish between them for resource manager specific queries and commands.

Resource Manager Attributes

The following table lists the possible resource manager attributes that can be configured.

ADMINEXEC on page 508
AUTHTYPE on page 509
BANDWIDTH on page 509
CHECKPOINTSIG on page 509
CHECKPOINTTIMEOUT on page 510
CLIENT on page 510
CLUSTERQUERYURL on page 511
CONFIGFILE on page 511
DATARM on page 511
DEFAULTCLASS on page 512
DEFAULTHIGHSPEEDADAPTER on page 512
DESCRIPTION on page 512
ENV on page 513
EPORT on page 513
FAILTIME on page 513
FLAGS on page 514
FNLIST on page 514
HOST on page 514
IGNHNODES on page 515
JOBCANCELURL on page 515
JOBEXTENDDURATION on page 515
JOBIDFORMAT on page 516
JOBMODIFYURL on page 516
JOBRSVRECREATE on page 517
JOBSTARTURL on page 517
JOBSUBMITURL on page 518
JOBSUSPENDURL on page 518
JOBVALIDATEURL on page 518
MAXDSOP on page 518
MAXITERATIONFAILURECOUNT on page 519
MAXJOBPERMINUTE on page 519
MAXJOBS on page 519
MINETIME on page 520
NMPORT on page 520
NODEFAILURERSVPROFILE on page 521
NODESTATEPOLICY on page 521
OMAP on page 521
PORT on page 522
PROVDURATION on page 522
PTYSTRING on page 522
RESOURCECREATEURL on page 523
RESOURCETYPE on page 523
RMSTARTURL on page 524
RMSTOPURL on page 524
SBINDIR on page 524
SLURMFLAGS on page 525
SOFTTERMSIG on page 525
STAGETHRESHOLD on page 526
STARTCMD on page 526
SUBMITCMD on page 527
SUBMITPOLICY on page 527
SUSPENDSIG on page 527
SYNCJOBID on page 528
SYSTEMMODIFYURL on page 528
SYSTEMQUERYURL on page 528
TARGETUSAGE on page 529
TIMEOUT on page 529
TRIGGER on page 529
TYPE on page 530
USEVNODES on page 530
VARIABLES on page 530
VERSION on page 531
VMOWNERRM on page 531
WORKLOADQUERYURL on page 531

ADMINEXEC

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	`RMCFG[base] ADMINEXEC=jobsubmit` *Moab will not use the user's UID and GID for executing the **JOBSUBMITURL**.*

AUTHTYPE	
Format	One of *CHECKSUM*, *OTHER*, *PKI*, *SECUREPORT*, or *NONE*.
Default	*CHECKSUM*
Description	Specifies the security protocol to be used in scheduler-resource manager communication. Only valid with WIKI based interfaces.
Example	`RMCFG[base] AUTHTYPE=CHECKSUM` *Moab requires a secret key-based checksum associated with each resource manager message.*

BANDWIDTH	
Format:	*<FLOAT>*`[{M|G|T}]`
Default:	*-1* (unlimited)
Description:	Specifies the maximum deliverable bandwidth between the Moab server and the resource manager for staging jobs and data. Bandwidth is specified in units per second and defaults to a unit of MB/s. If a unit modifier is specified, the value is interpreted accordingly (M - megabytes/sec, G - gigabytes/sec, T - terabytes/sec).
Example:	`RMCFG[base] BANDWIDTH=340G` *Moab will reserve up to 340 GB of network bandwidth when scheduling job and data staging operations to and from this resource manager.*

CHECKPOINTSIG	
Format	One of *suspend*, *<INTEGER>*, or *SIG<X>*
Description	Specifies what signal to send the resource manager when a job is checkpointed (See Checkpoint Overview.).

CHECKPOINTSIG

Example	`RMCFG[base] CHECKPOINTSIG=SIGKILL` *Moab routes the signal SIGKILL through the resource manager to the job when a job is checkpointed.*

CHECKPOINTTIMEOUT

Format	`[[[DD:]HH:]MM:]SS`
Default	*0* (no timeout)
Description	Specifies how long Moab waits for a job to checkpoint before canceling it. If set to *0*, Moab does not cancel the job if it fails to checkpoint (See Checkpoint Overview.).
Example	`RMCFG[base] CHECKPOINTTIMEOUT=5:00` *Moab cancels any job that has not exited 5 minutes after receiving a checkpoint request.*

CLIENT

Format	*<PEER>*
Default	Use name of resource manager for peer client lookup
Description	If specified, the resource manager will use the peer value to authenticate remote connections. (See configuring peers). If not specified, the resource manager will search for a **CLIENTCFG[<X>]** on page 811 entry of `RM:`*<RMNAME>*in the `moab-private.cfg` file.
Example	`RMCFG[clusterBI] CLIENT=clusterB` Moab will look up and use information for peer *clusterB* when authenticating the *clusterBI* resource manager.

CLUSTERQUERYURL

Format	[*file://<path>* | *http://<address>* | *<path>*] If *file://* is specified, Moab treats the destination as a flat text file. If *http://* is specified, Moab treats the destination as a hypertext transfer protocol file. If just a path is specified, Moab treats the destination as an executable.
Description	Specifies how Moab queries the resource manager (See Native RM, URL Notes, and interface details.).
Example	`RMCFG[base] CLUSTERQUERYURL=file:///tmp/cluster.config` *Moab reads `/tmp/cluster.config` when it queries base resource manager.*

CONFIGFILE

Format	*<STRING>*
Description	Specifies the resource manager specific configuration file that must be used to enable correct API communication. Only valid with LL- and SLURM-based interfaces.
Example	`RMCFG[base] TYPE=LL CONFIGFILE=/home/loadl/loadl_config` *The scheduler uses the specified file when establishing the resource manager/scheduler interface connection.*

DATARM

Format	*<RM NAME>*
Description	If specified, the resource manager uses the given storage resource manager to handle staging data in and out.
Example	`RMCFG[clusterB] DATARM=clusterB_storage` *When data staging is required by jobs starting/completing on clusterB, Moab uses the storage interface defined by clusterB_storage to stage and monitor the data.*

DEFAULTCLASS

Format	*<STRING>*
Description	Specifies the class to use if jobs submitted via this resource manager interface do not have an associated class.
Example	`RMCFG[internal] DEFAULTCLASS=batch` *Moab assigns the class batch to all jobs from the resource manager internal that do not have a class assigned.* If you are using PBS as the resource manager, a job will never come from PBS without a class, and the default will never apply.

DEFAULTHIGHSPEEDADAPTER

Format:	*<STRING>*
Default:	*sn0*
Description:	Specifies the default high speed switch adapter to use when starting LoadLeveler jobs (supported in version 4.2.2 and higher of Moab and 3.2 of LoadLeveler).
Example:	`RMCFG[base]  DEFAULTHIGHSPEEDADAPTER=sn1` *The scheduler will start jobs requesting a high speed adapter on sn1.*

DESCRIPTION

Format	*<STRING>*
Description	Specifies the human-readable description for the resource manager interface. If white space is used, the description should be quoted.
Example	`RMCFG[torque] TYPE=NATIVE DESCRIPTION='Torque RM for launching jobs'` *Moab annotates the TORQUE resource manager accordingly.*

ENV	
Format	Semi-colon-delimited (;) list of *<KEY>=<VALUE>* pairs
Default	*MOABHOMEDIR=<MOABHOMEDIR>*
Description	Specifies a list of environment variables that will be passed to URLs of type *exec://* for that resource manager.

Example	`RMCFG[base] ENV=HOST=node001;RETRYTIME=50`
	`RMCFG[base] CLUSTERQUERYURL=exec:///opt/moab/tools/cluster.query.pl`
	`RMCFG[base] WORKLOADQUERYURL=exec:///opt/moab/tools/workload.query.pl`
	The environment variables HOST and RETRYTIME (with values `node001` and 50 respectively) are passed to the `/opt/moab/tools/cluster.query.pl` and `/opt/moab/tools/workload.query.pl` when they are executed.

EPORT	
Format:	*<INTEGER>*
Description:	Specifies the event port to use to receive resource manager based scheduling events.
Example:	`RMCFG[base] EPORT=15017` *The scheduler will look for scheduling events from the resource manager host at port 15017.*

FAILTIME	
Format:	`[[[DD:]HH:]MM:]SS`
Description:	Specifies how long a resource manager must be down before any failure triggers associated with the resource manager fire.
Example:	`RMCFG[base] FAILTIME=3:00` *If the base resource manager is down for three minutes, any resource manager failure triggers fire.*

FLAGS	
Format	Comma-delimited list of zero or more of the following: asyncdelete, asyncstart, *autostart*, *autosync*, *client*, *fullcp*, *executionServer*, *hostingCenter*, *ignqueuestate*, *private*, *pushslavejobupdates*, *report*, *shared*, *slavepeer* or *static*
Description	Specifies various attributes of the resource manager. See Flag Details for more information.
Example	`RMCFG[base] FLAGS=static,slavepeer` *Moab uses this resource manager to perform a single update of node and job objects reported elsewhere.*

FNLIST	
Format	Comma-delimited list of zero or more of the following: *clusterquery*, *jobcancel*, *jobrequeue*, *jobresume*, *jobstart*, *jobsuspend*, *queuequery*, *resourcequery* or *workloadquery*
Description	By default, a resource manager utilizes all functions supported to query and control batch objects. If this parameter is specified, only the listed functions are used.
Example	`RMCFG[base] FNLIST=queuequery` *Moab only uses this resource manager interface to load queue configuration information.*

HOST	
Format	*<STRING>*
Default	*localhost*
Description	The host name of the machine on which the resource manager server is running.
Example	`RMCFG[base] host=server1`

IGNHNODES	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether to read in the PBSPro host nodes. This parameter is used in conjunction with **USEVNODES** on page 530. When both are set to *TRUE*, the host nodes are not queried.
Example	`RMCFG[pbs] IGNHNODES=TRUE`

JOBCANCELURL	
Format	*<protocol>*`://[`*<host>*`[:`*<port>*`]][`*<path>*`]`
Default	---
Description	Specifies how Moab cancels jobs via the resource manager. (See URL Notes below.)
Example	`RMCFG[base] JOBCANCELURL=exec:///opt/moab/job.cancel.lsf.pl` *Moab executes `/opt/moab/job.cancel.lsf.pl` to cancel specific jobs.*

JOBEXTENDDURATION	
Format	`[[[DD:]HH:]MM:]SS[,[[[DD:]HH:]MM:]SS][!][<]` (or *<MIN TIME>*`[,`*<MAX TIME>*`][!]`)
Default	---

JOBEXTENDDURATION

Description

Specifies the minimum and maximum amount of time that can be added to a job's walltime if it is possible for the job to be extended. (See MINWCLIMIT.) As the job runs longer than its current specified minimum wallclock limit (-l minwclimit, for example), Moab attempts to extend the job's limit by the minimum **JOBEXTENDDURATION**. This continues until either the extension can no longer occur (it is blocked by a reservation or job), the maximum **JOBEXTENDDURATION** is reached, or the user's specified wallclock limit (-l 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 < is at the end of the string, however, the job is extended the maximum amount possible.

JOBEXTENDDURATION and JOBEXTENDSTARTWALLTIME TRUE cannot be configured together. If they are in the same moab.cfg or are both active, then the JOBEXTENDDURATION will not be honored.

For example, comment out the JOBEXTENDSTARTWALLTIME.

```
RMCFG[base] JOBEXTENDDURATION=30,1:00:00
#JOBEXTENDSTARTWALLTIME TRUE
```

Example

```
RMCFG[base] JOBEXTENDDURATION=30,1:00:00
```

Moab extends a job's walltime by 30 seconds each time the job is about to run out of walltime until it is bound by one hour, a reservation/job, or the job's original "maximum" wallclock limit.

JOBIDFORMAT

Format

INTEGER

Default

Description

Specifies that Moab should use numbers to create job IDs. This eliminates multiple job IDs associated with a single job.

Example

```
RMCFG[base] JOBIDFORMAT=INTEGER
```

Job IDs are generated as numbers.

JOBMODIFYURL

Format

<protocol>://[<host>[:<port>]][<path>]

JOBMODIFYURL	
Default	---
Description	Specifies how Moab modifies jobs via the resource manager. (See URL Notes, and interface details.)
Example	`RMCFG[base] JOBMODIFYURL=exec://$TOOLSDIR/job.modify.dyn.pl` *Moab executes `/opt/moab/job.modify.dyn.pl` to modify specific jobs.*

JOBRSVRECREATE	
Format	Boolean
Default	*TRUE*
Description	Specifies whether Moab will re-create a job reservation each time job information is updated by a resource manager (See Considerations for Large Clusters for more information.).
Example	`RMCFG[base] JOBRSVRECREATE=FALSE` *Moab only creates a job reservation once when the job first starts.*

JOBSTARTURL	
Format	`<protocol>://[<host>[:<port>]][<path>]`
Default	*TRUE*
Description	Specifies how Moab starts jobs via the resource manager. (See URL Notes below.)
Example	`RMCFG[base] JOBSTARTURL=http://orion.bsu.edu:1322/moab/jobstart.cgi` *Moab triggers the `jobstart.cgi` script via http to start specific jobs.*

JOBSUBMITURL	
Format	*<protocol>*://[*<host>*[:*<port>*]][*<path>*]
Description	Specifies how Moab submits jobs to the resource manager (See URL Notes below.).
Example	`RMCFG[base] JOBSUBMITURL=exec://$TOOLSDIR/job.submit.dyn.pl` *Moab submits jobs directly to the database located on host `dbserver.flc.com`.*

JOBSUSPENDURL	
Format	*<protocol>*://[*<host>*[:*<port>*]][*<path>*]
Description	Specifies how Moab suspends jobs via the resource manager. (See URL Notes below.)
Example	`RMCFG[base] JOBSUSPENDURL=EXEC://$HOME/scripts/job.suspend` *Moab executes the `job.suspend` script when jobs are suspended.*

JOBVALIDATEURL	
Format	*<protocol>*://[*<host>*[:*<port>*]][*<path>*]
Description	Specifies how Moab validates newly submitted jobs (See URL Notes below.). If the script returns with a non-zero exit code, the job is rejected. (See User Proxying/Alternate Credentials.)
Example	`RMCFG[base] JOBVALIDATEURL=exec://$TOOLS/job.validate.pl` *Moab executes the '`job.validate.pl`' script when jobs are submitted to verify they are acceptable.*

MAXDSOP	
Format	*<INTEGER>*
Default	*-1* (unlimited)

MAXDSOP	
Description	Specifies the maximum number of data staging operations that may be simultaneously active.
Example	`RMCFG[ds] MAXDSOP=16`

MAXITERATIONFAILURECOUNT	
Format	*<INTEGER>*
Default	*80*
Description	Specifies the number of times the RM must fail within a certain iteration before Moab considers it down or corrupt. When an RM is down or corrupt, Moab will not attempt to interact with it.
Example	`RMCFG[base] MAXITERATIONFAILURECOUNT=25` *The RM `base` must fail 25 times in a single iteration for Moab to consider it down and cease interacting with it.*

MAXJOBPERMINUTE	
Format	*<INTEGER>*
Default	*-1* (unlimited)
Description	Specifies the maximum number of jobs allowed to start per minute via the resource manager.
Example	`RMCFG[base] MAXJOBPERMINUTE=5` *The scheduler only allows five jobs per minute to launch via the resource manager base.*

MAXJOBS	
Format	*<INTEGER>*

MAXJOBS	
Default	*0* (limited only by the Moab **MAXJOB** setting)
Description	Specifies the maximum number of active jobs that this interface is allowed to load from the resource manager. Only works with Moab peer resource managers at this time.
Example	`RMCFG[cluster1] SERVER=moab://cluster1 MAXJOBS=200` *The scheduler loads up to 200 active jobs from the remote Moab peer cluster1.*

MINETIME	
Format	*<INTEGER>*
Default	*1*
Description	Specifies the minimum time in seconds between processing subsequent scheduling events.
Example	`RMCFG[base] MINETIME=5` *The scheduler batch-processes scheduling events that occur less than five seconds apart.*

NMPORT	
Format	*<INTEGER>*
Default	(any valid port number)
Description	Allows specification of the resource manager's node manager port and is only required when this port has been set to a non-default value.
Example	`RMCFG[base] NMPORT=13001` *The scheduler contacts the node manager located on each compute node at port 13001.*

NODEFAILURERSVPROFILE

Format	*<STRING>*
Description	Specifies the rsv template to use when placing a reservation onto failed nodes (See also **NODEFAILURERESERVETIME** on page 877.).

Example	`# moab.cfg`
	`RMCFG[base] NODEFAILURERSVPROFILE=long`
	`RSVPROFILE[long]  DURATION=25:00RSVPROFILE[long]  USERLIST=john`
	The scheduler will use the long rsv profile when creating reservations over failed nodes belonging to base.

NODESTATEPOLICY

Format	One of *OPTIMISTIC* or *PESSIMISTIC*
Default	*PESSIMISTIC*
Description	Specifies how Moab should determine the state of a node when multiple resource managers are reporting state. *OPTIMISTIC* specifies that if any resource manager reports a state of up, that state will be used. *PESSIMISTIC* specifies that if any resource manager reports a state of down, that state will be used.
Example	`# moab.cfg` `RMCFG[native] TYPE=NATIVE NODESTATEPOLICY=OPTIMISTIC`

OMAP

Format	*<protocol>*`://[`*<host>*`[:`*<port>*`]][`*<path>*`]`
Description	Specifies an object map file that is used to map credentials and other objects when using this resource manager peer

Example	`moab.cfg`
	`RMCFG[peer1] OMAP=file:///opt/moab/omap.dat`
	When communicating with the resource manager peer1, objects are mapped according to the rules defined in the `/opt/moab/omap.dat` file.

PORT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the port on which the scheduler should contact the associated resource manager. The value *0* specifies that the resource manager default port should be used.
Example	`RMCFG[base] TYPE=PBS HOST=cws PORT=20001` *Moab attempts to contact the PBS server daemon on host cws, port 20001.*

PROVDURATION	
Format	`[[[DD:]HH:]MM:]SS`
Default	*2:30*
Description	Specifies the upper bound (walltime) of a provisioning request. After this duration, Moab will consider the privisioning attempt failed.
Example	`RMCFG[base] PROVDURATION=5:00` *When RM `base` provisions a node for more than 5 minutes, Moab considers the provisioning as having failed.*

PTYSTRING	
Format	*<STRING>*
Default	*srun -n1 -N1 --pty*

PTYSTRING

Description

When a SLURM interactive job is submitted, it builds an salloc command that gets the requested resources and an srun command that creates a terminal session on one of the nodes. The srun command is called the PTYString. PTYString is configured in `moab.cfg`.

There are two special things you can do with PTYString:

1. You can have `PTYSTRING=$salloc` which says to use the default salloc command (SallocDefaultCommand, look in the `slurm.conf` man page) defined in `slurm.conf`. Internally, Moab won't add a PTYString because SLURM will call the `SallocDefaultCommand`.
2. As in the example below, you can add `$SHELL`. `$SHELL`will be expanded to either what you request on the command line (such as msub -S /bin/tcsh -l) or to the value of `$SHELL` in your current session.

PTYString works only with SLURM.

Example

```
RMCFG[slurm] PTYSTRING="srun -n1 -N1 --pty --preserve-env $SHELL"
```

RESOURCECREATEURL

Format

<STRING>

Default

[*exec://<path>* | *http://<address>* | *<path>*]

If *exec://* is specified, Moab treats the destination as an executable file; if *http://* is specified, Moab treats the destination as a hypertext transfer protocol file.

Description

Specifies a script or method that can be used by Moab to create resources dynamically, such as creating a virtual machine on a hypervisor.

Example

```
RMCFG[base] RESOURCECREATEURL=exec:///opt/script/vm.provision.py
```

Moab invokes the `vm.provision.py` script, passing in data as command line arguments, to request a creation of new resources.

RESOURCETYPE

Format

{*COMPUTE*|*FS*|*LICENSE*|*NETWORK*|*PROV*}

Description

Specifies which type of resource this resource manager is configured to control. See Native Resource Managers for more information.

RESOURCETYPE

Example

```
RMCFG[base] TYPE=NATIVE RESOURCETYPE=FS
```

Resource manager base will function as a NATIVE resource manager and control file systems.

RMSTARTURL

Format

[*exec://<path>* | *http://<address>* | *<path>*]

If *exec://* is specified, Moab treats the destination as an executable file; if *http://* is specified, Moab treats the destination as a hypertext transfer protocol file.

Description

Specifies how Moab starts the resource manager.

Example

```
RMCFG[base] RMSTARTURL=exec:///tmp/nat.start.pl
```

Moab executes `/tmp/nat.start.pl` to start the resource manager base.

RMSTOPURL

Format

[*exec://<path>* | *http://<address>* | *<path>*]

If *exec://* is specified, Moab treats the destination as an executable file; if *http://* is specified, Moab treats the destination as a hypertext transfer protocol file.

Description

Specifies how Moab stops the resource manager.

Example

```
RMCFG[base] RMSTOPURL=exec:///tmp/nat.stop.pl
```

Moab executes `/tmp/nat.stop.pl` to stop the resource manager base.

SBINDIR

Format

<PATH>

Description

For use with TORQUE; specifies the location of the TORQUE system binaries (supported in TORQUE 1.2.0p4 and higher).

SBINDIR	
Example	`RMCFG[base] TYPE=pbs  SBINDIR=/usr/local/torque/sbin` *Moab tells TORQUE that its system binaries are located in `/usr/local/torque/sbin`.*

SERVER	
Format	*<URL>*
Description	Specifies the resource management service to use. If not specified, the scheduler locates the resource manager via built-in defaults or, if available, with an information service.
Example	`RMCFG[base] server=ll://supercluster.org:9705` *Moab attempts to use the Loadleveler scheduling API at the specified location.*

SLURMFLAGS	
Format	*<STRING>*
Description	Specifies characteristics of the SLURM resource manager interface. The *COMPRESSOUTPUT* flag instructs Moab to use the compact host list format for job submissions to SLURM. The flag *NODEDELTAQUERY* instructs Moab to request delta node updates when it queries SLURM for node configuration.
Example	`RMCFG[slurm] SLURMFLAGS=COMPRESSOUTPUT` *Moab uses the COMPRESSOUTPUT flag to determine interface characteristics with SLURM.*

SOFTTERMSIG	
Format	*<INTEGER>*or SIG*<X>*
Description	Specifies what signal to send the resource manager when a job reaches its soft wallclock limit. (See JOBMAXOVERRUN.)

SOFTTERMSIG

Example

```
RMCFG[base] SOFTTERMSIG=SIGUSR1
```

Moab routes the signal SIGUSR1 through the resource manager to the job when a job reaches its soft wallclock limit.

STAGETHRESHOLD

Format

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

Description

Specifies the maximum time a job waits to start locally before considering being migrated to a remote peer. In other words, if a job's start time on a remote cluster is less than the start time on the local cluster, but the difference between the two is less than **STAGETHRESHOLD**, then the job is scheduled locally. The aim is to avoid job/data staging overhead if the difference in start times is minimal.

If this attribute is used, backfill is disabled for the associated resource manager.

Example

```
RMCFG[remote_cluster] STAGETHRESHOLD=00:05:00
```

Moab only migrates jobs to remote_cluster if the jobs can start five minutes sooner on the remote cluster than they could on the local cluster.

STARTCMD

Format

<STRING>

Description

Specifies the full path to the resource manager job start client. If the resource manager API fails, Moab executes the specified start command in a second attempt to start the job.

Moab calls the start command with the format *<CMD><JOBID> -H <HOSTLIST>* unless the environment variable MOABNOHOSTLIST is set in which case Moab will only pass the job ID.

Example

```
RMCFG[base] STARTCMD=/usr/local/bin/qrun
```

Moab uses the specified start command if API failures occur when launching jobs.

SUBMITCMD	
Format	*<STRING>*
Description	Specifies the full path to the resource manager job submission client.
Example	`RMCFG[base] SUBMITCMD=/usr/local/bin/qsub` *Moab uses the specified submit command when migrating jobs.*

SUBMITPOLICY	
Format	One of *NODECENTRIC* or *PROCCENTRIC*
Default	*PROCCENTRIC*
Description	If set to *NODECENTRIC*, each specified node requested by the job is interpreted as a true compute host, not as a task or processor.
Example	`RMCFG[base] SUBMITPOLICY=NODECENTRIC` *Moab uses the specified submit policy when migrating jobs.*

SUSPENDSIG	
Format	*<INTEGER>* (valid UNIX signal between 1 and 64)
Default	RM-specific default
Description	If set, Moab sends the specified signal to a job when a job suspend request is issued.
Example	`RMCFG[base] SUSPENDSIG=19` *Moab uses the specified suspend signal when suspending jobs within the base resource manager.* **SUSPENDSIG** should not be used with TORQUE or other PBS-based resource managers.

SYNCJOBID

Format	*<BOOLEAN>*
Description	Specifies that Moab should migrate jobs to the local resource manager with the job's Moab-assigned job ID. In a grid, the grid-head will only pass dependencies to the underlying Moab if **SYNCJOBID** is set. This attribute can be used with the **JOBIDFORMAT** on page 516 attribute and **PROXYJOBSUBMISSION** on page 535 flag in order to synchronize job IDs between Moab and the resource manager. For more information about all steps necessary to synchronize job IDs between Moab and TORQUE, see **Synchronizing Job IDs in TORQUE and Moab** on page 532.
Example	`RMCFG[slurm] TYPE=wiki:slurm SYNCJOBID=TRUE`

SYSTEMMODIFYURL

Format	[*exec://<path>* | *http://<address>* | *<path>*] If *exec://* is specified, Moab treats the destination as an executable file; if *http://* is specified, Moab treats the destination as a hypertext transfer protocol file.
Description	Specifies how Moab modifies attributes of the system. This interface is used in data staging.
Example	`RMCFG[base] SYSTEMMODIFYURL=exec:///tmp/system.modify.pl` *Moab executes `/tmp/system.modify.pl` when it modifies system attributes in conjunction with the resource manager base.*

SYSTEMQUERYURL

Format	[*exec://<path>* | *http://<address>* | *<path>*] If *file://* is specified, Moab treats the destination as a flat text file; if *http://* is specified, Moab treats the destination as a hypertext transfer protocol file; if just a path is specified, Moab treats the destination as an executable.
Description	Specifies how Moab queries attributes of the system. This interface is used in data staging.
Example	`RMCFG[base] SYSTEMQUERYURL=file:///tmp/system.query` *Moab reads `/tmp/system.query` when it queries the system in conjunction with base resource manager.*

TARGETUSAGE	
Format	*<INTEGER>*[%]
Default	90%
Description	Amount of resource manager resources to explicitly use. In the case of a storage resource manager, indicates the target usage of data storage resources to dedicate to active data migration requests. If the specified value contains a percent sign (%), the target value is a percent of the configured value. Otherwise, the target value is considered to be an absolute value measured in megabytes (MB).

Example	`RMCFG[storage] TYPE=NATIVE RESOURCETYPE=storage`
	`RMCFG[storage] TARGETUSAGE=80%`
	Moab schedules data migration requests to never exceed 80% usage of the storage resource manager's disk cache and network resources.

TIMEOUT	
Format	*<INTEGER>*
Default	*30*
Description	Time (in seconds) the scheduler waits for a response from the resource manager.
Example	`RMCFG[base] TIMEOUT=40` *Moab waits 40 seconds to receive a response from the resource manager before timing out and giving up. Moab tries again on the next iteration.*

TRIGGER	
Format	*<TRIG_SPEC>*
Description	A trigger specification indicating behaviors to enforce in the event of certain events associated with the resource manager, including resource manager start, stop, and failure.
Example	`RMCFG[base] TRIGGER=<X>`

TYPE	
Format	*<RMTYPE>*[:*<RMSUBTYPE>*] where *<RMTYPE>* is one of the following: TORQUE, NATIVE, PBS, RMS, SSS, or WIKI and the optional *<RMSUBTYPE>* value is one of RMS.
Default	PBS
Description	Specifies type of resource manager to be contacted by the scheduler. For TYPE WIKI, AUTHTYPE must be set to CHECKSUM. The *<RMSUBTYPE>* option is currently only used to support Compaq's RMS resource manager in conjunction with PBS. In this case, the value PBS:RMS should be specified.

Example	`RMCFG[clusterA] TYPE=PBS HOST=clusterA PORT=15003`
	`RMCFG[clusterB] TYPE=PBS HOST=clusterB PORT=15005`
	Moab interfaces to two different PBS resource managers, one located on server clusterA at port 15003 and one located on server clusterB at port 15005.

USEVNODES	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether to schedule on PBS virtual nodes. When set to *TRUE*, Moab queries PBSPro for vnodes and puts jobs on vnodes rather than hosts. In some systems, such as PBS + Altix, it may not be desirable to read in the host nodes; for such situations refer to the IGNHNODES attribute.
Example	`RMCFG[pbs] USEVNODES=TRUE`

VARIABLES	
Format	*<VAR>*=*<VAL>*[,*<VAR>*=*<VAL>*]
Description	Opaque resource manager variables.
Example	`RMCFG[base] VARIABLES=SCHEDDHOST=head1` *Moab associates the variable SCHEDDHOST with the value head1 on resource manager base.*

VERSION	
Format	<STRING>
Default	*SLURM: 10200* (i.e., 1.2.0)
Description	Resource manager-specific version string.
Example	`RMCFG[base] VERSION=10124` *Moab assumes that resource manager base has a version number of 1.1.24.*

VMOWNERRM	
Format	*<STRING>*
Description	Used with provisioning resource managers that can create VMs. It specifies the resource manager that will own any VMs created by the resource manager.
Example	`RMCFG[torque]` `RMCFG[prov] RESOURCETYPE=PROV VMOWNERRM=torque`

WORKLOADQUERYURL	
Format	[*file://<path>* | *http://<address>* | *<path>*] If *file://* is specified, Moab treats the destination as a flat text file; if *http://* is specified, Moab treats the destination as a hypertext transfer protocol file; if just a path is specified, Moab treats the destination as an executable.
Description	Specifies how Moab queries the resource manager for workload information. (See Native RM, URL Notes, and interface details.)
Example	`RMCFG[TORQUE] WORKLOADQUERYURL=exec://$TOOLSDIR/job.query.dyn.pl` *Moab executes `/opt/moab/tools/job.query.dyn.pl` to obtain updated workload information from resource manager TORQUE.*

URL notes

URL parameters can load files by using the *file*, *exec*, and *http* protocols.

For the protocol *file*, Moab loads the data directly from the text file pointed to by path.

```
RMCFG[base] SYSTEMQUERYURL=file:///tmp/system.query
```

For the protocol *exec*, Moab executes the file pointed to by path and loads the output written to STDOUT. If the script requires arguments, you can use a question mark (?) between the script name and the arguments, and an ampersand (&) for each space.

```
RMCFG[base] JOBVALIDATEURL=exec://$TOOLS/job.validate.pl
RMCFG[native] CLUSTERQUERYURL=exec://opt/moab/tools/cluster.query.pl?-group=group1&-
arch=x86
```

Synchronizing Job IDs in TORQUE and Moab

Unless you use an **msub** on page 203 submit filter or you're in a grid, it is recommended that you use your RM-specific job submission command (for instance, qsub).

In order to synchronize your job IDs between TORQUE and Moabyou must perform the following steps:

1. Verify that you are using TORQUE version 2.5.6 or later.
2. Set **SYNCJOBID** on page 528 to *TRUE* in all resource managers.

   ```
   RMCFG[torque] TYPE=PBS SYNCJOBID=TRUE
   ```

3. Set the **PROXYJOBSUBMISSION** on page 535 flag. With *PROXYJOBSUBMISSION* enabled, you must run Moab as a TORQUE manager or operator. Verify that other users can submit jobs using msub. Moab, as a non-root user, should still be able to submit jobs to TORQUE and synchronize job IDs.

   ```
   RMCFG[torque] TYPE=PBS SYNCJOBID=TRUE
   RMCFG[torque] FLAGS=PROXYJOBSUBMISSION
   ```

4. Add **JOBIDFORMAT** on page 516=*INTEGER* to the internal RM. Adding this parameter forces Moab to only use numbers as job IDs and those numbers to synchronize across Moab, TORQUE, and the entire grid. This enhances the end-user experience as it eliminates multiple job IDs associated with a single job.

   ```
   RMCFG[torque] TYPE=PBS SYNCJOBID=TRUE
   RMCFG[torque] FLAGS=PROXYJOBSUBMISSION

   RMCFG[internal] JOBIDFORMAT=INTEGER
   ```

Resource Manager Configuration Details

As with all scheduler parameters, follows the syntax described within the Parameters Overview.

Resource Manager Types

The **RMCFG** parameter allows the scheduler to interface to multiple types of resource managers using the **TYPE** or **SERVER** attributes. Specifying these attributes, any of the following listed resource managers may be supported.

Type	Resource managers	Details
Moab	Moab Workload Manager	Use the Moab peer-to-peer (grid) capabilities to enable grids and other configurations. (See Grid Configuration.)
Native	Moab *Native* Interface	Used for connecting directly to scripts, files, databases, and Web services. (See Managing Resources Directly with the Native Interface.)
PBS	TORQUE (all versions)	N/A
SSS	Scalable Systems Software Project version 2.0 and higher	N/A
WIKI	Wiki interface specification version 1.0 and higher	Used for LRM, YRM, ClubMASK, BProc, SLURM, and others.

Resource Manager Name

Moab can support more than one resource manager simultaneously. Consequently, the **RMCFG** parameter takes an index value such as `RMCFG[clusterA]`. This index value essentially names the resource manager (as done by the deprecated parameter RMNAME). The resource manager name is used by the scheduler in diagnostic displays, logging, and in reporting resource consumption to the allocation manager. For most environments, the selection of the resource manager name can be arbitrary.

Resource Manager Location

The **HOST**, **PORT**, and **SERVER** attributes can be used to specify how the resource manager should be contacted. For many resource managers the interface correctly establishes contact using default values. These parameters need only to be specified for resource managers such as the WIKI interface (that do not include defaults) or with resources managers that can be configured to run at non-standard locations (such as PBS). In all other cases, the resource manager is automatically located.

Resource Manager Flags

The **FLAGS** attribute can be used to modify many aspects of a resources manager's behavior.

Flag	Description
ASYNCSTART	Jobs started on this resource manager start asynchronously. In this case, the scheduler does not wait for confirmation that the job correctly starts before proceeding. (See Large Cluster Tuning for more information.)
AUTOSTART	Jobs staged to this resource manager do not need to be explicitly started by the scheduler. The resource manager itself handles job launch.

Flag	Description
AUTOSYNC	Resource manager starts and stops together with Moab. This requires that the resource manager support a resource manager start and stop API or the RMSTARTURL and RMSTOPURL attributes are set.
BECOMEMASTER	Nodes reported by this resource manager will transfer ownership to this resource manager if they are currently owned by another resource manager that does not have this flag set.
CLIENT	A client resource manager object is created for diagnostic/statistical purposes or to configure Moab's interaction with this resource manager. It represents an external entity that consumes server resources or services, allows a local administrator to track this usage, and configures specific policies related to that resource manager. A client resource manager object loads no data and provides no services.
CLOCKSKEWCHECKING	Setting *CLOCKSKEWCHECKING* allows you to configure clock skew adjustments. Most of the time it is sufficient to use an NTP server to keep the clocks in your system synchronized.
COLLAPSEDVIEW	Does not work — not supported The resource manager masks details about local workload and resources and presents only information relevant to the remote server.
DYNAMICCRED	The resource manager creates credentials within the cluster as needed to support workload. (See Identity Manager Overview.)
EXECUTIONSERVER	The resource manager is capable of launching and executing batch workload.
FSISREMOTE	Add this flag if the working file system doesn't exist on the server to prevent Moab from validating files and directories at migration.
FULLCP	Always checkpoint full job information (useful with Native resource managers).
HOSTINGCENTER	The resource manager interface is used to negotiate an adjustment in dynamic resource access.
IGNQUEUESTATE	The queue state reported by the resource manager should be ignored. May be used if queues must be disabled inside of a particular resource manager to allow an external scheduler to properly operate.

Flag	Description
IGNWORKLOADSTATE	When this flag is applied to a native resource manager, any jobs that are reported via that resource manager's "workload query URL" have their reported state ignored. For example, if an RM has the *IgnWorkloadState* flag and it reports that a set of jobs have a state of "Running," this state is ignored and the jobs will either have a default state set or will inherit the state from another RM reporting on that same set of jobs. This flag only changes the behavior of RMs of type *NATIVE*.
LOCALWORKLOADEXPORT	When set, destination peers share information about local and remote jobs, allowing job management of different clusters at a single peer. For more information, see Workload Submission and Control.
MIGRATEALLJOBATTRIBUTES	When set, this flag causes additional job information to be migrated to the resource manager; additional job information includes things such as node features applied via `CLASSCFG[name] DEFAULT.FEATURES`, the account to which the job was submitted, and job walltime limit.
NOAUTORES	If the resource manager does not report CPU usage to Moab because CPU usage is at 0%, Moab assumes full CPU usage. When set, Moab recognizes the resource manager report as 0% usage. This is only valid for PBS.
NOCREATERESOURCE	To use resources discovered from this resource manager, they must be created by another resource manager first. For example, if you set *NOCREATERESOURCE* on RM `A`, which reports nodes 1 and 2, and RM `B` only reports node 1, then node 2 will not be created because RM `B` did not report it.
PRIVATE	The resources and workload reported by the resource manager are not reported to non-administrator users.
PROXYJOBSUBMISSION	Enables Admin proxy job submission, which means administrators may submit jobs in behalf of other users.
PUSHSLAVEJOBUPDATES	Enables job changes made on a grid slave to be pushed to the grid head or master. Without this flag, jobs being reported to the grid head do not show any changes made on the remote Moab server (via mjobctl and so forth).
RECORDGPUMETRICS	Enables the recording of GPU metrics for nodes.
RECORDMICMETRICS	Enables the recording of MIC metrics for nodes.
REPORT	N/A

Flag	Description
SHARED	Resources of this resource manager may be scheduled by multiple independent sources and may not be assumed to be owned by any single source.
SLAVEPEER	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.
STATIC	This resource manager only provides partial object information and this information does not change over time. Consequently, this resource manager may only be called once per object to modify job and node information.
USEPHYSICALMEMORY	This tells Moab to use a node's physical memory instead of the swap space. For example, if a node has 12 GB of RAM and an additional 12 GB of swap space, it has 24 GB of virtual memory. If a 4 GB job is assigned to that node, the reported available memory shows 12 GB because the job is using the swap space not the physical memory. The reported available memory doesn't decrease until the swap space is used up. When this flag is set, the 4 GB job immediately reduces the available memory to 8 GB (physical memory - used memory).
USERSPACEISSEPARATE	This tells Moab to ignore validating the user's uid and gid in the case that information doesn't exist on the Moab server.

Example

```
# resource manager 'torque' should use asynchronous job start
# and report resources in 'grid' mode
RMCFG[torque] FLAGS=asyncstart,grid
```

Scheduler/Resource Manager Interactions

In the simplest configuration, Moab interacts with the resource manager using the following four primary functions:

Function	Description
GETJOBINFO	Collect detailed state and requirement information about idle, running, and recently completed jobs.
GETNODEINFO	Collect detailed state information about idle, busy, and defined nodes.
STARTJOB	Immediately start a specific job on a particular set of nodes.

Function	Description
CANCELJOB	Immediately cancel a specific job regardless of job state.

Using these four simple commands, Moab enables nearly its entire suite of scheduling functions. More detailed information about resource manager specific requirements and semantics for each of these commands can be found in the specific resource manager (such as WIKI) overviews.

In addition to these base commands, other commands are required to support advanced features such as suspend/resume, gang scheduling, and scheduler initiated checkpoint restart.

Information on creating a new scheduler resource manager interface can be found in the Adding New Resource Manager Interfaces section.

12.3 Resource Manager Extensions

- Resource Manager Extension Specification
- Resource Manager Extension Values
- Resource Manager Extension Examples

All resource managers are not created equal. There is a wide range in what capabilities are available from system to system. Additionally, there is a large body of functionality that many, if not all, resource managers have no concept of. A good example of this is job QoS. Since most resource managers do not have a concept of quality of service, they do not provide a mechanism for users to specify this information. In many cases, Moab is able to add capabilities at a global level. However, a number of features require a *per job* specification. Resource manager extensions allow this information to be associated with the job.

Resource Manager Extension Specification

Specifying resource manager extensions varies by resource manager. TORQUE, OpenPBS, PBSPro, Loadleveler, LSF, S3, and Wiki each allow the specification of an *extension* field as described in the following table:

Resource manager	Specification method
TORQUE 2.0+	-l `> qsub -l nodes=3,qos=high sleepy.cmd`

Resource manager	Specification method
TORQUE 1.x/OpenPBS	-W x= `> qsub -l nodes=3 -W x=qos:high sleepy.cmd` OpenPBS does not support this ability by default but can be patched as described in the PBS Resource Manager Extension Overview.

Loadleveler	#@comment
	`#@nodes = 3`
	`#@comment = qos:high`

LSF	-ext `> bsub -ext advres:system.2`

PBSPro	-l
	`> qsub -l advres=system.2`
	Use of PBSPro resources requires configuring the `server_priv/resourcedef` file to define the needed extensions as in the following example:
	`advres type=string`
	`qos type=string`
	`sid type=string`
	`sjid type=string`

Wiki	comment `comment=qos:high`

Resource Manager Extension Values

Using the resource manager specific method, the following job extensions are currently available:

ADVRES on page 539
BANDWIDTH on page 539
DDISK on page 540
DEADLINE on page 540
DEPEND on page 540
DMEM on page 540
EPILOGUE on page 541
EXCLUDENODES on page 541
FEATURE on page 541
GATTR on page 542
GEOMETRY on page 542
GMETRIC on page 542
GPUs on page 543
GRES
HOSTLIST on page 544
JGROUP on page 545
JOBFLAGS
JOBREJECTPOLICY on page 546
LOGLEVEL on page 546
MAXMEM on page 547
MAXPROC on page 547
MEM on page 547
MICs on page 548
MINPREEMPTTIME on page 548
MINPROCSPEED on page 549
MINWCLIMIT on page 549
MSTAGEIN on page 550
MSTAGEOUT on page 550
NACCESSPOLICY on page 551
NALLOCPOLICY on page 552
NCPUS on page 552
NMATCHPOLICY on page 552
NODESET on page 553
NODESETCOUNT on page 553
NODESETDELAY on page 553
NODESETISOPTIONAL on page 553
OPSYS on page 554
PARTITION on page 554
PLACEMENT on page 554
PMEM on page 554
PROCS on page 555
PROLOGUE on page 556
PVMEM on page 556
QoS on page 556
QUEUEJOB on page 556
REQATTR on page 557
RESFAILPOLICY on page 557
RMTYPE on page 558
SIGNAL on page 558
SOFTWARE
SPRIORITY on page 558
TEMPLATE on page 558
TERMTIME on page 559
TPN on page 559
TRIG on page 560
TRL
VAR on page 561
VC on page 562
VMEM on page 562

ADVRES

Format	`[!]<RSVID>`
Description	Specifies that reserved resources are required to run the job. If *<RSVID>* is specified, then only resources within the specified reservation may be allocated (see Job to Reservation Binding). You can request to not use a specific reservation by using `advres=!<reservationname>`.

Example	`> qsub -l advres=grid.3`
	Resources for the job must come from grid.3.
	`> qsub -l advres=!grid.5`
	Resources for the job must not come from grid.5

BANDWIDTH

Format	*<DOUBLE>* (in MB/s)
Description	Minimum available network bandwidth across allocated resources (See Network Management.).
Example	`> bsub -ext bandwidth=120 chemjob.txt`

DDISK	
Format	*<INTEGER>*
Default	*0*
Description	Dedicated disk per task in MB.
Example	`> qsub -l ddisk=2000`

DEADLINE	
Format	Relative time: `[[[DD:]HH:]MM:]SS` Absolute time: `hh:mm:ss_mm/dd/yy`
Description	Either the relative completion deadline of job (from job submission time) or an absolute deadline in which you specify the date and time the job will finish.
Example:	`> qsub -l deadline=2:00:00,nodes=4 /tmp/bio3.cmd` *The job's deadline is 2 hours after its submission.*

DEPEND	
Format	`[`*<DEPENDTYPE>*`:][{jobname|jobid}.]`*<ID>*`[:[{jobname|jobid}.]`*<ID>*`]...`
Description	Allows specification of job dependencies for compute or system jobs. If no ID prefix (jobname or jobid) is specified, the ID value is interpreted as a job ID.

Example	`# submit job which will run after job 1301 and 1304 complete`
	`> msub -l depend=orion.1301:orion.1304 test.cmd`
	`orion.1322`
	`# submit jobname-based dependency job`
	`> msub -l depend=jobname.data1005 dataetl.cmd`
	`orion.1428`

DMEM	
Format	*<INTEGER>*

DMEM	
Default	*0*
Description	Dedicated memory per task in bytes.
Example	`> msub -l dmem=20480` *Moab will dedicate 20 MB of memory to the task.*

EPILOGUE	
Format	*<STRING>*
Description	Specifies a user owned epilogue script which is run before the system epilogue and `epilogue.user` scripts at the completion of a job. The syntax is `epilogue=`*<file>*. The file can be designated with an absolute or relative path. This parameter works only with TORQUE.
Example	`> msub -l epilogue=epilogue_script.sh job.sh`

EXCLUDENODES	
Format	{*<nodeid>*|*<node_range>*}[:...]
Description	Specifies nodes that should not be considered for the given job.

Example	`> msub -l excludenodes=k1:k2:k[5-8]`
	`# Comma separated ranges work only with SLURM`
	`> msub -l excludenodes=k[1-2,5-8]`

FEATURE	
Format	*<FEATURE>*[{:|}*<FEATURE>*]...

FEATURE

Description	Required list of node attribute/node features. If the *pipe* (|) character is used as a delimiter, the features are logically ORed together and the associated job may use resources that match any of the specified features.
Example	`> qsub -l feature='fastos:bigio' testjob.cmd`

GATTR

Format	*<STRING>*
Description	Generic job attribute associated with job. The maximum size for an attribute is 63 bytes (the core Moab size limit of 64, including a null byte)
Example	`> qsub -l gattr=bigjob`

GEOMETRY

Format:	`{(<TASKID>[,<TASKID>[,...]])[(<TASKID>[,...])...]}`
Description:	Explicitly specified task geometry.
Example:	`> qsub -l nodes=2:ppn=4 -W x=geometry:'{(0,1,4,5)(2,3,6,7)}' quanta2.cmd` *The job quanta2.cmd runs tasks 0, 1, 4, and 5 on one node, while tasks 2, 3, 6, and 7 run on another node.*

GMETRIC

Format	Generic metric requirement for allocated nodes where the requirement is specified using the format `<GMNAME>[:{lt:,le:,eq:,ge:,gt:,ne:}<VALUE>]`
Description	Indicates generic constraints that must be found on all allocated nodes. If a *<VALUE>* is not specified, the node must simply possess the generic metric (See Generic Metrics for more information.).
Example	`> qsub -l gmetric=bioversion:ge:133244 testj.txt`

GPUs

Format

`msub -l nodes=<VALUE>:ppn=<VALUE>:gpus=<VALUE>[:mode][:reseterr]`

Where `mode` is one of:

exclusive - The default setting. The GPU is used exclusively by one process thread.

exclusive_thread - The GPU is used exclusively by one process thread.

exclusive_process - The GPU is used exclusively by one process regardless of process thread.

If present, `reseterr` resets the ECC memory bit error counters. This only resets the volatile error counts, or errors since the last reboot. The permanent error counts are not affected.

Moab passes the `mode` and `reseterr` portion of the request to TORQUE for processing.

Moab does not support requesting GPUs as a GRES. Submitting `msub -l gres=gpus:x` does not work.

Description

Moab schedules GPUs as a special type of node-locked generic resources. When TORQUE reports GPUs to Moab, Moab can schedule jobs and correctly assign GPUs to ensure that jobs are scheduled efficiently. To have Moab schedule GPUs, configure them in TORQUE then submit jobs using the "GPU" attribute. Moab automatically parses the "GPU" attribute and assigns them in the correct manner. For information about GPU metrics, see GPGPUMetrics.

Examples

```
> msub -l nodes=2:ppn=2:gpus=1:exclusive_process:reseterr
```

Submits a job that requests 2 tasks, 2 processors and 1 GPU per task (2 GPUs total). Each GPU runs only threads related to the task and resets the volatile ECC memory big error counts at job start time.

```
> msub -l nodes=4:gpus=1,tpn=2
```

Submits a job that requests 4 tasks, 1 GPU per node (4 GPUs total), and 2 tasks per node. Each GPU is dedicated exclusively to one task process and the ECC memory bit error counters are not reset.

```
> msub -l nodes=4:gpus=1:reseterr
```

Submits a job that requests 4 tasks, 1 processor and 1 GPU per task (4 GPUs total). Each GPU is dedicated exclusively to one task process and resets the volatile ECC memory bit error counts at job start time.

```
> msub -l nodes=4:gpus=2+1:ppn=2,walltime=600
```

Submits a job that requests two different types of tasks, the first is 4 tasks, each with 1 processor and 2 gpus, and the second is 1 task with 2 processors. Each GPU is dedicated exclusively to one task process and the ECC memory bit error counters are not reset.

GRES and SOFTWARE	
Format	Percent sign (%) delimited list of generic resources where each resource is specified using the format *<RESTYPE>*[{+|:}*<COUNT>*]
Description	Indicates generic resources required by the job. If the generic resource is node-locked, it is a per-task count. If a *<COUNT>* is not specified, the resource count defaults to 1.

Example	`> qsub -W x=GRES:tape+2%matlab+3 testj.txt`
	When specifying more than one generic resource with -l, use the percent (%) character to delimit them.
	`> qsub -l gres=tape+2%matlab+3 testj.txt`
	`> qsub -l software=matlab:2 testj.txt`

HOSTLIST	
Format	+ delimited list of host names; also, ranges and regular expressions
Description	Indicates an *exact set*, *superset*, or *subset* of nodes on which the job must run. Use the caret (^) or asterisk (*) characters to specify a host list as *superset* or *subset* respectively. An exact set is defined without a caret or asterisk. An exact set means *all* the hosts in the specified hostlist must be selected for the job. A subset means the specified hostlist is used first to select hosts for the job. If the job requires more hosts than are in the subset hostlist, they will be obtained from elsewhere if possible. If the job does not require all of the nodes in the subset hostlist, it will use only the ones it needs. A superset means the hostlist is the *only* source of hosts that should be considered for running the job. If the job can't find the necessary resources in the superset hostlist it should *not* run. No other hosts should be considered in allocating the job.

HOSTLIST

Examples

```
> msub -l hostlist=nodeA+nodeB+nodeE
```

```
hostlist=foo[1-5]
```

This is an exact set of (foo1,foo2,...,foo5). The job must run on all these nodes.

```
hostlist=foo1+foo[3-9]
```

This is an exact set of (foo1,foo3,foo4,...,foo9). The job must run on all these nodes.

```
hostlist=foo[1,3-9]
```

This is an exact set of the same nodes as the previous example.

```
hostlist=foo[1-3]+bar[72-79]
```

This is an exact set of (foo1,foo2,foo3,bar72,bar73,...,bar79). The job must run on all these nodes.

```
hostlist=^node[1-50]
```

This is a superset of (node1,node2,...,node50). These are the only nodes that can be considered for the job. If the necessary resources for the job are not in this hostlist, the job is not run. If the job does not require all the nodes in this hostlist, it will use only the ones that it needs.

```
hostlist=*node[15-25]
```

This is a subset of (node15,node16,...,node25). The nodes in this hostlist are considered first for the job. If the necessary resources for the job are not in this hostlist, Moab tries to obtain the necessary resources from elsewhere. If the job does not require all the nodes in this hostlist, it will use only the ones that it needs.

JGROUP

Format	*<JOBGROUPID>*
Description	ID of job group to which this job belongs (different from the GID of the user running the job).
Example	`> msub -l JGROUP=bluegroup`

JOBFLAGS (aka FLAGS)	
Format	One or more of the following colon delimited job flags including ADVRES[:RSVID], NOQUEUE, NORMSTART, PREEMPTEE, PREEMPTOR, RESTARTABLE, or SUSPENDABLE (see job flag overview for a complete listing).
Description	Associates various flags with the job.
Example	`> qsub -l nodes=1,walltime=3600,jobflags=advres myjob.py`

JOBREJECTPOLICY	
Format:	One or more of *CANCEL*, *HOLD*, *IGNORE* (beta), *MAIL*, or *RETRY*
Default:	*HOLD*
Details:	Specifies the action to take when the scheduler determines that a job can never run. *CANCEL* issues a call to the resource manager to cancel the job. *HOLD* places a batch hold on the job preventing the job from being further evaluated until released by an administrator. Administrators can dynamically alter job attributes and possibly *fix* the job with mjobctl -m. With *IGNORE* (currently in beta), the scheduler will allow the job to exist within the resource manager queue but will neither process it nor report it. MAIL will send email to both the admin and the user when rejected jobs are detected. If *RETRY* is set, then Moab will allow the job to remain idle and will only attempt to start the job when the policy violation is resolved. Any combination of attributes may be specified. See QOSREJECTPOLICY. This is a per-job policy specified with msub -l. JOBREJECTPOLICY also exists as a global parameter.
Example:	`> msub -l jobrejectpolicy=cancel:mail`

LOGLEVEL	
Format	*<INTEGER>*
Description	Per job log verbosity.
Example	`> qsub -l -W x=loglevel:5 bw.cmd`

MAXMEM	
Forma:	*<INTEGER>* (in megabytes)
Description	Maximum amount of memory the job may consume across all tasks before the JOBMEM action is taken.
Example	`> qsub -W x=MAXMEM:1000mb bw.cmd` *If a RESOURCELIMITPOLICY is set for per-job memory utilization, its action will be taken when this value is reached.*

MAXPROC	
Format	*<INTEGER>*
Description	Maximum CPU load the job may consume across all tasks before the JOBPROC action is taken.
Example	`> qsub -W x=MAXPROC:4 bw.cmd` *If a RESOURCELIMITPOLICY is set for per-job processor utilization, its action will be taken when this value is reached.*

MEM	
Format	*<INTEGER>*
Description	Specify the maximum amount of physical memory used by the job.
Example	`> msub -l nodes=4:ppn=2,mem=1024mb` *The job must have 4 compute nodes with 2 processors per node. The job is limited to 1024 MB of memory.*

MICs

Format

`msub -l nodes=<VALUE>:ppn=<VALUE>:mics=<VALUE>[:mode]`

Where `mode` is one of:

exclusive - The default setting. The MIC is used exclusively by one process thread.

exclusive_thread - The MIC is used exclusively by one process thread.

exclusive_process - The MIC is used exclusively by one process regardless of process thread.

Moab passes the `mode` portion of the request to TORQUE for processing.

Moab does not support requesting MICs as a GRES. Submitting `msub -l gres=mics:x` does not work.

Description

Moab schedules MICs as a special type of node-locked generic resources. When TORQUE reports MICs to Moab, Moab can schedule jobs and correctly assign MICs to ensure that jobs are scheduled efficiently. To have Moab schedule MICs , configure them in TORQUE then submit jobs using the "MIC" attribute. Moab automatically parses the "MIC" attribute and assigns them in the correct manner.

Examples

```
> msub -l nodes=2:ppn=2:mics=1:exclusive_process
```

Submits a job that requests 2 tasks, 2 processors and 1 MIC per task (2 MICs total). Each MIC runs only threads related to the task.

```
> msub -l nodes=4:mics=1,tpn=2
```

Submits a job that requests 4 tasks, 1 MIC per node (4 MICs total), and 2 tasks per node. Each MIC is dedicated exclusively to one task process.

```
> msub -l nodes=4:mics=1
```

Submits a job that requests 4 tasks, 1 processor and 1 MIC per task (4 MICs total). Each MIC is dedicated exclusively to one task process.

```
> msub -l nodes=4:mics=2+1:ppn=2,walltime=600
```

Submits a job that requests two different types of tasks, the first is 4 tasks, each with 1 processor and 2 MICs , and the second is 1 task with 2 processors. Each MIC is dedicated exclusively to one task process.

MINPREEMPTTIME

Format

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

MINPREEMPTTIME

Description	Minimum time job must run before being eligible for preemption. Can only be specified if associated QoS allows per-job preemption configuration by setting the preemptconfig flag.
Example	`> qsub -l minpreempttime=900 bw.cmd` *Job cannot be preempted until it has run for 15 minutes.*

MINPROCSPEED

Format	*<INTEGER>*
Default	*0*
Description	Minimum processor speed (in MHz) for every node that this job will run on.
Example	`> qsub -W x=MINPROCSPEED:2000 bw.cmd` *Every node that runs this job must have a processor speed of at least 2000 MHz.*

MINWCLIMIT

Format	`[[DD:]HH:]MM:]SS`
Default	---
Description	Minimum wallclock limit job must run before being eligible for extension (See JOBEXTENDDURATION or JOBEXTENDSTARTWALLTIME.).
Example	`> qsub -l minwclimit=300,walltime=16000 bw.cmd` *Job will run for at least 300 seconds but up to 16,000 seconds if possible (without interfering with other jobs).*

MSTAGEIN	
Format	[<*SRCURL*>[|<*SRCRUL*>...]%]<*DSTURL*>
Description	Indicates a job has data staging requirements. The source URL(s) listed will be transferred to the execution system for use by the job. If more than one source URL is specified, the destination URL must be a directory. The format of <*SRCURL*> is: [PROTO://][HOST][:PORT]][/PATH] where the path is local. The format of <*DSTURL*> is: [PROTO://][HOST][:PORT]][/PATH] where the path is remote. PROTO can be any of the following protocols: ssh, file, or gsiftp. HOST is the name of the host where the file resides. PATH is the path of the source or destination file. The destination path may be a directory when sending a single file and must be a directory when sending multiple files. If a directory is specified, it must end with a forward slash (/). Valid variables include: $JOBID $HOME - Path the script was run from $RHOME - Home dir of the user on the remote system $SUBMITHOST $DEST - This is the Moab where the job will run $LOCALDATASTAGEHEAD If no destination is given, the protocol and file name will be set to the same as the source. The $RHOME (remote home directory) variable is for when a user's home directory on the compute node is different than on the submission host.
Example:	Copy helperscript.sh and datafile.txt from the local machine to /home/dev/ on host for use in execution of script.sh. $HOME is a path containing a preceding / (i.e. /home/adaptive)

MSTAGEOUT	
Format	[<*SRCURL*>[|<*SRCRUL*>...]%]<*DSTURL*>

MSTAGEOUT

Description

Indicates whether a job has data staging requirements. The source URL(s) listed will be transferred from the execution system after the completion of the job. If more than one source URL is specified, the destination URL must be a directory.

The format of *<SRCURL>* is: `[PROTO://][HOST][:PORT]][/PATH]` where the path is remote.

The format of *<DSTURL>* is: `[PROTO://][HOST][:PORT]][/PATH]` where the path is local.

PROTO can be any of the following protocols: ssh, file, or gsiftp.
HOST is the name of the host where the file resides.
PATH is the path of the source or destination file. The destination path may be a directory when sending a single file and must be a directory when sending multiple files. If a directory is specified, it must end with a forward slash (/).

Valid variables include:
$JOBID
$HOME - Path the script was run from
$RHOME - Home dir of the user on the remote system
$SUBMITHOST
$DEST - This is the Moab where the job will run
$LOCALDATASTAGEHEAD

If no destination is given, the protocol and file name will be set to the same as the source.

The $RHOME (remote home directory) variable is for when a user's home directory on the compute node is different than on the submission host.

Example

Copy `resultfile1.txt` and `resultscript.sh` from the execution system to `/home/dev/` after the execution of script.sh is complete. $HOME is a path containing a preceding `/` (i.e. `/home/adaptive`).

NACCESSPOLICY

Format

One of SHARED, SINGLEJOB, SINGLETASK, SINGLEUSER, or UNIQUEUSER

Description

Specifies how node resources should be accessed. (See Node Access Policies for more information).

The **naccesspolicy** option can only be used to make node access more constraining than is specified by the system, partition, or node policies. If the effective node access policy is *shared*, **naccesspolicy** can be set to *singleuser*, if the effective node access policy is *singlejob*, **naccesspolicy** can be set to *singletask*.

NACCESSPOLICY

Example

```
> qsub -l naccesspolicy=singleuser bw.cmd
```

```
> bsub -ext naccesspolicy=singleuser lancer.cmd
```

Job can only allocate free nodes or nodes running jobs by same user.

NALLOCPOLICY

Format

One of the valid settings for the parameter NODEALLOCATIONPOLICY

Description

Specifies how node resources should be selected and allocated to the job. (See Node Allocation Policies for more information.)

Example

```
> qsub -l nallocpolicy=minresource bw.cmd
```

Job should use the minresource node allocation policy.

NCPUS

Format

<INTEGER>

Description

The number of processors in one task where a task cannot span nodes. If **NCPUS** is used, then the resource manager's SUBMITPOLICY should be set to *NODECENTRIC* to get correct behavior. `-l ncpus=<#>` is equivalent to `-l nodes=1:ppn=<#>`when **JOBNODEMATCHPOLICY** is set to *EXACTNODE*. **NCPUS** is used when submitting jobs to an SMP. When using GPUs to submit to an SMP, use `-l ncpus=<#>:GPUs=<#>`.

You cannot request both ncpus and nodes in the same queue.

NMATCHPOLICY

Format

One of the valid settings for the parameter JOBNODEMATCHPOLICY

Description

Specifies how node resources should be selected and allocated to the job.

Example

```
> qsub -l nodes=2 -W x=nmatchpolicy:exactnode bw.cmd
```

*Job should use the EXACTNODE **JOBNODEMATCHPOLICY**.*

NODESET	
Format	*<SETTYPE>*:*<SETATTR>*[:*<SETLIST>*]
Description	Specifies node set constraints for job resource allocation (See the Node Set Overview for more information.).
Example	`> qsub -l nodeset=ONEOF:FEATURE:fastos:hiprio:bigmem bw.cmd`

NODESETCOUNT	
Format	*<INTEGER>*
Description	Specifies how many node sets a job uses.
Example	`> msub -l nodesetcount=2`

NODESETDELAY	
Format	`[[[DD:]HH:]MM:]SS`
Description	Causes Moab to attempt to span a job evenly across node sets unless doing so delays the job beyond the requested **NODESETDELAY**.
Example	`> qsub -l nodesetdelay=300,walltime=16000 bw.cmd`

NODESETISOPTIONAL	
Format	*<BOOLEAN>*
Description	Specifies whether the nodeset constraint is optional (See the NodeSet Overview for more information.). Requires SCHEDCFG[] FLAGS=allowperjobnodesetisoptional.
Example	`> msub -l nodesetisoptional=true bw.cmd`

OPSYS	
Format	*<OperatingSystem>*
Description	Specifies the job's required operating system.
Example	`> qsub -l nodes=1,opsys=rh73 chem92.cmd`

PARTITION	
Format	*<STRING>*`[:`*<STRING>*`]...`
Description	Specifies the partition (or partitions) in which the job must run. The job must have access to this partition based on system wide or credential based partition access lists.
Example	`> qsub -l nodes=1,partition=math:geology` *The job must only run in the math partition or the geology partition.*

PLACEMENT	
Format	`[numa=X][[:]sockets=Y][:usethreads]`
Description	Specifies the task placement of jobs.
Example	`> msub -l nodes=4:ppn=2,placement=numa=2` *This means to place the job on 4 compute nodes with 2 processors per node, with 2 different NUMA nodes per compute node, and 1 processor per NUMA node.*

PMEM	
Format	*<INTEGER>*
Description	Specifies the maximum amount of physical memory used by any single process of the job.

PMEM

Example

```
> msub -l nodes=4:ppn=2,pmem=1024mb
```

The job must have 4 compute nodes with 2 processors per node, and each process of the job is limited to 1024 MB of physical memory.

PREF

Format

`[{feature|variable}:]<STRING>[:<STRING>]...`

If feature or variable are not specified, then feature is assumed.

Description

Specifies which node features are preferred by the job and should be allocated if available. If preferred node criteria are specified, Moab favors the allocation of matching resources but is not bound to only consider these resources.

Preferences are not honored unless the node allocation policy is set to PRIORITY and the PREF priority component is set within the node's PRIORITYF attribute.

Example

```
> qsub -l nodes=1,pref=bigmem
```

The job may run on any nodes but prefers to allocate nodes with the bigmem feature.

PROCS

Format

`<INTEGER>`

Description

Requests a specific amount of processors for the job. Instead of users trying to determine the amount of nodes they need, they can instead decide how many processors they need and Moab will automatically request the appropriate amount of nodes from the RM. This also works with feature requests, such as `procs=12[:feature1[:feature2[-]]]`.

Using this resource request overrides any other processor or node related request, such as `nodes=4`.

Example

```
> msub -l procs=32 myjob.pl
```

Moab will request as many nodes as is necessary to meet the 32-processor requirement for the job.

PROLOGUE	
Format	*<STRING>*
Description	Specifies a user owned prologue script which will be run after the system prologue and `prologue.user` scripts at the beginning of a job. The syntax is `prologue=<file>`. The file can be designated with an absolute or relative path. This parameter works only with TORQUE.
Example	`> msub -l prologue=prologue_script.sh job.s`

PVMEM	
Format	*<INTEGER>*
Description	Specify the maximum amount of virtual memory used by any single process in the job.
Example	`> msub -l nodes=4:ppn=2,pvmem=1024mb` *The job must have 4 compute nodes with 2 processors per node, and each process of the job is limited to 1024 MB of virtual memory.*

QoS	
Format	*<STRING>*
Description	Requests the specified QoS for the job.
Example	`> qsub -l walltime=1000,qos=highprio biojob.cmd`

QUEUEJOB	
Format	*<BOOLEAN>*
Default	*TRUE*

QUEUEJOB	
Description	Indicates whether or not the scheduler should queue the job if resources are not available to run the job immediately
Example	`> msub -l nodes=1,queuejob=false test.cmd`

REQATTR	
Format	Required node attributes with version number support: *<ATTRIBUTE>*[{>=|>|<=|<|=} *<VERSION>*]
Description	Indicates required node attributes. Values may include letters, numbers, dashes, underscores, and spaces. You can choose one of four requirement types for each node attribute you request: • **must** – The node on which this job runs must include the attribute at the value specified. If no node matches this requirement, Moab will not schedule the job. • **must not** – The node on which this job runs must not include the attribute at the value specified. If no node matches this requirement, Moab will not schedule the job. • **should** – If possible, the node on which this job runs should include the attribute at the value specified. If no node matches this requirement, Moab selects a node without it. • **should not** – If possible, the node on which this job runs should not include the attribute at the value specified. If no node matches this requirement, Moab selects a node without it. If you do not specify a requirement type, Moab assumes "must." For information about using reqattr to request dynamic features, see **Configuring dynamic features in TORQUE and Moab** on page 563.
Example	`> qsub -l reqattr=matlab=7.1 testj.txt`

RESFAILPOLICY	
Format	One of *CANCEL*, *HOLD*, *IGNORE*, *NOTIFY*, or *REQUEUE*
Description	Specifies the action to take on an executing job if one or more allocated nodes fail. This setting overrides the global value specified with the NODEALLOCRESFAILUREPOLICY parameter.
Example	`> msub -l resfailpolicy=ignore` *For this particular job, ignore node failures.*

RMTYPE	
Format	*<STRING>*
Description	One of the resource manager types currently available within the cluster or grid. Typically, this is one of *PBS*, *LSF*, *LL*, *SGE*, *SLURM*, *BProc*, and so forth.
Example	`> msub -l rmtype=ll` Only run job on a Loadleveler destination resource manager.

SIGNAL	
Format	*<INTEGER>*[@*<OFFSET>*]
Description	Specifies the pre-termination signal to be sent to a job prior to it reaching its walltime limit or being terminated by Moab. The optional offset value specifies how long before job termination the signal should be sent. By default, the pre-termination signal is sent one minute before a job is terminated
Example	`> msub -l signal=32@120 bio45.cmd`

SPRIORITY	
Format	*<INTEGER>*
Default	*0*
Description	Allows Moab administrators to set a system priority on a job (similar to setspri). This only works if the job submitter is an administrator.
Example	`> qsub -l nodes=16,spriority=100 job.cmd`

TEMPLATE	
Format	*<STRING>*
Description	Specifies a job template to be used as a set template. The requested template must have `SELECT=TRUE` (See Job Templates.).

TEMPLATE	
Example	`> msub -l walltime=1000,nodes=16,template=biojob job.cmd`

TERMTIME	
Format	*<TIMESPEC>*
Default	*0*
Description	Specifies the time at which Moab should cancel a queued or active job (See Job Deadline Support.).
Example	`> msub -l nodes=10,walltime=600,termtime=12:00_Jun/14 job.cmd`

TPN	
Format	*<INTEGER>*[+]
Default	*0*

TPN	
Description	Tasks per node allowed on allocated hosts. If the plus (+) character is specified, the tasks per node value is interpreted as a minimum tasks per node constraint; otherwise it is interpreted as an exact tasks per node constraint. **Differences between TPN and PPN**: There are two key differences between the following: (A) `qsub -l nodes=12:ppn=3` and (B) `qsub -l nodes=12,tpn=3`. The first difference is that **ppn** is interpreted as the *minimum* required tasks per node while **tpn** defaults to exact tasks per node; case (B) executes the job with exactly *3* tasks on each allocated node while case (A) executes the job with at least *3* tasks on each allocated node- `nodeA:4,nodeB:3,nodeC:5` The second major difference is that the line, `nodes=X:ppn=Y` actually requests X*Y tasks, whereas `nodes=X,tpn=Y` requests only X tasks. **TPN with TORQUE as an RM**: Moab interprets nodes loosely as procs. TORQUE interprets nodes as the number of nodes from the actual number of nodes that you have in your nodes file, not your total number of procs. This means that if TORQUE is your resource manager and you specify `msub -l nodes=16:tpn=8` but do not have 16 nodes, TORQUE will not run the job. Instead, you should specify `msub -l procs=16:tpn=8`. To resolve the problem long term, you can also set `server resources_available.nodect` to the total number of procs in your system and use `msub -l nodes=16:tpn=8` as you would in a non-TORQUE Moab environment. For more information, see resources_available in the TORQUE Administrator Guide.
Example	`> msub -l nodes=10,walltime=600,tpn=4 job.cmd`

TRIG	
Format:	*<TRIGSPEC>*
Description:	Adds trigger(s) to the job (See **Creating a trigger** on page 658 for specific syntax.). Job triggers can only be specified if allowed by the QoS flag trigger.
Example:	`> qsub -l trig=etype=start\&atype=exec\&action="/tmp/email.sh job.cmd"`

TRL (Format 1)	
Format	*<INTEGER>*[@*<INTEGER>*][:*<INTEGER>*[@*<INTEGER>*]]...

TRL (Format 1)	
Default:	*0*
Description:	Specifies alternate task requests with their optional walltimes (See Malleable Jobs.).

Example:	`> msub -l trl=2@500:4@250:8@125:16@62 job.cmd`
	or
	`> qsub -l trl=2:3:4`

TRL (Format 2)	
Format	*<INTEGER>-<INTEGER>*
Default	*0*
Description	Specifies a range of task requests that require the same walltime (See Malleable Jobs.).
Example	`> msub -l trl=32-64 job.cmd` For optimization purposes Moab does not perform an exhaustive search of all possible values but will at least do the beginning, the end, and 4 equally distributed choices in between.

VAR	
Format	*<ATTR>*[:*<VALUE>*]
Description	Adds a generic variable or variables to the job.
Example	`> msub -l VAR=testvar1:testvalue1` *Single variable* *Multiple variables*

VC	
Format	vc=<*NAME*>
Description	Submits the job or workflow to a virtual container (VC).
Example	vc=vc13

VMEM	
Format:	<*INTEGER*>
Description:	Specify the maximum amount of virtual memory used by all concurrent processes in the job.
Example:	> msub -l nodes=4:ppn=2,vmem=1024mb *The job must have 4 compute nodes with 2 processors per node, and the job is limited to 1024 MB of virtual memory.*

Resource Manager Extension Examples

If more than one extension is required in a given job, extensions can be concatenated with a semicolon separator using the format <*ATTR*>:<*VALUE*>[;<*ATTR*>:<*VALUE*>]...

Example 12-1:

```
#@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 12-2:

```
# 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 12-3:

```
>  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.

Configuring dynamic features in TORQUE and Moab

Used together, the reqattr RM extension and TORQUE $varattr parameter allow you to create jobs that request resources that may change or disappear. For example, if you wanted a job to request a certain version of Octave but different versions are configured on each node and updated at any time, you can create a script that searches for the feature and version on the nodes at a specified interval. Your Moab job can then retrieve the dynamic node attributes from the latest poll and use them for scheduling.

This functionality is available when you use the TORQUE `$varattr` parameter to configure a script that regularly retrieves updates on the nodes' feature(s) and the `reqattr` RM extension to require a feature with a certain value.

To set up a dynamic feature in TORQUE and Moab

1. Create a script that pulls the information you need. For instance, the following script pulls the version of Octave on each node and prints it.

```
#!/bin/bash
# pull the version string for octave and print it for $varattr
version_str=`octave -v | grep version`
[[ $version_str =~ ([[:digit:]].[[:digit:]].[[:digit:]]) ]]
echo "Octave: ${BASH_REMATCH[1]}"
```

2. Use the TORQUE `$varattr` parameter to configure the script. Specify both the number of seconds between each time TORQUE runs the script and the path to the script. If you set the seconds to `-1`, the script will run just once. You may include arguments if desired. In the following example, the `varattr` parameter specifies that TORQUE calls the Octave script every 30 seconds.

```
$varattr 30 /usr/local/scripts/octave.sh
```

3. Submit your job in Moab, specifying `reqattr` as a resource. In this example, the job requests a node where the octave feature has a value of 3.2.4 (that the node has Octave version 3.2.4 installed).

```
> msub -l rerqattr=octave=3.2.4 myJob.sh
```

Your job requests a node with Octave version 3.2.4. TORQUE passes the most recent (pulled within the last 30 seconds) version of Octave on each node. Moab then schedules the job on a node that currently has Octave 3.2.4.

Related topics

- Resource Manager Overview

12.3.1 PBS Resource Manager Extensions

Resource manager extensions within PBS are used by setting the -W flag. To enable this flag, some versions of PBS must be rebuilt. TORQUE and recent OSCAR distributions come with the flag enabled by default. Most other versions do not. The required steps are documented in what follows:

1. Shut down the PBS server.

```
> qterm -t quick
#shutdown PBS server
```

2. cd to the directory from which you executed the PBS 'configure' at install time

```
> make distclean
> ./configure <WITH OPTIONS>
```

3. Create addparam script
   (chmod +x addparam)

```
> addparam x
> make
```

Backup current $PBS_HOMEDIR directory contents

$PBS_HOMEDIR defaults to /usr/spool/PBS.

```
> make install
```

4. Restore old $PBS_HOMEDIR directory contents

```
> pbs_server    # restart PBS server
```

A job's QoS level can then be specified using the qsub -W flag. For example, `qsub -W x=iQOS:hi -l nodes=4` ...

```
  #!/bin/sh
#script:  addparam
#usage:   addparam $Parameter  [S|L]
NewParameter=$1
ParameterType=x$2
if [ ! -d src/include ];  then
  echo "error: `basename $0` src/include doesn't exist, run configure"
1>&2
  exit 1
fi
#   run make in this directory to pull over the template files
cd src/include
if make
then
  if grep -q "\"$NewParameter\"" site_*.h 2>/dev/null;  then
    echo "parameter $NewParameter previously added"
    exit 0
  fi
fi
chmod +w site_job_attr_enum.h
echo "
  JOB_SITE_ATR_$1,
" >> site_job_attr_enum.h
chmod +w site_job_attr_def.h
if [ $ParameterType = "xS" ]
then
  echo "
    {   \"$NewParameter\",
        decode_str,
        encode_str,
        set_str,
        comp_str,
        free_str,
```

```
        NULL_FUNC,
        READ_WRITE,
        ATR_TYPE_STR,
        PARENT_TYPE_JOB
    },
  " >> site_job_attr_def.h
else
  echo "
    {   \"$NewParameter\",
        decode_l,
        encode_l,
        set_l,
        comp_l,
        free_null,
        NULL_FUNC,
        READ_WRITE,
        ATR_TYPE_LONG,
        PARENT_TYPE_JOB
    },
  " >> site_job_attr_def.h
fi
exit 0
```

12.4 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.

- Resource Manager Specific Interfaces
- Wiki Interface
- SSS Interface

Resource Manager Specific Interfaces

If you require tighter integration and need additional instruction, see Managing Resources Directly with the Native Interface. If you would like consultation on support for a new resource manager type, please contact the Professional Services group at Adaptive Computing.

Wiki Interface

The Wiki interface is already defined as a resource manager type, so no modifications are required within Moab. Additionally, no resource manager specific library or header file is required. However,

within the resource manager, internal job and node objects and attributes must be manipulated and placed within Wiki based interface concepts as defined in the Wiki Overview. Additionally, resource manager parameters must be created to allow a site to configure this interface appropriately.

SSS Interface

The SSS interface is an XML based generalized resource manager interface. It provides an extensible, scalable, and secure method of querying and modifying general workload and resource information.

Related topics

- Creating New Tools within the Native Resource Manager Interface

12.5 Managing Resources Directly with the Native Interface

- Native Interface Overview
- Configuring the Native Interface
  - Configuring the Resource Manager
  - Reporting Resources
- Generating Cluster Query Data
  - Flat Cluster Query Data
  - Interfacing to FLEXlm
  - Interfacing to Nagios
  - Interfacing to Supermon
- Configuring Resource Types
- Creating New Tools to Manage the Cluster

Native Interface Overview

The Native interface allows a site to augment or even fully replace a resource manager for managing resources. In some situations, the full capabilities of the resource manager are not needed and a lower cost or lower overhead alternative is preferred. In other cases, the nature of the environment may make use of a resource manager impossible due to lack of support. Still, in other situations it is desirable to provide information about additional resource attributes, constraints, or state from alternate sources.

In any case, Moab provides the ability to directly query and manage resources alongside of or without the use of a resource manager. This interface, called the NATIVE interface can also be used to launch, cancel, and otherwise manage jobs. This NATIVE interface offers several advantages including the following:

- No cost associated with purchasing a resource manager
- No effort required to install or configure the resource manager
- Ability to support abstract resources
- Ability to support abstract jobs
- Ability to integrate node availability information from multiple sources
- Ability to augment node configuration and utilization information provided by a resource manager

However, the NATIVE interface may also have some drawbacks.

- No support for standard job submission languages
- Limited default configured and utilized resource tracking (additional resource tracking available with additional effort)

At a high level, the native interface works by launching threaded calls to perform standard resource manager activities such as managing resources and jobs. The desired calls are configured within Moab and used whenever an action or updated information is required.

Configuring the Native Interface

Using the native interface consists of defining the interface type and location. As mentioned earlier, a single object may be fully defined by multiple interfaces simultaneously with each interface updating a particular aspect of the object.

Configuring the Resource Manager

The Native resource manager must be configured using the RMCFG parameter. To specify the native interface, the **TYPE** attribute must be set to *NATIVE*.

```
RMCFG[local] TYPE=NATIVE
RMCFG[local] CLUSTERQUERYURL=exec:///tmp/query.sh
```

Reporting Resources

To indicate the source of the resource information, the **CLUSTERQUERYURL** attribute of the RMCFG parameter should be specified. This attribute is specified as a URL where the protocols *FILE*, *EXEC*, and *SQL* are allowed. If a protocol is not specified, the protocol *EXEC* is assumed.

Format	Description
EXEC	Execute the script specified by the URL path. Use the script stdout as data.
FILE	Load the file specified by the URL path. Use the file contents as data.
SQL	Load data directly from an SQL database using the FULL format described below.

Moab considers a NativeRM script to have failed if it returns with a non-zero exit code, or if the CHILDSTDERRCHECK parameter is set and its appropriate conditions are met. In addition, the NativeRM

script associated with a job submit URL will be considered as having failed if its standard output stream contains the text ERROR.

This simple example queries a file on the server for information about every node in the cluster. This differs from Moab remotely querying the status of each node individually.

```
RMCFG[local]    TYPE=NATIVE
RMCFG[local]    CLUSTERQUERYURL=file:///tmp/query.txt
```

Generating Cluster Query Data

Flat Cluster Query Data

If the *EXEC* or *FILE* protocol is specified in the **CLUSTERQUERYURL** attribute, the data should provide flat text strings indicating the state and attributes of the node. The format follows the Moab Resource Manager Language Interface Specification where attributes are delimited by white space rather than ';' (See Resource Data Format):

Describes any set of node attributes with format: *<NAME><ATTR>=<VAL> [<ATTR>=<VAL>]...*

<NAME>	Name of node
<ATTR>	Node attribute
<VAL>	Value of node attribute

```
n17 CPROC=4 AMEMORY=100980 STATE=idle
```

Interfacing to FLEXlm

Moab can interface with FLEXlm to provide scheduling based on license availability. Informing Moab of license dependencies can reduce the number of costly licenses required by your cluster by allowing Moab to intelligently schedule around license limitations.

Provided with Moab in the tools directory is a Perl script, license.mon.flexLM.pl. This script queries a FLEXlm license server and gathers data about available licenses. This script then formats this data for Moab to read through a native interface. This script can easily be used by any site to help facilitate FLEXlm integration--the only modification necessary to the script is setting the @FLEXlmCmd to specify the local command to query FLEXlm. To make this change, edit license.mon.flexLM.pl and, near the top of the file, look for the line:

```
my @FLEXlmCmd = ("SETME");
```

Set the @FLEXlmCmd to the appropriate value for your system to query a license server and license file (if applicable). If lmutil is not in the PATH variable, specify its full path. Using the lmutil -a argument will cause it to report all licenses. The -c option can be used to specify an optional license file.

To test this script, run it manually. If working correctly, it will produce output similar to the following:

```
> ./license.mon.flexLM.pl
```

```
GLOBAL UPDATETIME=1104688300 STATE=idle ARES=autoCAD:130,idl_mpeg:160
CRES=autoCAD:200,idl_mpeg:330
```

If the output looks incorrect, set the $LOGLEVEL variable inside of `license.mon.flexLM.pl`, run it again, and address the reported failure.

Once the license interface script is properly configured, the next step is to add a *license* native resource manager to Moab via the `moab.cfg` file:

```
RMCFG[FLEXlm]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm]   CLUSTERQUERYURL=exec://$TOOLSDIR/flexlm/license.mon.flexLM.pl
...
```

Once this change is made, restart Moab. The command mdiag -R can be used to verify that the resource manager is properly configured and is in the state `Active`. Detailed information regarding configured and utilized licenses can be viewed by issuing the mdiag -n. Floating licenses (non-node-locked) will be reported as belonging to the GLOBAL node.

Due to the inherent conflict with the plus sign (+), the provided license manager script replaces occurrences of the plus sign in license names with the underscore symbol (_). This replacement requires that licenses with a plus sign in their names be requested with an underscore in place of any plus signs.

Interfacing to Multiple License Managers Simultaneously

If multiple license managers are used within a cluster, Moab can interface to each of them to obtain the needed license information. In the case of FLEXlm, this can be done by making one copy of the `license.mon.flexLM.pl` script for each license manager and configuring each copy to point to a different license manager. Then, within Moab, create one native resource manager interface for each license manager and point it to the corresponding script as in the following example:

```
RMCFG[FLEXlm1]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm1]   CLUSTERQUERYURL=exec://$TOOLSDIR/flexlm/license.mon.flexLM1.pl
RMCFG[FLEXlm2]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm2]   CLUSTERQUERYURL=exec://$TOOLSDIR/flexlm/license.mon.flexLM2.pl
RMCFG[FLEXlm3]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm3]   CLUSTERQUERYURL=exec://$TOOLSDIR/flexlm/license.mon.flexLM3.pl
...
```

For an overview of license management, including job submission syntax, see Section 13.7, License Management.

It may be necessary to increase the default limit, MMAX_GRES. See Appendix D for more implementation details.

Interfacing to Nagios

Moab can interface with Nagios to provide scheduling based on network hosts and services availability.

Nagios installation and configuration documentation can be found at Nagios.org.

Provided with Moab in the tools directory is a Perl script, node.query.nagios.pl. This script reads the Nagios `status.dat` file and gathers data about network hosts and services. This script then formats data for Moab to read through a native interface. This script can be used by any site to help facilitate Nagios integration. To customize the data that will be formatted for Moab, make the changes in this script.

You may need to customize the associated configuration file in the etc directory, `config.nagios.pl`. The statusFile line in this script tells Moab where the Nagios `status.dat` file is located. Make sure that the path name specified is correct for your site. Note that the interval which Nagios updates the Nagios status.dat file is specified in the Nagios `nagios.cfg` file. Refer to Nagios documentation for further details.

To make these changes, familiarize yourself with the format of the Nagios `status.dat` file and make the appropriate additions to the script to include the desired Moab RM language (formerly WIKI) Interface attributes in the Moab output.

To test this script, run it manually. If working correctly, it will produce output similar to the following:

```
> ./node.query.nagios.pl
gateway STATE=Running
localhost STATE=Running CPULOAD=1.22 ADISK=75332
```

Once the Nagios interface script is properly configured, the next step is to add a Nagios native resource manager to Moab via the `moab.cfg` file:

```
RMCFG[nagios] TYPE=NATIVE
RMCFG[nagios] CLUSTERQUERYURL=exec://$TOOLSDIR/node.query.nagios.pl
...
```

Once this change is made, restart Moab. The command mdiag -R can be used to verify that the resource manager is properly configured and is in the state `Active`. Detailed information regarding configured Nagios node information can be viewed by issuing the mdiag -n -v.

```
> mdiag -n -v
compute node summary
Name                    State   Procs      Memory         Disk          Swap
Speed   Opsys   Arch Par   Load Rsv Classes                          Network
           Features
gateway               Running    0:0         0:0           0:0           0:0
1.00       -      - dav   0.00   0 -                                 -
          -
  WARNING:  node 'gateway' is busy/running but not assigned to an active job
  WARNING:  node 'gateway' has no configured processors
localhost             Running    0:0         0:0       75343:75347       0:0
1.00       -      - dav   0.48   0 -                                 -
          -
  WARNING:  node 'localhost' is busy/running but not assigned to an active job
  WARNING:  node 'localhost' has no configured processors
-----                     ---    3:8      1956:1956    75345:75349    5309:6273
Total Nodes: 2  (Active: 2  Idle: 0  Down: 0)
```

Interfacing to Supermon

Moab can integrate with Supermon to gather additional information regarding the nodes in a cluster. A Perl script is provided in the tools directory that allows Moab to connect to the Supermon server. By default the Perl script assumes that Supermon has been started on port 2709 on localhost. These defaults

can be modified by editing the respective parameter in `config.supermon.pl` in the `etc` directory. An example setup is shown below.

```
RMCFG[TORQUE]  TYPE=pbs
RMCFG[supermon] TYPE=NATIVE CLUSTERQUERYURL=exec://$HOME/tools/node.query.supermon.pl
```

To confirm that Supermon is properly connected to Moab, issue mdiag -R -v. The output should be similar to the following example, specifically there are no errors about the **CLUSTERQURYURL**.

```
diagnosing resource managers
RM[TORQUE]  State: Active
  Type:               PBS  ResourceType: COMPUTE
  Server:             keche
  Version:            '2.2.0-snap.200707181818'
  Job Submit URL:     exec:///usr/local/bin/qsub
  Objects Reported:   Nodes=3 (6 procs)  Jobs=0
  Flags:              executionServer
  Partition:          TORQUE
  Event Management:   EPORT=15004  (no events received)
  Note:  SSS protocol enabled
  Submit Command:     /usr/local/bin/qsub
  DefaultClass:       batch
  RM Performance:     AvgTime=0.26s  MaxTime=1.04s  (4 samples)
  RM Languages:       PBS
  RM Sub-Languages:   -
RM[supermon]  State: Active
  Type:               NATIVE  ResourceType: COMPUTE
  Cluster Query URL:  exec://$HOME/node.query.supermon.pl
  Objects Reported:   Nodes=3 (0 procs)  Jobs=0
  Partition:          supermon
  Event Management:   (event interface disabled)
  RM Performance:     AvgTime=0.03s  MaxTime=0.11s  (4 samples)
  RM Languages:       NATIVE
  RM Sub-Languages:   -

Note:  use 'mrmctl -f messages ' to clear stats/failures
```

Run the Perl script by itself. The script's results should look similar to this:

```
vm01 GMETRIC[CPULOAD]=0.571428571428571 GMETRIC[NETIN]=133 GMETRIC[NETOUT]=702 GMETRIC
[NETUSAGE]=835
vm02 GMETRIC[CPULOAD]=0.428571428571429 GMETRIC[NETIN]=133 GMETRIC[NETOUT]=687 GMETRIC
[NETUSAGE]=820
keche GMETRIC[CPULOAD]=31 GMETRIC[NETIN]=5353 GMETRIC[NETOUT]=4937 GMETRIC[NETUSAGE]
=10290
```

If the preceding functioned properly, issue a checknode command on one of the nodes that Supermon is gathering statistics for. The output should look similar to below.

```
node keche
State:      Idle  (in current state for 00:32:43)
Configured Resources: PROCS: 2  MEM: 1003M  SWAP: 3353M  DISK: 1M
Utilized   Resources: ---
Dedicated  Resources: ---
Generic Metrics:  CPULOAD=33.38,NETIN=11749.00,NETOUT=9507.00,NETUSAGE=21256.00
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   0.500
Network Load: 0.87 kB/s
```

```
Flags:      rmdetected
Network:    DEFAULT
Classes:    [batch 2:2][interactive 2:2]
RM[TORQUE]: TYPE=PBS
EffNodeAccessPolicy: SHARED
Total Time: 2:03:27  Up: 2:03:27 (100.00%)  Active: 00:00:00 (0.00%)
Reservations:  ---
```

Configuring Resource Types

Native Resource managers can also perform special tasks when they are given a specific resource type. These types are specified using the RESOURCETYPE attribute of the RMCFG parameter.

Type	Description
COMPUTE	Normal compute resources (no special handling)
FS	File system resource manager (see Multiple Resource Managers for an example)
LICENSE	Software license manager (see Interfacing with FLEXlm and License Management)
NETWORK	Network resource manager
PROV	Provisioning resource manager. This is the RM Moab uses to modify the OS of a node (not a VM) and to power a node on or off.

Creating New Tools to Manage the Cluster

Using the scripts found in the `$TOOLSDIR` (`$INSTDIR/tools`) directory as a template, new tools can be quickly created to monitor or manage most any resource. Each tool should be associated with a particular resource manager service and specified using one of the following resource manager URL attributes.

CLUSTERQUERYURL	
Description	Queries resource state, configuration, and utilization information for compute nodes, networks, storage systems, software licenses, and other resources. For more details, see RM configuration.
Output	Node status and configuration for one or more nodes. See Resource Data Format.
Example	`RMCFG[v-stor] CLUSTERQUERYURL=exec://$HOME/storquery.pl` *Moab will execute the `storquery.pl` script to obtain information about 'v-stor' resources.*

JOBCANCELURL

Description	Specifies how Moab cancels jobs via the resource manager. For more details, see RM configuration.
Input	*<protocol>*://[*<host>*[:*<port>*]][*<path>*]
Example	`RMCFG[base] JOBCANCELURL=exec:///opt/moab/job.cancel.lsf.pl` *Moab executes `/opt/moab/job.cancel.lsf.pl` to cancel specific jobs.*

JOBMODIFYURL

Description	Modifies a job or application. For more details, see RM configuration.
Input	[-j *<JOBEXPR>*] [--s[et]|--c[lear]|--i[ncrement]|--d[ecrement]] *<ATTR>*[=*<VALUE>*] [*<ATTR>*[=*<VALUE>*]]...
Example	`RMCFG[v-stor] JOBMODIFYURL=exec://$HOME/jobmodify.pl` *Moab will execute the `jobmodify.pl` script to modify the specified job.*

JOBREQUEUEURL

Description	Requeues a job.
Input	*<JOBID>*
Example	`RMCFG[v-stor] JOBREQUEUEURL=exec://$HOME/requeue.pl` *Moab will execute the `requeue.pl` script to requeue jobs.*

JOBRESUMEURL

Description	Resumes a suspended job or application.

JOBRESUMEURL

Input	*<JOBID>*
Example	`RMCFG[v-stor] JOBRESUMEURL=exec://$HOME/jobresume.pl` *Moab will execute the `jobresume.pl` script to resume suspended jobs.*

JOBSTARTURL

Description	Launches a job or application on a specified set of resources.
Input	*<JOBID><TASKLIST><USERNAME>* [ARCH=*<ARCH>*] [OS=*<OPSYS>*] [IDATA=*<STAGEINFILEPATH>*[,*<STAGEINFILEPATH>*]...] [EXEC=*<EXECUTABLEPATH>*]
Example	`RMCFG[v-stor] JOBSTARTURL=exec://$HOME/jobstart.pl` *Moab will execute the `jobstart.pl` script to execute jobs.*

JOBSUBMITURL

Description	Submits a job to the resource manager, but it does not execute the job. The job executes when the **JOBSTARTURL** is called.
Input	[ACCOUNT=*<ACCOUNT>*] [ERROR=*<ERROR>*] [GATTR=*<GATTR>*] [GNAME=*<GNAME>*] [GRES=*<GRES>*:*<Value>*[,*<GRES>*:*<Value>*]*] [HOSTLIST=*<HOSTLIST>*] [INPUT=*<INPUT>*] [IWD=*<IWD>*] [NAME=*<NAME>*] [OUTPUT=*<OUTPUT>*] [RCLASS=*<RCLASS>*] [REQUEST=*<REQUEST>*] [RFEATURES=*<RFEATURES>*] [RMFLAGS=*<RMFLAGS>*] [SHELL=*<SHELL>*] [TASKLIST=*<TASKLIST>*] [TASKS=*<TASKS>*] [TEMPLATE=*<TEMPLATE>*] [UNAME=*<UNAME>*] [VARIABLE=*<VARIABLE>*] [WCLIMIT=*<WCLIMIT>*] [ARGS=*<Value>*[*<Value>*]*] ARGS must be the last submitted attribute because there can be multiple space-separated values for ARGS.
Example	`RMCFG[v-stor] JOBSUBMITURL=exec://$HOME/jobsubmit.pl` *Moab submits the job to the `jobsubmit.pl` script for future job execution.*

JOBSUSPENDURL	
Description	Suspends in memory an active job or application.
Input	*<JOBID>*
Example	`RMCFG[v-stor] JOBSUSPENDURL=exec://$HOME/jobsuspend.pl` *Moab will execute the `jobsuspend.pl` script to suspend active jobs.*

NODEPOWERURL	
Description	Allows Moab to issue IPMI power commands.
Input	*<NODEID>*[,*<NODEID>*] ON | OFF
Example	`RMCFG[node17rm] NODEPOWERURL=exec://$TOOLSDIR/ipmi.power.pl` *Moab will issue a power command contained in the `ipmi.power.pl`script.*

SYSTEMMODIFYURL	
Description	Provide method to dynamically modify aspects of the compute environment which are directly associated with cluster resources. For more details, see RM configuration.

SYSTEMQUERYURL	
Description	Provide method to dynamically query aspects of the compute environment which are directly associated with cluster resources. For more details, see RM configuration.
Input	default *<ATTR>* ATTR is one of `images`
Output	*<STRING>*

SYSTEMQUERYURL	
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 is associated with managed resources. Job/workload information should be reported back from the URL (script, file, webservice, etc.) using the Moab RM language (formerly WIKI). For more details, see RM configuration.
Output:	*<STRING>*
Example:	`RMCFG[xt] WORKLOADQUERYURL=exec://$HOME/job.query.xt3.pl` *Moab will load job/workload information by executing the `job.query.xt3.pl` script.*

Related topics

- mdiag -R command (evaluate resource managers)
- License Management
- Moab Resource Manager Language Data Format
- Managing Resources with SLURM

12.6 Utilizing Multiple Resource Managers

Multi-RM Overview

In many instances a site may have certain resources controlled by different resource managers. For example, a site may use a particular resource manager for licensing software for jobs, another resource manager for managing file systems, another resource manager for job control, and another for node monitoring. Moab can be configured to communicate with each of these resource managers, gathering all their data and incorporating such into scheduling decisions. With a more distributed approach to resource handling, failures are more contained and scheduling decisions can be more intelligent.

Configuring Multiple Independent Resource Manager Partitions

Moab must know how to communicate with each resource manager. In most instances, this is simply done by configuring a query command.

Migrating Jobs between Resource Managers

With multi-resource manager support, a job may be submitted either to a local resource manager queue or to the Moab global queue. In most cases, submitting a job to a resource manager queue constrains the job to only run within the resources controlled by that resource manager. However, if the job is submitted to the Moab global queue, it can use resources of any active resource manager. This is accomplished through job translation and staging.

When Moab evaluates resource availability, it determines the cost in terms of both data and job staging. If staging a job's executable or input data requires a significant amount of time, Moab integrates data and compute resource availability to determine a job's earliest potential start time on a per resource manager basis and makes an optimal scheduling decision accordingly. If the optimal decision requires a data stage operation, Moab reserves the required compute resources, stages the data, and then starts the job when the required data and compute resources are available.

Aggregating Information into a Cohesive Node View

Using the native interface, Moab can actually perform most of these functions without the need for an external resource manager. First, configure the native resource managers:

```
RMCFG[base]    TYPE=PBS
RMCFG[network] TYPE=NATIVE
RMCFG[network] CLUSTERQUERYURL=/tmp/network.sh
RMCFG[fs]      TYPE=NATIVE
RMCFG[fs]      CLUSTERQUERYURL=/tmp/fs.sh
```

The network script can be as simple as the following:

```
> _RX=`/sbin/ifconfig eth0 | grep "RX by" | cut -d: -f2 | cut -d' ' -f1`; \
> _TX=`/sbin/ifconfig eth0 | grep "TX by" | cut -d: -f3 | cut -d' ' -f1`; \
> echo `hostname` GMETRIC[netusage]=`echo "$_RX + $_TX" | bc`;
```

The preceding script would output something like the following:

```
node01 GMETRIC[netusage]=10928374
```

Moab grabs information from each resource manager and includes its data in the final view of the node.

```
> checknode node01
node node01
State:   Running  (in current state for 00:00:20)
Configured Resources: PROCS: 2  MEM: 949M  SWAP: 2000M  disk: 1000000
Utilized   Resources: SWAP: 9M
Dedicated  Resources: PROCS: 1  disk: 1000
Opsys:      Linux-2.6.5-1.358  Arch:       linux
Speed:      1.00  CPULoad:       0.320
Location:   Partition: DEFAULT  Rack/Slot:  NA
Network Load: 464.11 b/s
Network:    DEFAULT
```

```
Features:   fast
Classes:    [batch 1:2][serial 2:2]
Total Time: 00:30:39  Up: 00:30:39 (100.00%)  Active: 00:09:57 (32.46%)
Reservations:
  Job '5452'(x1)  -00:00:20 -> 00:09:40 (00:10:00)
JobList:  5452
```

Notice that the Network Load is now being reported along with disk usage.

Example File System Utilization Tracker (per user)

The following configuration can be used to track file system usage on a per user basis:

```
.....
RMCFG[file]    TYPE=NATIVE
RMCFG[file]    RESOURCETYPE=FS
RMCFG[file]    CLUSTERQUERYURL=/tmp/fs.pl
.....
```

Assuming that */tmp/fs.pl* outputs something of the following format:

```
DEFAULT STATE=idle AFS=<fs id="user1" size="789456"></fs><fs
id="user2" size="123456"></fs>
```

This will track disk usage for users *user1* and *user2* every 24 hours.

12.7 License Management

- License Management Overview
- Controlling and Monitoring License Availability
- Requesting Licenses w/in Jobs

License Management Overview

Software license management is typically enabled in one of two models: node-locked and floating. Under a node-locked license, use of a given application is constrained to certain hosts. For example, `node013` may support up to two simultaneous jobs accessing application matlab. In a floating license model, a limited number of software licenses are made available cluster wide, and these licenses may be used on any combination of compute hosts. In each case, these licenses are consumable and application access is denied once they are gone.

Moab supports both node-locked and floating license models and even allows mixing the two models simultaneously. Moab monitors license usage and only launches an application when required software license availability is guaranteed. In addition, Moab also reserves licenses in conjunction with future jobs to ensure these jobs can run at the appropriate time.

By default, Moab supports up to 128 independent license types.

Moab license recognition is case insensitive. This means that two licenses with the same spelling and different capitalization are still recognized as the same license. When this occurs, Moab considers the license invalid.

Controlling and Monitoring License Availability

Moab can use one of three methods to determine license availability. These methods include locally specifying consumable generic resources, obtaining consumable generic resource information from the resource manager, and interfacing directly with a license manager.

Local Consumable Resources

Both node-locked and floating licenses can be locally specified within Moab using the NODECFG parameter. In all cases, this is accomplished by associating the license with a node using the GRES (or generic resource) attribute. If floating, the total cluster-wide license count should be associated with the GLOBAL node. If node-locked, the per node license count should be associated with each compute host (or globally using the *DEFAULT* node). For example, if a site has two node-locked licenses for application *EvalA* and six floating licenses for application *EvalB*, the following configuration could be used:

```
NODECFG[node001]  GRES=EvalA:2
NODECFG[node002]  GRES=EvalA:2
NODECFG[GLOBAL]   GRES=EvalB:6
...
```

Resource Manager Based Consumable Resources

Some resource managers support the ability to define and track generic resource usage at a per node level. In such cases, support for node-locked licenses may be enabled by specifying this information within the resource manager. Moab automatically detects and schedules these resources. For example, in the case of TORQUE, this can be accomplished by adding generic resource specification lines to the MOM configuration file.

Interfacing to an External License Manager

Moab may also obtain live software license information from a running license manager. Direct interfaces to supported license managers such as FlexLM may be created using the Native Resource Manager feature. A complete example on interfacing to an external license manager is provided in the FLEXlm section of the native resource manager overview.

Interfacing to Multiple License Managers

Moab may interface to multiple external license managers simultaneously simply by defining additional native resource manager interfaces. See the FLEXlm Native Resource Manager Overview for more information.

Requesting Licenses within Jobs

Requesting use of software licenses within jobs is typically done in one of two ways. In most cases, the native resource manager job submission language provides a direct method of license specification; for example, in the case of TORQUE, OpenPBS, or PBSPro, the software argument could be specified using the format `<SOFTWARE_NAME>[+<LICENSE_COUNT>]` as in the following example:

```
> qsub -l nodes=2,software=blast cmdscript.txt
```

The license count is a job total, not a per task total, and the license count value defaults to 1.

An alternative to direct specification is the use of the Moab resource manager extensions. With these extensions, licenses can be requested as generic resources, using the GRES attribute. The job in the preceding example could also be requested using the following syntax:

```
> qsub -l nodes=2 -W x=GRES:blast cmdscript.txt
```

In each case, Moab automatically determines if the software licenses are node-locked or floating and applies resource requirements accordingly.

If a job requires multiple software licenses, whether of the same or different types, a user would use the following syntax:

```
> qsub -l nodes=2 -W x=GRES:blast+2 cmdscript.txt   # two 'blast' licenses required
> qsub -l nodes=2 -W x=GRES:blast+2%bkeep+3 cmdscript.txt   # two 'blast' and three
'bkeep' licenses are required
```

Related topics

- Native Resource Manager License Configuration
- License Ownership with Advance Reservations

12.8 Resource Provisioning

- Resource Provisioning Overview
- Configuring Provisioning

Resource Provisioning Overview

When processing a resource request, Moab attempts to match the request to an existing available resource. However, if the scheduler determines that the resource is not available or will not be available due to load or policy for an appreciable amount of time, it can select a resource to modify to meet the needs of the current requests. This process of modifying resources to meet existing needs is called provisioning.

Currently, there are two types of provisioning supported: operating system (OS) and application. As its name suggests, OS provisioning allows the scheduler to modify the operating system of an existing compute node while application level provisioning allows the scheduler to request that a software application be made available on a given compute node. In each case, Moab evaluates the costs of making the change in terms of time and other resources consumed before making the decision. Only if the benefits outweigh the costs will the scheduler initiate the change required to support the current workload.

Preemption (requeueing) does not work with dynamic provisioning.

Configuring Provisioning

Enabling provisioning consists of configuring an interface to a provisioning manager, specifying which nodes can take advantage of this service, and what the estimated cost and duration of each change will be. This interface can be used to contact provisioning software such as xCat or HP's Server Automation tool. Additionally, locally developed systems can be interfaced via a script or web service.

Related topics

- Native Resource Manager Overview
- Appendix O: Resource Manager Integration

12.9 Intelligent Platform Management Interface

- IPMI Overview
- Node IPMI Configuration
- Installing IPMItool
- Setting-up the BMC-Node Map File
- Configuring Moab's IPMI Tools
- Configuring Moab
- Ensuring Proper Setup

IPMI Overview

The Intelligent Platform Management Interface (IPMI) specification defines a set of common interfaces system administrators can use to monitor system health and manage the system. The IPMI interface can monitor temperature and other sensor information, query platform status and power-on/power-off compute nodes. As IPMI operates independently of the node's OS interaction with the node can happen even when powered down. Moab can use IPMI to monitor temperature information, check power status, power-up, power-down, and reboot compute nodes.

Node IPMI Configuration

IPMI must be enabled on each node in the compute cluster. This is usually done either through the node's BIOS or by using a boot CD containing IPMI utilities provided by the manufacturer. With regard to configuring IPMI on the nodes, be sure to enable IPMI-over-LAN and set a common login and password on all the nodes. Additionally, you must set a unique IP address for each node's BMC. Take note of these addresses as you will need them when reviewing the Creating the IPMI BMC-Node Map File section.

Installing IPMItool

IPMItool is an open-source tool used to retrieve sensor information from the IPMI Baseboard Management Controller (BMC) or to send remote chassis power control commands. The IPMItool developer provides Fedora Core binary packages as well as a source tarball on the IPMItool download

page. Download and install IPMItool on the Moab head node and make sure the `ipmitool` binary is in the current shell PATH.

Proper IPMI setup and IPMItool configuration can be confirmed by issuing the following command on the Moab head node.

```
> ipmitool -I lan -U username -P password -H BMC IP chassis status
```

The output of this command should be similar to the following.

```
System Power         : off
Power Overload       : false
Power Interlock      : inactive
Main Power Fault     : false
Power Control Fault  : false
Power Restore Policy : previous
Last Power Event     :
Chassis Intrusion    : inactive
Front-Panel Lockout  : inactive
Drive Fault          : false
Cooling/Fan Fault    : false
```

Creating the IPMI BMC-Node Map File [OPTIONAL]

Since the BMC can be controlled via LAN, it is possible for the BMC to have its own unique IP address. Since this IP address is separate from the IP address of the node, a simple mapping file is required for Moab to know each node's BMC address. The file is a flat text file and should be stored in the Moab home directory. If a mapping file is needed, specify the name in the `config.ipmi.pl` configuration file in the `etc/` directory. The following is an example of the mapping file:

```
#<NodeID> <BMC IP>
node01  10.10.10.101
node02  10.10.10.102
node03  10.10.10.103
node04  10.10.10.104
node05  10.10.10.105
# NodeID = the name of the nodes returned with "mdiag -n"
# BMC IP = the IP address of the IPMI BMC network interface
```

Note that only the nodes specified in this file are queried for IPMI information. Also note that the mapping file is disabled by default and the nodes that are returned from Moab with mdiag -n are the ones that are queried for IPMI sensor data.

Configuring the Moab IPMI Tools

The `tools/` subdirectory in the install directory already contains the Perl scripts needed to interface with IPMI. The following is a list of the Perl scripts that should be in the `tools/` directory; confirm these are present and executable.

```
ipmi.mon.pl     # The daemon front-end called by Moab
ipmi.power.pl   # The power control script called by Moab
__mon.ipmi.pl   # The IPMI monitor daemon that updates and caches IPMI data from nodes
```

Next, a few configuration settings need to be adjusted in the `config.ipmi.pl` file found in the `etc` subdirectory. The IPMI-over-LAN username and password need to be set to the values that were set in

the Node IPMI Configuration section. Also, the IPMI query daemon's polling interval can be modified by adjusting $pollInterval. This specifies how often the IPMI-enabled nodes are queried to retrieve sensor data.

Configuring Moab

To allow Moab to use the IPMI tools, a native resource manager is configured. To do this, the following lines must be added to `moab.cfg`:

```
...
# IPMI - Node monitor script
RMCFG[ipminative] TYPE=NATIVE CLUSTERQUERYURL=exec://$TOOLSDIR/ipmi.mon.pl
...
```

Next, the following lines can be added to allow Moab to issue IPMI power commands.

```
...
# IPMI - Power on/off/reboot script
RMCFG[ipminative] NODEPOWERURL=exec://$TOOLSDIR/ipmi.power.pl
...
```

Moab can be configured to perform actions based on sensor data. For example, Moab can shut down a compute node if its CPU temperature exceeds 100 degrees Celsius, or it can power down idle compute nodes if workload is low. Generic event thresholds are used to tell Moab to perform certain duties given certain conditions. The following example is of a way for Moab to recognize it should power off a compute node if its CPU0 temperature exceeds 100 degrees Celsius.

```
...
# IPMI - Power off compute node if its CPU0 temperature exceeds 100 degrees Celsius.
GEVENTCFG[CPU0_TEMP>100] action=off
...
```

Ensuring Proper Setup

Once the preceding steps have been taken, Moab should be started as normal. The IPMI monitoring daemon should start automatically, which can be confirmed with the following:

```
moab@headnode:~/$ ps aux | grep __mon
moab   11444  0.0  0.3   6204  3172 pts/3    S    10:54   0:00 /usr/bin/perl -w
/opt/moab/tools/_mon.ipmi.pl --start
```

After a few minutes, IPMI data should be retrieved and cached. This can be confirmed with the following command:

```
moab@headnode:~/$ cat spool/ipmicache.gm
node01 GMETRIC[CPU0_TEMP]=49
node01 GMETRIC[CPU1_TEMP]=32
node01 GMETRIC[SYS_TEMP]=31
node01 POWER=ON
```

Finally, issue the following to ensure Moab is grabbing the IPMI data. Temperature data should be present in the `Generic Metrics` row.

```
moab@headnode:~/$ checknode node01
```

```
node node01
State:      Idle  (in current state for 00:03:12)
Configured Resources: PROCS: 1  MEM: 2000M  SWAP: 3952M  DISK: 1M
Utilized   Resources: ---
Dedicated  Resources: ---
Generic Metrics:  CPU0_TEMP=42.00,CPU1_TEMP=30.00,SYS_TEMP=29.00
...
```

12.10 Resource Manager Translation

- Translation Overview
- Translation Enablement Steps

Translation Overview

Resource manager translation allows end-users to continue to use existing job command scripts and familiar job management and resource query commands. This is accomplished by emulating external commands, routing the underlying queries to Moab, and then formatting the responses in a familiar manner. Using translation, job submission clients, job query clients, job control clients, and resource query clients can be emulated making switching from one resource manager to another transparent and preserving investment in legacy scripts, tools, and experience.

Translation Enablement Steps

To enable translation, you must:

- Edit the Moab tools configuration file.
- Copy, rename, and link the emulation scripts to a shorter, easier-to-use name.

Configure Translation Tools

Located in the `$MOABHOMEDIR/etc` directory are tools-specific configuration files. For each resource manager that has installed translation tools, edit the Moab tools configuration file in the `etc` directory. For example, if enabling LSF translation, do the following:

```
> vi $MOABHOMEDIR/etc/config.moab.pl
# Set the PATH to include directories for moab client commands — mjobctl, etc.
$ENV{PATH} = "/opt/moab/bin:$ENV{PATH}";
```

Add Translation Tools

In a directory accessible to users, create links to (or copy) the emulation scripts you want your users to use. For example, the emulation script `tools/bjobs.lsf.pl` could be copied to `bin/bjobs`, or, a symbolic link could be created in `bin/bjobs` that points to `tools/bjobs.lsf.pl`.

```
> ln -s tools/bjobs.lsf.pl bin/bjobs
> ln -s tools/bhosts.lsf.pl bin/bhosts
```

13.0 Troubleshooting and System Maintenance

- **Internal Diagnostics/Diagnosing System Behavior and Problems** on page 585
- **Logging Facilities** on page 588
- **Object Messages** on page 597
- **Notifying Administrators of Failures** on page 599
- **Issues with Client Commands** on page 600
- **Tracking System Failures** on page 601
- **Problems with Individual Jobs** on page 603
- **Diagnostic Scripts** on page 604

13.1 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.

The mdiag Command

The cornerstone of Moab's diagnostics is the mdiag command. This command provides detailed information about scheduler state and also performs a large number of internal sanity checks presenting problems it finds as warning messages.

Currently, the mdiag command provides in-depth analysis of the following objects and subsystems:

Object/Subsystem	mdiag Flag	Use
Account	-a	Shows detailed account configuration information.
Blocked	-b	Indicates why blocked (ineligible) jobs are not allowed to run.
Class	-c	Shows detailed class configuration information.
Config	-C	Shows configuration lines from `moab.cfg` and whether or not they are valid.
FairShare	-f	Shows detailed fairshare configuration information as well as current fair-share usage.
Group	-g	Shows detailed group information.
Job	-j	Shows detailed job information. Reports corrupt job attributes, unexpected states, and excessive job failures.
Frame/Rack	-m	Shows detailed frame/rack information.
Node	-n	Shows detailed node information. Reports unexpected node states and resource allocation conditions.
Priority	-p	Shows detailed job priority information including priority factor contributions to all idle jobs.
QoS	-q	Shows detailed QoS information.
Reservation	-r	Shows detailed reservation information. Reports reservation corruption and unexpected reservation conditions.
Resource Manager	-R	Shows detailed resource manager information. Reports configured and detected state, configuration, performance, and failures of all configured resource manager interfaces.

Object/Subsystem	mdiag Flag	Use
Standing Reservations	-s	Shows detailed standing reservation information. Reports reservation corruption and unexpected reservation conditions.
Scheduler	-S	Shows detailed scheduler state information. Indicates if scheduler is stopped and identifies and reports high-level scheduler failures.
Partition	-t	Shows detailed partition information.
User	-u	Shows detailed user information.

Other Diagnostic Commands

Beyond mdiag, the checkjob and checknode commands also provide detailed information and sanity checking on individual jobs and nodes respectively. These commands can indicate why a job cannot start, which nodes can be available, and information regarding the recent events impacting current job or nodes state.

Using Moab Logs for Troubleshooting

Moab logging is extremely useful in determining the cause of a problem. Where other systems may be cursed for not providing adequate logging to diagnose a problem, Moab may be cursed for the opposite reason. If the logging level is configured too high, huge volumes of log output may be recorded, potentially obscuring the problems in a flood of data. Intelligent searching combined with the use of the LOGLEVEL and LOGFACILITY parameters can mine out the needed information. Key information associated with various problems is generally marked with the keywords WARNING, ALERT, or ERROR. See the Logging Overview for further information.

Automating Recovery Actions after a Failure

The RECOVERYACTION parameter of SCHEDCFG can be used to control scheduler action in the case of a catastrophic internal failure. Valid actions include die, ignore, restart, and trap.

Recovery Mode	Description
die	Moab will exit and, if core files are externally enabled, will create a core file for analysis (This is the default behavior.).
ignore	Moab will ignore the signal and continue processing. This may cause Moab to continue running with corrupt data which may be dangerous. Use this setting with caution.

Recovery Mode	Description
restart	When a SIGSEGV is received, Moab will relaunch using the current checkpoint file, the original launch environment, and the original command line flags. The receipt of the signal will be logged but Moab will continue scheduling. Because the scheduler is restarted with a new memory image, no corrupt scheduler data should exist. One caution with this mode is that it may mask underlying system failures by allowing Moab to overcome them. If used, the event log should be checked occasionally to determine if failures are being detected.
trap	When a SIGSEGV is received, Moab stays alive but enters diagnostic mode. In this mode, Moab stops scheduling but responds to client requests allowing analysis of the failure to occur using internal diagnostics available via the mdiag command.

Related topics

- Troubleshooting Individual Jobs

13.2 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.

- Log Facility Configuration
- Status Information
- Scheduler Warnings
- Scheduler Alerts
- Scheduler Errors
- Searching Moab Logs
- Event Logs
    - Event Log Format
    - Exporting Events in Real-Time
- Sending event logs to Moab Web Services for storage
- Enabling Syslog
- Managing Log Verbosity

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 13-1:

```
# moab.cfg
 # allow up to 30 100MB logfiles
 LOGLEVEL         5
 LOGDIR           /var/tmp/moab
 LOGFILEMAXSIZE   100000000
 LOGFILEROLLDEPTH 30
```

The logging that occurs is of the following major types: subroutine information, status information, scheduler warnings, scheduler alerts, and scheduler errors.

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)
```

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'
```

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
```

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
```

Searching Moab Logs

While major failures are reported via the mdiag -S command, these failures can also be uncovered by searching the logs using the grep command as in the following:

```
> grep -E "WARNING|ALERT|ERROR" moab.log
```

On a production system working normally, this list usually includes some ALERT and WARNING messages. The messages are usually self-explanatory, but if not, viewing the log can give context to the message.

If a problem is occurring early when starting the Moab scheduler (before the configuration file is read) Moab can be started up using the `-L <LOGLEVEL>`flag. If this is the first flag on the command line, then the **LOGLEVEL** is set to the specified level immediately before any setup processing is done and additional logging is recorded.

If problems are detected in the use of one of the client commands, the client command can be re-issued with the `--loglevel=<LOGLEVEL>` command line argument specified. This argument causes log information to be written to STDERR as the client command is running. As with the server, <LOGLEVEL> values from 0 to 9 are supported.

The **LOGLEVEL** can be changed dynamically by use of the mschedctl -m command, or by modifying the `moab.cfg` file and restarting the scheduler. Also, if the scheduler appears to be hung or is not properly responding, the log level can be incremented by one by sending a *SIGUSR1* signal to the scheduler

process. Repeated *SIGUSR1*signals continue to increase the log level. The *SIGUSR2* signal can be used to decrease the log level by one.

If an unexpected problem does occur, save the log file as it is often very helpful in isolating and correcting the problem.

Event Logs

Major events are reported to both the Moab log file as well as the Moab event log. By default, the event log is maintained in the statistics directory and rolls on a daily basis, using the naming convention `events.WWW_MMM_DD_YYYY` as in `events.Tue_Mar_18_2008`.

Event Log Format

The event log contains information about major job, reservation, node, and scheduler events and failures and reports this information in the following format:

```
<EVENTTIME> <EPOCHTIME>:<EID> <OBJECT> <OBJECTID> <EVENT> <DETAILS>
```

Example 13-2:

```
VERSION 500
07:03:21 110244322:0 sched clusterA   start
07:03:26 110244327:1 rsv   system.1   start   1124142432 1324142432 2 2 0.0 2342155.3
node1|node2 NA RSV=%=system.1=
07:03:54 110244355:2 job   1413       end     8 16 llw mcc 432000 Completed [batch:1]
11 08708752 1108703981 ...
07:04:59 110244410:3 rm    base       failure cannot connect to RM
07:05:20 110244431:4 sched clusterA   stop    admin
...
```

The parameter RECORDEVENTLIST can be used to control which events are reported to the event log. See the sections on job and reservation trace format for more information regarding the values reported in the details section for those records.

Record Type Specific Details Format

The format for each record type is unique and is described in the following table:

Record Type	Event Types	Description
gevent	See Enabling Generic Events for gevent information.	Generic events are included within node records. See node detail format that follows.

Record Type	Event Types	Description
job	*JOBCANCEL*, *JOBCHECKPOINT*, *JOBEND*, *JOBHOLD*, *JOBMIGRATE*, *JOBMODIFY*, *JOBPREEMPT*, *JOBREJECT*, *JOBRESUME*, *JOBSTART*, *JOBSUBMIT*	See Workload Accounting Records.
node	*NODEDOWN*, *NODEFAILURE*, *NODEUP*	The following fields are displayed in the event file in a space-delimited line as long as Moab has information pertaining to it: state, partition, disk, memory, maxprocs, swap, os, rm, nodeaccesspolicy, class, and message, where state is the node's current state and message is a human readable message indicating reason for node state change.
rm	*RMDOWN*, *RMPOLLEND*, *RMPOLLSTART*, *RMUP*	Human readable message indicating reason for resource manager state change. For *SCHEDCOMMAND*, only create/modify commands are recorded. No record is created for general list/query commands. *ALLSCHEDCOMMAND* does the same thing as *SCHEDCOMMAND*, but it also logs info query commands.
trigger	*TRIGEND*, *TRIGFAILURE*, *TRIGSTART*	*<ATTR>*="*<VALUE>*"[*<ATTR>*="*<VALUE>*"]... where *<ATTR>* is one of the following: actiondata, actiontype, description, ebuf, eventtime, eventtype, flags, name, objectid, objecttype, obuf, offset, period, requires, sets, threshold, timeout, and so forth. See **About object triggers** on page 655 for more information.
vm	*VMCREATE*, *VMDESTROY*, *VMMIGRATE*, *VMPOWEROFF*, *VMPOWERON*	The following fields are displayed in the event file in a space-delimited line as long as Moab has information pertaining to it: name, sovereign, powerstate, parentnode, swap, memory, disk, maxprocs, opsys, class, and variables, where class and variables may have 0 or multiple entries.

Exporting Events in Real-Time

Moab event information can be exported to external systems in real-time using the ACCOUNTINGINTERFACEURL parameter. When set, Moab activates this URL each time one of the default events or one of the events specified by the RECORDEVENTLIST occurs.

While various protocols can be used, the most common protocol is `exec`, which indicates that Moab should launch the specified tool or script and pass in event information as command line arguments. This

tool can then select those events and fields of interest and re-direct them as appropriate providing significant flexibility and control to the organization.

Exec Protocol Format

When a URL with an `exec` protocol is specified, the target is launched with the event fields passed in as STDIN. These fields appear exactly as they do in the event logs with the same values and order.

The `tools/sql` directory included with the Moab distribution contains `event.create.sql.pl`, a sample accounting interface processing script that may be used as a template.

Event logging with web services

Administrators can configure Moab to push event data to Moab Web Services or other web services. This allows you to manage and store event logs from a single location. Currently, Moab pushes the following events to web services for storage:

These event logs are separate from the old Moab event logs.

Event type	Facility	Category
jobcancel	job	cancel
jobend	job	end
jobhold	job	hold
jobmodify	job	modify
jobreject	job	reject
jobrelease	job	release
jobstart	job	start
jobsubmit	job	submit
rsvcreate	reservation	create
rsvend	reservation	end
rsvstart	reservation	start

Event type	Facility	Category
allschedcommand	scheduler	command
schedcommand	scheduler	command
schedcycleend	scheduler	end
schedcyclestart	scheduler	start
schedpause	scheduler	pause
schedrecycle	scheduler	recycle
schedresume	scheduler	resume
schedstart	scheduler	start
schedend	scheduler	end
trigcreate	trigger	create
trigend	trigger	end
trigstart	trigger	start
vmcancel	vm	cancel
vmdestroy	vm	destroy
vmend	vm	end
vmmigrateend	vm	migrate
vmmigratestart	vm	migrate
vmready	vm	ready
vmsubmit	vm	submit

The *SCHEDCOMMAND* and *ALLSCHEDCOMMAND* event type log information is not pushed to web services unless you have specified that you want to include it in the RECORDEVENTLIST parameter.

Event category	Description
cancel	Indicates the object was canceled.
command	Indicates that Moab received a command.
create	Indicates the object was created.
destroy	Indicates the object was destroyed. "end" can occur before "destroy".
end	Indicates the object ended normally (it reached its end of life; completed).
hold	Indicates the job had a hold placed on it.
migrate	Indicates a VM migration event.
modify	Indicates the object was modified.
pause	Indicates the scheduler paused.
ready	Indicates the object was ready. "submit" can occur before "ready".
recycle	Indicates the scheduler recycled.
reject	Indicates the object was rejected as invalid.
release	Indicates all holds have been removed from a job.
resume	Indicates the scheduler resumed. "pause" can occur before "resume".

Event category	Description
start	Indicates the object started. "submit" can occur before "start".
stop	Indicates the scheduler stopped.
submit	Indicates the object was submitted to Moab. (Note that this does not indicate that Moab accepted it.)

To enable Moab to push event logging to web services, you will need to set the following parameters in `moab.cfg`:

- Set PUSHEVENTSTOWEBSERVICE to *TRUE*.
- Use EVENTLOGWSURL to specify your web services event log URL.

And these parameters in the `moab-private.cfg` file:

- Use EVENTLOGWSUSER to specify the username required to log in to your web services.
- Use EVENTLOGWSPASSWORD to specify the user password required to log in to your web services.

For more information about Moab event logging in Moab Web Services, see the "Events" section of the Moab Web Services Reference Guide.

Enabling Syslog

In addition to the log file, the Moab scheduler can report events it determines to be critical to the UNIX syslog facility via the daemon facility using priorities ranging from `INFO` to `ERROR`. (See USESYSLOG). The verbosity of this logging is not affected by the LOGLEVEL parameter. In addition to errors and critical events, user commands that affect the state of the jobs, nodes, or the scheduler may also be logged to syslog. Moab syslog messages are reported using the INFO, NOTICE, and ERR syslog priorities.

By default, messages are logged to syslog's user facility. However, using the **USESYSLOG** parameter, Moab can be configured to use any of the following:

- *user*
- *daemon*
- *local0*
- *local1*
- *local2*
- *local3*
- *local4*

- *local5*
- *local6*
- *local7*

Managing Verbosity

In very large systems, a highly verbose log may roll too quickly to be of use in tracking specific targeted behaviors. In these cases, one or more of the following approaches may be of use:

- Use the LOGFACILITY parameter to log only functions and services of interest.
- Use syslog to maintain a permanent record of critical events and failures.
- Specify higher object loglevels on jobs, nodes, and reservations of interest (such as `NODECFG [orion13] LOGLEVEL=6`).
- Increase the range of events reported to the event log using the RECORDEVENTLIST parameter.
- Review object messages for required details.
- Run Moab in monitor mode using IGNOREUSERS, IGNOREJOBS, IGNORECLASSES, or IGNORENODES.

Related topics

- RECORDEVENTLIST parameter
- USESYSLOG parameter
- Notifying Admins
- Simulation Workload Trace Overview
- mschedctl -L command

13.3 Object Messages

Object Message Overview

Messages can be associated with the scheduler, jobs, and nodes. Their primary use is a line of communication between resource managers, the scheduler, and end-users. When a node goes offline, or when a job fails to run, both the resource manager and the scheduler will post messages to the object's message buffer, giving the administrators and end-users a reason for the failure. They can also be used as a way for different administrators and users to send messages associated with the various objects. For example, an administrator can set the message `Node going down for maintenance Apr/6/08 12pm," on node node01`, which would then be visible to other administrators.

Viewing Messages

To view messages associated with a job (either from users, the resource manager, or Moab), run the checkjob command.

To view messages associated with a node (either from users, the resource manager, or Moab), run the checknode command.

To view system messages, use the mschedctl -l message command.

To view the messages associated with a credential, run the mcredctl -c command.

Creating Messages

To create a message use the mschedctl -c message *<STRING>* `[-o <OBJECTTYPE>:<OBJECTID>] [-w <ATTRIBUTE>=<VALUE>[-w ...]]` command.

The *<OBJECTTYPE>* can be one of the following:

- node
- job
- rsv
- user
- acct
- qos
- class
- group

The *<ATTRIBUTE>* can be one of the following:

- owner
- priority
- expiretime
- type

Valid types include:

- annotation
- other
- hold
- pendactionerror

Deleting Messages

Deleting, or removing, messages is straightforward. The commands used depend on the type of object to which the message is attached:

- Scheduler: Use the "mschedctl -d message:*<INDEX>*" command (where INDEX is the index of the message you want to delete).

- Node: Use the mnodectl*<NODE>* -d message:*<INDEX>* command.

13.4 Notifying Administrators of Failures

Enabling Administrator Email

In the case of certain events, Moab can automatically send email to administrators. To enable mail notification, the MAILPROGRAM parameter must be set to *DEFAULT* or point to the locally available mail client. With this set, policies such as JOBREJECTPOLICY will send email to administrators if set to a value of *MAIL*.

Handling Events with the Notification Routine

Moab possesses a primitive event management system through the use of the notify program. The program is called each time an event of interest occurs. Currently, most events are associated with failures of some sort but use of this facility need not be limited in this way. The NOTIFICATIONPROGRAM parameter allows a site to specify the name of the program to run. This program is most often locally developed and designed to take action based on the event that has occurred. The location of the notification program may be specified as a relative or absolute path. If a relative path is specified, Moab looks for the notification relative to the `$(INSTDIR)/tools` directory. In all cases, Moab verifies the existence of the notification program at start up and disables it if it cannot be found or is not executable.

The notification program's action may include steps such as reporting the event via email, adjusting scheduling parameters, rebooting a node, or even recycling the scheduler.

For most events, the notification program is called with command line arguments in a simple *<EVENTTYPE>: <MESSAGE>* format. The following event types are currently enabled:

Event Type	Format	Description
JOBCORRUPTION	*<MESSAGE>*	An active job is in an unexpected state or has one or more allocated nodes that are in unexpected states.
JOBHOLD	*<MESSAGE>*	A job hold has been placed on a job.
JOBWCVIOLATION	*<MESSAGE>*	A job has exceeded its wallclock limit.
RESERVATIONCORRUPTION	*<MESSAGE>*	Reservation corruption has been detected.
RESERVATIONCREATED	*<RSVNAME> <RSVTYPE> <NAME> <PRESENTTIME> <STARTTIME> <ENDTIME> <NODECOUNT>*	A new reservation has been created.

Event Type	Format	Description
RESERVATIONDESTROYED	*<RSVNAME> <RSVTYPE> <PRESENTTIME> <STARTTIME> <ENDTIME> <NODECOUNT>*	A reservation has been destroyed.
RMFAILURE	*<MESSAGE>*	The interface to the resource manager has failed.

Perhaps the most valuable use of the notify program stems from the fact that additional notifications can be easily inserted into Moab to handle site specific issues. To do this, locate the proper block routine, specify the correct conditional statement, and add a call to the routine `notify(<MESSAGE>);`.

Related topics

- JOBREJECTPOLICY parameter
- MAILPROGRAM parameter
- Event Log Overview

13.5 Issues with Client Commands

- Client Overview
- Diagnosing Client Problems

Client Overview

Moab client commands are implemented as links to the executable `mclient`. When a Moab client command runs, the client executable determines the name under which it runs and behaves accordingly. At the time Moab was configured, a home directory was specified. The Moab client attempts to open the configuration file, `moab.cfg`, in the `etc/` folder of this home directory on the node where the client command executes. This means that the home directory specified at configure time must be available on all hosts where the Moab client commands are executed. This also means that a `moab.cfg` file must be available in the `etc/` folder of this home directory. When the clients open this file, they will try to load the **SCHEDCFG** parameter to determine how to contact the Moab server.

The home directory value specified at configure time can be overridden by creating an `/etc/moab.cfg` file or by setting the MOABHOMEDIR environment variable.

Once the client has determined where the Moab server is located, it creates a message, adds an encrypted checksum, and sends the message to the server. The Moab client and Moab server must use a shared secret key for this to work. When the Moab server receives the client request and verifies the message, it processes the command and returns a reply.

Diagnosing Client Problems

The easiest way to determine where client failures are occurring is to use built-in Moab logging. On the client side, use the `--loglevel` flag. For example:

```
> showq --loglevel=9
```

This will display verbose logging information regarding the loading of the configuration file, connecting to the Moab server, sending the request, and receiving a response.

This information almost always reveals the source of the problem. If it does not, the next step is to look at the Moab server side logs; this is done using the following steps:

- Stop Moab scheduling so that the only activity is handling Moab client requests.

  ```
  > mschedctl -s
  ```

- Set the logging level to *very* verbose.

  ```
  > mschedctl -m loglevel 7
  ```

- Watch Moab activity.

  ```
  > tail -f log/moab.log | more
  ```

Now, in a second window, issue any failing client command, such as showq.

The `moab.log` file will record the client request and any reasons it was rejected.

13.6 Tracking System Failures

System Failures

The scheduler has a number of dependencies that may cause failures if not satisfied. These dependencies are in the areas of disk space, network access, memory, and processor utilization.

Disk Space

The scheduler uses a number of files. If the file system is full or otherwise inaccessible, the following behaviors might be noted:

Unavailable File	Behavior
moab.pid	Scheduler cannot perform single instance check.
moab.ck*	Scheduler cannot store persistent record of reservations, jobs, policies, summary statistics, and so forth.

Unavailable File	Behavior
moab.cfg /moab.dat	Scheduler cannot load local configuration.
log/*	Scheduler cannot log activities.
stats/*	Scheduler cannot write job records.

When possible, configure Moab to use local disk space for configuration files, statistics files, and logs files. If any of these files are located in a networked file system (such as NFS, DFS, or AFS) and the network or file server experience heavy loads or failures, Moab server may appear sluggish or unresponsive and client command may fail. Use of local disk space eliminates susceptibility to this potential issue.

Network

The scheduler uses a number of socket connections to perform basic functions. Network failures may affect the following facilities.

Network Connection	Behavior
scheduler client	Scheduler client commands fail.
resource manager	Scheduler is unable to load/update information regarding nodes and jobs.
allocation manager	Scheduler is unable to validate account access or reserve/debit account balances.

Memory

Depending on cluster size and configuration, the scheduler may require up to 120 MB of memory on the server host. If inadequate memory is available, multiple aspects of scheduling may be negatively affected. The scheduler log files should indicate if memory failures are detected and mark any such messages with the ERROR or ALERT keywords.

Processor Utilization

On a heavily loaded system, the scheduler may appear sluggish and unresponsive. However, no direct failures should result from this slowdown. Indirect failures may include timeouts of peer services (such as the resource manager or allocation manager) or timeouts of client commands. All timeouts should be recorded in the scheduler log files.

Internal Errors

The Moab scheduling system contains features to assist in diagnosing internal failures. If the scheduler exits unexpectedly, the scheduler logs may provide information regarding the cause. If no reason can be determined, use of a debugger may be required.

Logs

The first step in any exit failure is to check the last few lines of the scheduler log. In many cases, the scheduler may have exited due to misconfiguration or detected system failures. The last few lines of the log should indicate why the scheduler exited and what changes would be required to correct the situation. If the scheduler did not intentionally exit, increasing the LOGLEVEL parameter to 7, or higher, may help isolate the problem.

Reporting Failures

If an internal failure is detected on your system, the information of greatest value to developers in isolating the problem will be the output of the gdb where subcommand and a printout of all variables associated with the failure. In addition, a level 7 log covering the failure can also help in determining the environment that caused the failure. If you encounter such and require assistance, please submit a ticket at the following address:

http://www.adaptivecomputing.com/services/techsupport.php

If you do not already have a support username and password, please create a free account to request a support ticket.

13.7 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.

Flag	Description
-l ***<POLICYLEVEL>***	Evaluates impact of throttling policies on job feasibility.
-n ***<NODENAME>***	Evaluates resource access on specific node.
-r ***<RESERVATION_LIST>***	Evaluates access to specified reservations.

checknode

Displays detailed status of node.

mdiag -b

Displays various reasons job is considered blocked or non-queued.

mdiag -j

Displays high level summary of job attributes and performs sanity check on job attributes/state.

showbf -v

Determines general resource availability subject to specified constraints.

13.8 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 Adaptive Computing support staff. This section introduces available diagnostic scripts.

The support.diag.pl Script

The `support.diag.pl` script has a two-fold purpose. First, it can be used by a Moab trigger or cron job to create a regular snapshot of the state of Moab. The script captures the output of several Moab diagnostic commands (such as showq, mdiag -n, and mdiag -S), gathers configuration/log files, and records pertinent operating system information. This data is then compressed in a time-stamped tarball for easy long-term storage.

The second purpose of the `support.diag.pl` script is to provide Adaptive Computing support personnel with a complete package of information that can be used to help diagnose configuration issues or system bugs. After capturing the state of Moab, the resulting tarball could be sent to your Adaptive Computing support contact for further diagnosis.

The `support.diag.pl` will ask you for the trouble ticket number then guide you through the process of uploading the data to Adaptive Computing Customer Support. The uploading and ticket number request may be prevented using the `--no-upload` and `--support-ticket=<SUPPORT_TICKET_ID>` flags detailed in the Arguments table that follows.

Synopsis

`support.diag.pl [--include-log-lines=<NUM>] [--diag-torque]`

Arguments

Argument	Description
`--include-log-lines=<NUM>`	Instead of including the entire `moab.log` file, only the last *<NUM>* lines are captured in the diagnostics.
`--diag-torque`	Diagnostic commands pertinent to the TORQUE resource manager are included.

Argument	Description
--no-upload	Prevents the system from asking the user if they want to upload the tarball to Adaptive Computing Customer Support.
--support-ticket=*<SUPPORT_TICKET_ID>*	Prevents the system from asking the user for a support ticket number.

14.0 Improving User Effectiveness

- **User Feedback Loops** on page 607
- **User Level Statistics** on page 608
- **Enhancing Wallclock Limit Estimates** on page 608
- **Job Start Time Estimates** on page 608
- **Providing Resource Availability Information** on page 610
- **Collecting Performance Information on Individual Jobs** on page 610

14.1 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.

Improving Job Size/Duration Requests

Moab provides a number of informational commands that help users make improved job management decisions based on real-time cluster wide resource availability information. These commands include showbf, showstats -f, and showq. Using these commands, a user can determine what resources are available and what job configurations statistically receive the best scheduling performance.

Improving Resource Requirement Specification

A job's resource requirement specification tells the scheduler what type of compute nodes are required to run the job. These requirements may state that a certain amount of memory is required per node or that a node has a minimum processor speed. At many sites, users will determine the resource requirements needed to run an initial job. Then, for the next several years, they will use the same basic batch command file to run all of their remaining jobs even though the resource requirements of their

subsequent jobs may be very different from their initial run. Users often do not update their batch command files even though these constraints may be unnecessarily limiting the resources available to their jobs for two reasons: (1) users do not know how much their performance will improve if better information were provided and (2) users do not know exactly what resources their jobs are using and are afraid to lower their job's resource requirements since doing so might cause their job to fail.

To help with determining accurate per job resource utilization information, Moab provides the FEEDBACKPROGRAM facility. This tool allows sites to send detailed resource utilization information back to users via email, to store it in a centralized database for report preparation, or use it in other ways to help users refine their batch jobs.

14.2 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     %   FSTgt  AvgXF
MaxXF  AvgQH   Effic  WCAcc
john         1     1     30.96    9   0.00  300.0   0.00  148.9   0.00 -----   0.62
0.00   4.33 100.00  48.87
```

Users can query available system resources with the showbf command. This can aid users in requesting node configurations that are idle. Also, users can use the checkjob command to determine what parameter(s) are restricting their job from running. Moab performs better with more accurate wallclock estimates.

Moab must use an ODBC-compliant database to report statistics with Viewpoint reports.

14.3 Enhancing Wallclock Limit Estimates

As explained in the previous section, showstats -u *<USER_ID>* reports statistics for a given user. The showstats -u command can be accessed by all users. They can use fields such as `PHReq`, `PHDed`, or `WCAcc` to gauge wallclock estimates. Accurate wallclock estimates allow a job to be scheduled as soon as possible in a slot that it will fit in. Low or high estimates can cause a job to be scheduled in a less favorable position.

14.4 Job Start Time Estimates

Each user can use the showstart command to display estimated start and completion times. The following example illustrates a typical response from issuing this command:

```
> showstart orion.13762
job orion.13762 requires 2 procs for 0:33:20
Estimated Rsv based start in                1:04:55 on Fri Jul 15 12:53:40
Estimated Rsv based completion in           2:44:55 on Fri Jul 15 14:33:40
Estimated Priority based start in           5:14:55 on Fri Jul 15 17:03:40
Estimated Priority based completion in      6:54:55 on Fri Jul 15 18:43:40
Estimated Historical based start in        00:00:00 on Fri Jul 15 11:48:45
Estimated Historical based completion in    1:40:00 on Fri Jul 15 13:28:45
Best Partition: fast
```

Estimation Types

Reservation-Based Estimates

Reservation-based start time estimation incorporates information regarding current administrative, user, and job reservations to determine the earliest time the specified job can allocate the needed resources and start running. In essence, this estimate indicates the earliest time the job will start, assuming this job is the highest priority job in the queue.

For reservation-based estimates, the information provided by this command is more highly accurate if the job is highest priority, if the job has a reservation, or if the majority of the jobs that are of higher priority have reservations. Consequently, site administrators wanting to make decisions based on this information may want to consider using the RESERVATIONDEPTH parameter to increase the number of priority-based reservations. This can be set so that most, or even all, idle jobs receive priority reservations and make the results of this command generally useful. The only caution of this approach is that increasing the **RESERVATIONDEPTH** parameter more tightly constrains the decisions of the scheduler and may result in slightly lower system utilization (typically less than 8% reduction).

Backlog/Priority Estimates

Priority-based job start analysis determines when the queried job will fit in the queue and determines the estimated amount of time required to complete the jobs currently running or scheduled to run before this job can start.

In all cases, if the job is running, this command returns the time the job starts. If the job already has a reservation, this command returns the start time of the reservation.

Historical Estimates

Historical analysis uses historical queue times for jobs that match a similar processor count and job duration profile. This information is updated on a sliding window that is configurable within `moab.cfg`.

Related topics

- ENABLESTARTESTIMATESTATS parameter
- showstart command

14.5 Providing Resource Availability Information

Moab provides commands to allow the user to query available resources. The showbf command displays what resources are available for immediate use. Using different command line parameters, such as `-m`, `-n`, and `-q` allows the user to query resources based on memory, nodecount, or QoS respectively.

14.6 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.

15.0 Cluster Analysis, Testing, and Simulation

- **Testing New Releases and Policies** on page 611
- **Testing New Middleware** on page 615
- **Simulations** on page 618

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?

15.1 Testing New Releases and Policies

- Moab Evaluation Modes
  - MONITOR Mode
  - TEST Mode
  - INTERACTIVE Mode
- Testing New Releases

- Testing New Policies
  - Verifying Correct Specification of New Policies
  - Verifying Correct Behavior of New Policies
  - Determining Long Term Impact of New Policies
- Moab Side-by-Side

Moab Evaluation Modes

MONITOR Mode

Moab supports a scheduling mode called *MONITOR*. In this mode, the scheduler initializes, contacts the resource manager and other peer services, and conducts scheduling cycles exactly as it would if running in *NORMAL* or production mode. Jobs are prioritized, reservations created, policies and limits enforced, and administrator and end-user commands enabled. The key difference is that although live resource management information is loaded, *MONITOR* mode disables Moab's ability to start, preempt, cancel, or otherwise modify jobs or resources. Moab continues to attempt to schedule exactly as it would in *NORMAL* mode, but its ability to actually impact the system is disabled. Using this mode, a site can quickly verify correct resource manager configuration and scheduler operation. This mode can also be used to validate new policies and constraints. In fact, Moab can be run in *MONITOR* mode on a production system while another scheduler or even another version of Moab is running on the same system. This unique ability can allow new versions and configurations to be fully tested without any exposure to potential failures and with no cluster downtime.

To run Moab in *MONITOR* mode, simply set the **MODE** attribute of the **SCHEDCFG** parameter to *MONITOR* and start Moab. Normal scheduler commands can be used to evaluate configuration and performance. Diagnostic commands can be used to look for any potential issues. Further, the Moab log file can be used to determine which jobs Moab attempted to start, and which resources Moab attempted to allocate.

If another instance of Moab is running in production and a site administrator wants to evaluate an alternate configuration or new version, this is easily done but care should be taken to avoid conflicts with the primary scheduler. Potential conflicts include statistics files, logs, checkpoint files, and user interface ports. One of the easiest ways to avoid these conflicts is to create a new test directory with its own log and statistics subdirectories. The new `moab.cfg` file can be created from scratch or based on the existing `moab.cfg` file already in use. In either case, make certain that the **PORT** attribute of the **SCHEDCFG** parameter differs from that used by the production scheduler by at least two ports. If testing with the production binary executable, the MOABHOMEDIR environment variable should be set to point to the new test directory to prevent Moab from loading the production `moab.cfg` file.

TEST Mode

TEST mode behaves much like *MONITOR* mode with the exception that Moab will log the scheduling actions it would have taken to the `stats/<DAY>.events` file. Using this file, sites can determine the actions Moab would have taken if running in *NORMAL* mode and verify all actions are in agreement with expected behavior.

INTERACTIVE Mode

INTERACTIVE mode allows for evaluation of new versions and configurations in a manner different from *MONITOR* mode. Instead of disabling all resource and job control functions, Moab sends the desired change request to the screen and requests permission to complete it. For example, before starting a job, Moab may print something like the following to the screen:

```
Command:  start job 1139.ncsa.edu on node list test013,test017,test018,test021
Accept:  (y/n) [default: n]?
```

The administrator must specifically accept each command request after verifying it correctly meets desired site policies. Moab will then execute the specified command. This mode is highly useful in validating scheduler behavior and can be used until configuration is appropriately tuned and all parties are comfortable with the scheduler's performance. In most cases, sites will want to set the scheduling mode to *NORMAL* after verifying correct behavior.

Testing New Releases

By default, Moab runs in a mode called *NORMAL*, which indicates that it is responsible for the cluster. It loads workload and resource information, and is responsible for managing that workload according to mission objectives and policies. It starts, cancels, preempts, and modifies jobs according to these policies.

If Moab is configured to use a mode called TEST, it loads all information, performs all analysis, but, instead of actually starting or modifying a job, it merely logs the fact that it would have done so. A test instance of Moab can run at the same time as a production instance of Moab. A test instance of Moab can also run while a production scheduler of another type (such as PBS, LSF, or SLURM) is simultaneously running. This multi-scheduler ability allows stability and performance tests to be conducted that can help answer the following questions:

- What impact do Moab services have on network, processor, and memory load?
- What impact do Moab services have on the underlying resource manager?
- Is Moab able to correctly import resource, workload, policy, and credential information from the underlying resource manager?
- Are Moab's logged scheduling decisions in line with mission objectives?

In test mode, all of Moab's commands and services operate normally allowing the use of client commands to perform analysis. In most cases, the mdiag command is of greatest value, displaying loaded values as well as reporting detected failures, inconsistencies, and object corruption. The following table highlights the most common diagnostics performed.

Command	Object
mdiag -n	Compute nodes, storage systems, network systems, and generic resources
mdiag -j	Applications and static jobs

Command	Object
mdiag -u **mdiag -g** **mdiag -a**	User, group, and account credentials
mdiag -c	Queues and policies
mdiag -R	Resource manager interface and performance
mdiag -S	Scheduler/system level failures introduced by corrupt information

These commands will not only verify proper scheduling objects but will also analyze the behavior of each resource manager, recording failures, and delivered performance. If any misconfiguration, corruption, interface failure, or internal failure is detected, it can be addressed in the test mode instance of Moab with no urgency or risk to production cluster activities.

Testing New Policies

Verifying Correct Specification of New Policies

The first aspect of verifying a new policy is verifying correct syntax and semantics. If using Moab Cluster Manager, this step is not necessary as this tool automatically verifies proper policy specification. If manually editing the `moab.cfg` file, the following command can be used for validation:

```
> mdiag -C
```

This command will validate the configuration file and report any misconfiguration.

Verifying Correct Behavior of New Policies

If concern exists over the impact of a new policy, an administrator can babysit Moab by putting it into INTERACTIVE mode. In this mode, Moab will schedule according to all mission objectives and policies, but before taking any action, it will request that the administrator confirm the action. See the interactive mode overview for more information.

In this mode, only actions approved by the administrator will be carried out. Once proper behavior is verified, the Moab mode can be set to *NORMAL*.

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.

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 testing so that there is no impact on the production workload.

```
SCHEDCFG[prod] MODE=NORMAL SERVER=orion.cxz.com:42020
RMCFG[TORQUE]  TYPE=PBS
IGNORENODES    node61,node62,node63,node64
IGNOREUSERS    gridtest1,gridtest2
...
SCHEDCFG[prod] MODE=NORMAL SERVER=orion.cxz.com:42030
RMCFG[TORQUE]  TYPE=PBS
IGNORENODES    !node61,node62,node63,node64
IGNOREUSERS    !gridtest1,gridtest2
...
```

Two completely independent Moab servers schedule the cluster. The first server handles all jobs and nodes except for the ones involved in the test. The second server handles only test nodes and test jobs. While both servers actively talk and interact with a single TORQUE resource manager, the IGNORE parameters cause them to not schedule, nor even see the other partition and its associated workload.*

When enabling Moab side-by-side, each Moab server should have an independent home directory to prevent logging and statistics conflicts. Also, in this environment, each Moab server should communicate with its client commands using a different port as shown in the previous example.

When specifying the IGNORENODES parameter, the exact node names, as returned by the resource manager, should be specified.

Related topics

- Testing New Versions and Configurations

15.2 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.

System	Details
Allocation Manager	Use *test* or *simulation* mode to drive scheduling queries, allocation debits, and reservations to accounting packages. Verify synchronization of cluster statistics and stress test interfaces and underlying databases.
On-Demand/Provisioning Services	Use *simulation* or native resource manager mode to drive triggers and resource management interfaces to enable dynamic provisioning of hardware, operating systems, application software, and services. Test reliability and scalability of data servers, networks, and provisioning software as well as the interfaces and business logic coordinating these changes.
Resource Monitoring	Use *test* or native resource manager mode to actively load information from compute, network, storage, and software license managers confirming validity of data, availability during failures, and scalability.

With each evaluation, the following tests can be enabled:

- functionality
- reliability
    - hard failure
        - hardware failure - compute, network, and data failures
        - software failure - loss of software services (NIS, LDAP, NFS, database)
        - soft failure
        - network delays, full file system, dropped network packets
    - corrupt data
- performance
- determine peak responsiveness in seconds/request
- determine peak throughput in requests/second
- determine responsiveness under heavy load conditions
- determine throughput under external load conditions
    - large user base (many users, groups, accounts)
    - large workload (many jobs)
    - large cluster (many nodes)

- manageability
  - full accounting for all actions/events
  - actions/failures can be easily and fully diagnosed

If using a native resource manager and you do not want to actually submit real workload, you can set the environment variable MFORCESUBMIT to allow virtual workload to be managed without ever launching a real process.

General Analysis

For all middleware interfaces, Moab provides built-in performance analysis and failure reporting. Diagnostics for these interfaces are available via the mdiag command.

Native Mode Analysis

Using native mode analysis, organizations can run Moab in *normal* mode with all facilities fully enabled, but with the resource manager fully emulated. With a native resource manager interface, any arbitrary cluster can be emulated with a simple script or flat text file. Artificial failures can be introduced, jobs can be virtually running, and artificial performance information generated and reported.

In the simplest case, emulation can be accomplished using the following configuration:

```
SCHEDCFG[natcluster] MODE=NORMAL SERVER=test1.bbli.com
ADMINCFG[1] USERS=dev
RMCFG[natcluster] TYPE=NATIVE CLUSTERQUERYURL=file://$HOME/cluster.dat
```

The preceding configuration will load cluster resource information from the file `cluster.dat`. An example resource information file follows:

```
node01 state=idle cproc=2
node02 state=idle cproc=2
node03 state=idle cproc=2
node04 state=idle cproc=2
node05 state=idle cproc=2
node06 state=idle cproc=2
node07 state=idle cproc=2
node08 state=idle cproc=2
```

In actual usage, any number of node attributes may be specified to customize these nodes, but in this example, only the node state and node configured processors attributes are specified.

The RMCFG flag *NORMSTART* indicates that Moab should not actually issue a job start command to an external entity to start the job, but rather start the job logically internally only.

If it is desirable to take an arbitrary action at the start of a job, end of a job, or anywhere in between, the JOBCFG parameter can be used to create one or more arbitrary triggers to initiate internal or external events. The triggers can do anything from executing a script, to updating a database, to using a Web service.

Using native resource manager mode, jobs may be introduced using the msub command according to any arbitrary schedule. Moab will load them, schedule them, and start them according to all site mission objectives and policies and drive all interfaced services as if running in a full production environment.

15.3 Simulations

- **Configuring Simulation** on page 618
- **Configuring Resources for Simulation** on page 620
- **Workload Event Format** on page 621
- **Interactive Simulation Tutorial** on page 631

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.

Image 15-1: Traditional TORQUE/Moab setup

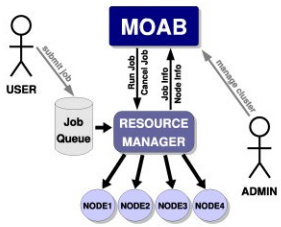
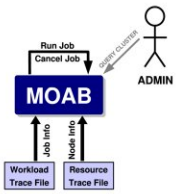


Image 15-2: Moab simulation setup (information retrieved from a workload trace file)



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?

15.3.1 Configuring Simulation

This section explains how to set up simulation mode in Moab.

Configuring Moab Simulation

1. Determine the 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.
2. Select 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.
3. Select 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.
4. Select 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.
5. Set up the 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:
   a. Change the SCHEDCFG attribute **MODE** from *NORMAL* or *MONITOR* to *SIMULATION*.

      Once *SIMULATION* is set, the following parameters control the environment, behavior, and policies used within the simulation:

      - Simulation Workload Specification, Queuing, and Management

Parameter	Description
SIMINITIALQUEUEDEPTH	Specifies simulation backlog.
SIMJOBSUBMISSIONPOLICY	Specifies how simulation backlog is managed.
SIMPURGEBLOCKEDJOBS	Removes jobs that can never run.
SIMWORKLOADTRACEFILE	Specifies source of job traces.

      - Time/Iteration Management

Parameter	Description
SIMAUTOSHUTDOWN	Shuts down when all jobs have been scheduled.
SIMSTARTTIME	Sets simulation clock to specified time.
STOPITERATION	Pauses simulation on specified iteration.

b. You may need to add these lines to the `moab.cfg` file:

```
SIMWORKLOADTRACEFILE    samples/workload.testcluster.txt
STOPITERATION           0
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 first two lines specify that the scheduler should run in simulation mode and use the referenced resource and workload trace files. In addition, leaving the STOPITERATION 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.

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 is 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.

6. Start 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.

15.3.2 Configuring Resources for Simulation

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 resources Moab uses to simulate are created using Moab RM language. To create a Moab simulation this way, perform the following steps:

1. Load resources into Moab using a native RM. Create a file manually using the format for different resource attributes (see **W.1 Moab Resource Manager Language Data Format** on page 1113) or run `mnodectl -q wiki ALL > nodes.dat` to create a resources simulation file from the resources in your environment and write it to a file (in this example, a file called `nodes.dat`). You can modify the resource file to add resources, change attributes, etc.
2. Set up the resource manager interface by inserting the following into your `moab.cfg` file.

   ```
   RMCFG[rmName] TYPE=NATIVE CLUSTERQUERYURL=FILE:///<locationOfFile>/<nameOfFile>
   ```

   For the example in step 1, you would replace *<nameOfFile>* with `nodes.dat`.
3. Restart Moab.

Sample Resource Trace

```
n8 STATE=Idle PARTITION=base AMEMORY=16000 APROC=4 CMEMORY=16000 CPROC=4 RM=base
NODEACCESSPOLICY=SHARED FEATURE=linux ARCH=x86_64
```

Related topics

- mnodectl -q wiki - outputs Moab RM language format directly to a file

15.3.3 Workload Event Format

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 white space 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.

- Workload Event Record Format
- Creating New Workload Accounting Records/Traces
- Reservation Records/Traces
- Recording Job events

Workload Event Record Format

All job events (*JOBSUBMIT*, *JOBSTART*, *JOBEND*, and so forth) provide job data in a standard format as described in the following table:

Field Name	Field Index	Data Format	Default Value	Details
Event Time (Human Readable)	1	HH:MM:SS	-	Specifies time event occurred.
Event Time (Epoch)	2	*<epochtime>*	-	Specifies time event occurred.
Object Type	3	job	-	Specifies record object type.
Object ID	4	*<STRING>*	-	Unique object identifier.
Object Event	5	one of *jobcancel*, *jobcheckpoint*, *jobend*, *jobfailure*, *jobhold*, *jobmigrate*, *jobpreempt*, *jobreject*, *jobresume*, *jobstart* or *jobsubmit*	-	Specifies record event type.
Nodes Requested	6	*<INTEGER>*	0	Number of nodes requested (0 = no node request count specified).
Tasks Requested	7	*<INTEGER>*	1	Number of tasks requested.
User Name	8	*<STRING>*	-	Name of user submitting job.
Group Name	9	*<STRING>*	-	Primary group of user submitting job.
Wallclock Limit	10	*<INTEGER>*	1	Maximum allowed job duration (in seconds).
Job Event State	11	*<STRING>*	-	Job state at time of event.
Required Class	12	*<STRING>*	[DEFAULT:1]	Class/queue required by job specified as square bracket list of <QUEUE>[:<QUEUEINSTANCE>] requirements. (For example: [batch:1]).

Field Name	Field Index	Data Format	Default Value	Details
Submission Time	13	*<INTEGER>*	0	Epoch time when job was submitted.
Dispatch Time	14	*<INTEGER>*	0	Epoch time when scheduler requested job begin executing.
Start Time	15	*<INTEGER>*	0	Epoch time when job began executing. This is usually identical to Dispatch Time.
Completion Time	16	*<INTEGER>*	0	Epoch time when job completed execution.
Required Node Architecture	17	*<STRING>*	-	Required node architecture if specified.
Required Node Operating System	18	*<STRING>*	-	Required node operating system if specified.
Required Node Memory Comparison	19	one of >, >=, =, <=, <	>=	Comparison for determining compliance with required node memory.
Required Node Memory	20	*<INTEGER>*	0	Amount of required configured RAM (in MB) on each node.
Required Node Disk Comparison	21	one of >, >=, =, <=, <	>=	Comparison for determining compliance with required node disk.
Required Node Disk	22	*<INTEGER>*	0	Amount of required configured local disk (in MB) on each node.
Required Node Attributes/Features	23	*<STRING>*	-	Square bracket enclosed list of node features required by job if specified. (For example: [fast][ethernet])
System Queue Time	24	*<INTEGER>*	0	Epoch time when job met all fairness policies.

Field Name	Field Index	Data Format	Default Value	Details
Tasks Allocated	25	*<INTEGER>*	*<TASKS REQUESTED>*	Number of tasks actually allocated to job. In most cases, this field is identical to field #7, Tasks Requested.
Required Tasks Per Node	26	*<INTEGER>*	-1	Number of Tasks Per Node required by job or '-1' if no requirement specified.
QOS	27	*<STRING>* `[:<STRING>]`	-	QoS requested/assigned using the format *<QOS_REQUESTED>* `[:<QOS_DELIVERED>]`. (For example: hipriority:bottomfeeder)
JobFlags	28	*<STRING>* `[:` *<STRING>*`]...`	-	Square bracket delimited list of job attributes. (For example: `[BACKFILL] [PREEMPTEE]`)
Account Name	29	*<STRING>*	-	Name of account associated with job if specified.
Executable	30	*<STRING>*	-	Name of job executable if specified.
Resource Manager Extension String	31	*<STRING>*	-	Resource manager specific list of job attributes if specified. See the Resource Manager Extension Overview for more information.
Bypass Count	32	*<INTEGER>*	-1	Number of times job was bypassed by lower priority jobs via backfill or '-1' if not specified.
ProcSeconds Utilized	33	*<DOUBLE>*	0	Number of processor seconds actually used by job.
Partition Name	34	*<STRING>*	[DEFAULT]	Name of partition in which job ran.
Dedicated Processors per Task	35	*<INTEGER>*	1	Number of processors required per task.

Field Name	Field Index	Data Format	Default Value	Details
Dedicated Memory per Task	36	*<INTEGER>*	0	Amount of RAM (in MB) required per task.
Dedicated Disk per Task	37	*<INTEGER>*	0	Amount of local disk (in MB) required per task.
Dedicated Swap per Task	38	*<INTEGER>*	0	Amount of virtual memory (in MB) required per task.
Start Date	39	*<INTEGER>*	0	Epoch time indicating earliest time job can start.
End Date	40	*<INTEGER>*	0	Epoch time indicating latest time by which job must complete.
Allocated Host List	41	*<hostname>* [, *<hostname>*]...	-	Comma delimited list of hosts allocated to job. (For example: node001,node004)
Resource Manager Name	42	*<STRING>*	-	Name of resource manager if specified.
Required Host List	43	*<hostname>* [, *<hostname>*]...	-	List of hosts required by job. (If the job's task count is greater than the specified number of hosts, the scheduler must use these nodes in addition to others; if the job's task count is less than the specified number of hosts, the scheduler must select needed hosts from this list.)
Reservation	44	*<STRING>*	-	Name of reservation required by job if specified.
Application Simulator Data	45	*<STRING>* [:*<STRING>*]	-	Name of application simulator module and associated configuration data. (For example: HSM:IN-N=infile.txt:140000;OUT=outfile.txt:500000)

Field Name	Field Index	Data Format	Default Value	Details
Set Description	46	*<STRING>*:*<STRING>*[:*<STRING>*]	-	Set constraints required by node in the form *<SetConstraint>*:*<SetType>*[:*<SetList>*] 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)
Job Message	47	*<STRING>*	-	Job messages including resource manager, scheduler, and administrator messages if specified.
Job Cost	48	*<DOUBLE>*	0.0	Cost of executing job incorporating resource consumption metric, resource quantity consumed, and credential, allocated resource, and delivered QoS charge rates.
History	49	*<STRING>*	-	List of job events impacting resource allocation (XML). History information is only reported in Moab 5.1.0 and higher.

Field Name	Field Index	Data Format	Default Value	Details
Utilization	50	Comma-delimited list of one or more of the following: *<ATTR>*=*<VALUE>* pairs where *<VALUE>* is a double and *<ATTR>* is one of the following: `network` (in MB transferred), `license` (in license-seconds), `storage` (in MB-seconds stored), or `gmetric:`*<TYPE>*.	-	Cumulative resources used over life of job.
Estimate Data	51	*<STRING>*	-	List of job estimate usage.
Completion Code	52	*<INTEGER>*	-	Job exit status/completion code.
Extended Memory Load Information	53	*<STRING>*	-	**Deprecated.** Extended memory usage statistics (max, mem, avg, and so forth).
Extended CPU Load Information	54	*<STRING>*	-	Extended CPU usage statistics (max, mem, avg, and so forth).
Generic Metric Averages	55	*<STRING>*	-1	Generic metric averages.
Effective Queue Duration	56	*<INTEGER>*	-1	The amount of time, in seconds, that the job was eligible for scheduling.
Job Submission Arguments	57	*<STRING>*	-	The job's submit arguments and script. This field is enabled by setting STOREJOBSUBMISSION to *TRUE*.

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 - - - 0 - -
```

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:

1.
```
# collect all job records for July
> cat /opt/moab/stats/events.*July*2012 | grep JOBEND > /opt/moab/DecJobs.txt
```

2.
```
edit moab.cfg for use job records
> vi /opt/moab/etc/moab.cfg
   (add 'SIMWORKLOADTRACEFILE /opt/moab/DecJobs.txt')
```

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.

JOBSUBMISSIONPOLICY Value	Critical Time Based Fields
NORMAL	WallClock Limit Submission Time StartTime Completion Time
CONSTANTJOBDEPTH **CONSTANTPSDEPTH**	WallClock Limit StartTime Completion Time

Dispatch Time should always be identical to Start Time.

In all cases, the difference of 'Completion Time - Start Time' is used to determine actual job run time.

System Queue Time and Proc-Seconds Utilized are only used for statistics gathering purposes and will not alter the behavior of the simulation.

In all cases, relative time values are important, i.e., Start Time must be greater than or equal to Submission Time and less than Completion Time.

Creating Workload Traces From Scratch

There is nothing which prevents a completely new workload trace from being created from scratch. To do this, simply create a file with fields matching the format described in the Workload Event Record Format section.

Reservation Records/Traces

All reservation events provide reservation data in a standard format as described in the following table:

Field Name	Field Index	Data Format	Default Value	Details
Event Time (Human)	0	`[HH:MM:SS]`	-	Specifies time event occurred.

Field Name	Field Index	Data Format	Default Value	Details
Event Time (Epoch)	1	*<epochtime>*	-	Specifies time event occurred.
Object Type	2	rsv	-	Specifies record object type.
Object ID	3	*<STRING>*	-	Unique object identifier.
Object Event	4	one of *rsvcreate*, *rsvstart*, *rsvmodify*, *rsvfail* or *rsvend*	-	Specifies record event type.
Creation Time	5	*<EPOCHTIME>*	-	Specifies epoch time of reservation start date.
Start Time	6	*<EPOCHTIME>*	-	Specifies epoch time of reservation start date.
End Time	7	*<EPOCHTIME>*	-	Specifies epoch time of reservation end date.
Tasks Allocated	8	*<INTEGER>*	-	Specifies number of tasks allocated to reservation at event time.
Nodes Allocated	9	*<INTEGER>*	-	Specifies number of nodes allocated to reservation at event time.
Total Active Proc-Seconds	10	*<INTEGER>*	-	Specifies proc-seconds reserved resources were dedicated to one or more job at event time.
Total Proc-Seconds	11	*<INTEGER>*	-	Specifies proc-seconds resources were reserved at event time.
Hostlist	12	*<comma-delimited list of hostnames>*	-	Specifies list of hosts reserved at event time.
Owner	13	*<STRING>*	-	Specifies reservation ownership credentials.
ACL	14	*<STRING>*	-	Specifies reservation access control list.

Field Name	Field Index	Data Format	Default Value	Details
Comment	15	*<STRING>*	-	Specifies general human readable event message.
Command Line	16	*<STRING>*	-	Displays the command line arguments used to create the reservation (only shows on the rsvcreate event).

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 example:

```
RECORDEVENTLIST JOBCANCEL,JOBCOMPLETE,JOBSTART,JOBSUBMIT
...
```

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.

Related topics

- Event Logging Overview
- SIMWORKLOADTRACEFILE

15.3.4 Interactive Simulation Tutorial

This of this section provides an interactive tutorial to demonstrate the basics of the simulator's capacities in Moab. It is an example of what you can do once you have set up simulation. 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

To run through the simulation mode tutorial

1. Run moab.

```
> moab&
```

2. Check the status of the queue (see Checking the Queue Status).
3. Check the status of the job. If any jobs are not running, find the problem (see Determining Why Jobs Are Not Running).
4. Advance and check the status of Moab iterations and time in simulation mode (see Controlling Iterations).
5. View and manage reservations and their nodes and jobs (see Managing Reservations Applying to the Queue).
6. Verify that the Moab simulation is scheduling fairly (see Verifying Fair Scheduling).
7. Take down the entire system for maintenance (see Taking the System Down for Maintenance).

15.3.4.1 Checking the Queue Status

Verify that Moab is running by executing showq:

```
> showq
active jobs------------------------
JOBNAME            USERNAME      STATE  PROC   REMAINING            STARTTIME

fr8n01.187.0            570    Running    20  1:00:00:00  Mon Feb 16 11:54:03
fr8n01.189.0            570    Running    20  1:00:00:00  Mon Feb 16 11:54:03
fr8n01.190.0            570    Running    20  1:00:00:00  Mon Feb 16 11:54:03
fr8n01.191.0            570    Running    20  1:00:00:00  Mon Feb 16 11:54:03
fr8n01.276.0            550    Running    20  1:00:00:00  Mon Feb 16 11:54:03
fr1n04.369.0            550    Running    20  1:00:00:00  Mon Feb 16 11:54:03
fr1n04.487.0            550    Running    20  1:00:00:00  Mon Feb 16 11:54:03

     7 active jobs     140 of  196 Processors Active (71.43%)

eligible jobs----------------------
JOBNAME            USERNAME      STATE  PROC     WCLIMIT            QUEUETIME

fr1n04.362.0            550       Idle    20  1:00:00:00  Mon Feb 16 11:53:33
fr1n04.363.0            550       Idle    20  1:00:00:00  Mon Feb 16 11:53:33
fr1n04.365.0            550       Idle    20  1:00:00:00  Mon Feb 16 11:53:33
fr1n04.366.0            550       Idle    20  1:00:00:00  Mon Feb 16 11:53:33
fr1n04.501.0            570       Idle    20  1:00:00:00  Mon Feb 16 11:53:33
fr1n04.580.0            570       Idle    20  1:00:00:00  Mon Feb 16 11:53:33
fr1n04.597.0            570       Idle    20  1:00:00:00  Mon Feb 16 11:53:33
fr1n04.598.0            570       Idle    20  1:00:00:00  Mon Feb 16 11:53:33
fr1n04.602.0            570       Idle    20  1:00:00:00  Mon Feb 16 11:53:33

9 eligible jobs
blocked jobs-----------------------
JOBNAME            USERNAME      STATE  PROC     WCLIMIT            QUEUETIME

0 blocked jobs
Total jobs:  16
```

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 SIMINITIALQUEUEDEPTH 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).

15.3.4.2 Determining Why Jobs Are Not Running

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.

```
> 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:

```
> 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 `>= 256 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.

15.3.4.3 Controlling Iterations

Advancing the simulator an iteration, the following happens:

```
> 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.

```
> 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. By default, RMPOLLINTERVAL is set to 30 seconds.

To see what RMPOLLINTERVAL is set to, use the showconfig command:

```
> showconfig | grep RMPOLLINTERVAL
RMPOLLINTERVAL                  30,30
```

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              S PAR  EFFIC  XFACTOR  Q      USER    GROUP    MHOST PROCS
REMAINING            STARTTIME
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 STOPITERATION 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                PRIORITY  XFACTOR  Q      USER    GROUP  PROCS     WCLIMIT
CLASS      SYSTEMQUEUETIME
fr1n04.362.0*               1      1.0  -       550      519     20  1:00:00:00
batch  Mon Feb 16 11:53:33
fr1n04.363.0                1      1.0  -       550      519     20  1:00:00:00
batch  Mon Feb 16 11:53:33
fr1n04.365.0                1      1.0  -       550      519     20  1:00:00:00
batch  Mon Feb 16 11:53:33
```

```
fr1n04.366.0                1      1.0  -       550      519     20  1:00:00:00
batch  Mon Feb 16 11:53:33
fr1n04.501.0                1      1.0  -       570      519     20  1:00:00:00
batch  Mon Feb 16 11:53:33
fr1n04.580.0                1      1.0  -       570      519     20  1:00:00:00
batch  Mon Feb 16 11:53:33
fr1n04.597.0                1      1.0  -       570      519     20  1:00:00:00
batch  Mon Feb 16 11:53:33
fr1n04.598.0                1      1.0  -       570      519     20  1:00:00:00
batch  Mon Feb 16 11:53:33
fr1n04.602.0                1      1.0  -       570      519     20  1:00:00:00
batch  Mon Feb 16 11:53:33
fr1n04.743.0                1      1.0  -       570      519     20  1:00:00:00
batch  Mon Feb 16 11:54:03
fr1n04.744.0                1      1.0  -       570      519     20  1:00:00:00
batch  Mon Feb 16 11:54:03
fr1n04.746.0                1      1.0  -       570      519     20  1:00:00:00
batch  Mon Feb 16 11:54:03
fr1n04.747.0                1      1.0  -       570      519     20  1:00:00:00
batch  Mon Feb 16 11:54:03
fr8n01.388.0                1      1.0  -       550      519     20  1:00:00:00
batch  Mon Feb 16 11:54:03
14 eligible jobs
Total jobs:  14
```

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.

15.3.4.4 Managing Reservations Applying to the Queue

The showres command shows all reservations currently on the system.

```
> showres
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
```

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>*.

```
> showres -n fr1n04.362.0
reservations on Mon Feb 16 11:54:03
NodeName                    Type      ReservationID   JobState Task       Start
Duration  StartTime
fr5n09                       Job       fr1n04.362.0       Idle    1  1:00:00:00
1:00:00:00  Tue Feb 17 11:54:03
...
fr7n15                       Job       fr1n04.362.0       Idle    1  1:00:00:00
1:00:00:00  Tue Feb 17 11:54:03
20 nodes reserved
```

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
fr8n01.804.0            529    Running     5    00:04:30  Mon Feb 16 11:54:03
fr8n01.187.0            570    Running    20    23:59:30  Mon Feb 16 11:54:03
...
     9 active jobs     177 of  196 Processors Active (90.31%)
eligible jobs----------------------
JOBNAME            USERNAME      STATE  PROC     WCLIMIT            QUEUETIME
...
fr8n01.963.0            586       Idle    32     9:00:00  Mon Feb 16 11:54:33
fr8n01.1016.0           570       Idle    20  1:00:00:00  Mon Feb 16 11:54:33
16 eligible jobs
...
```

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.

```
> showq
active jobs------------------------
JOBNAME            USERNAME      STATE  PROC   REMAINING            STARTTIME
fr8n01.804.0            529    Running     5    00:02:30  Mon Feb 16 11:54:03
...
fr1n04.501.0            570    Running    20  1:00:00:00  Mon Feb 16 11:56:33
```

```
fr8n01.388.0            550    Running    20  1:00:00:00  Mon Feb 16 11:56:33
     9 active jobs     177 of  196 Processors Active (90.31%)
...
    14 eligible jobs
...
```

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
fr1n04.1953.0           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
...
```

```
Job Eligibility Analysis -------
job cannot run in partition DEFAULT (idle procs do not meet requirements : 20 of 32
procs found)
idle procs:  40  feasible procs:  20
Rejection Reasons: [Memory : 20][State : 156]

job fr8n01.1075.0
...
Network: hps_user  Memory >= 256M  Disk >= 0  Swap >= 0
...
job cannot run in partition DEFAULT (idle procs do not meet requirements : 0 of 32
procs found)
idle procs:  40  feasible procs:  0
Rejection Reasons: [Memory : 20][State : 156][ReserveTime : 20]

job fr8n01.1076.0
...
Network: hps_user  Memory >= 256M  Disk >= 0  Swap >= 0
...
job cannot run in partition DEFAULT (idle procs do not meet requirements : 0 of 16
procs found)
idle procs:  40  feasible procs:  0
Rejection Reasons: [Memory : 20][State : 156][ReserveTime : 20]
```

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, blocked jobs, fairshare, groups, jobs, nodes, QoS, 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
Name      State  Procs Memory         Disk          Swap      Speed  Opsys   Arch Par
 Load Rsv  ...
fr10n09   Idle   1:1   256:256      9780:9780   411488:411488  1.00  AIX43  R6000 DEF
 0.00 001  .
fr10n11   Idle   1:1   256:256      8772:8772   425280:425280  1.00  AIX43  R6000 DEF
 0.00 001  .
fr10n13   Idle   1:1   256:256      9272:9272   441124:441124  1.00  AIX43  R6000 DEF
 0.00 001  .
fr10n15   Idle   1:1   256:256      8652:8652   440776:440776  1.00  AIX43  R6000 DEF
 0.00 001
fr11n01   Idle   1:1   256:256      7668:7668   438624:438624  1.00  AIX43  R6000 DEF
 0.00 001
fr11n03   Idle   1:1   256:256      9548:9548   424584:424584  1.00  AIX43  R6000 DEF
 0.00 001
fr11n05   Idle   1:1   256:256     11608:11608  454476:454476  1.00  AIX43  R6000 DEF
 0.00 001
fr11n07   Idle   1:1   256:256      9008:9008   425292:425292  1.00  AIX43  R6000 DEF
 0.00 001
fr11n09   Idle   1:1   256:256      8588:8588   424684:424684  1.00  AIX43  R6000 DEF
 0.00 001
fr11n11   Idle   1:1   256:256      9632:9632   424936:424936  1.00  AIX43  R6000 DEF
 0.00 001
fr11n13   Idle   1:1   256:256      9524:9524   425432:425432  1.00  AIX43  R6000 DEF
```

```
 0.00 001
fr11n15   Idle   1:1   256:256      9388:9388   425728:425728  1.00  AIX43  R6000 DEF
 0.00 001
fr14n01   Idle   1:1   256:256      6848:6848   424260:424260  1.00  AIX43  R6000 DEF
 0.00 001
fr14n03   Idle   1:1   256:256      9752:9752   424192:424192  1.00  AIX43  R6000 DEF
 0.00 001
fr14n05   Idle   1:1   256:256      9920:9920   434088:434088  1.00  AIX43  R6000 DEF
 0.00 001
fr14n07   Idle   1:1   256:256      2196:2196   434224:434224  1.00  AIX43  R6000 DEF
 0.00 001
fr14n09   Idle   1:1   256:256      9368:9368   434568:434568  1.00  AIX43  R6000 DEF
 0.00 001
fr14n11   Idle   1:1   256:256      9880:9880   434172:434172  1.00  AIX43  R6000 DEF
 0.00 001
fr14n13   Idle   1:1   256:256      9760:9760   433952:433952  1.00  AIX43  R6000 DEF
 0.00 001
fr14n15   Idle   1:1   256:256     25000:25000  434044:434044  1.00  AIX43  R6000 DEF
 0.00 001
fr17n05   Idle   1:1   128:128     10016:10016  182720:182720  1.00  AIX43  R6000 DEF
 0.00 000
...
Total Nodes: 196  (Active: 156  Idle: 40  Down: 0)
```

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.

15.3.4.5 Verifying Fair Scheduling

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
statistics initialized Wed Dec 31 17:00:00
         |------ Active ------|--------------------------------- Completed -----------
------------------------|
user      Jobs Procs ProcHours Jobs    %    PHReq    %    PHDed    %   FSTgt  AvgXF
MaxXF  AvgQH  Effic  WCAcc
520          1    46    172.88    1   0.00  356.5   0.00  541.3   0.00 -----   1.04
0.00   0.35 100.00 100.00
550          1    20    301.83    7   0.00 3360.0   0.00  283.7   0.00 -----   0.03
0.00   0.06 100.00   3.17
524          1    32    239.73 ---- ------ ------ ------  272.3   0.00 ----- ------ --
---- ------ 100.00 ------
570          1    20    301.00   14   0.00 6720.0   0.00  199.5   0.00 -----   0.01
0.00   0.20 100.00   0.34
588          0     0      0.00    1   0.00  768.0   0.00  159.7   0.00 -----   0.21
0.00   0.00 100.00  20.80
578          6     6    146.82 ---- ------ ------ ------   53.2   0.00 ----- ------ --
---- ------ 100.00 ------
586          1    32    265.07 ---- ------ ------ ------   22.9   0.00 ----- ------ --
---- ------ 100.00 ------
517          0     0      0.00    1   0.00  432.0   0.00    4.8   0.00 -----   0.02
0.00   0.12 100.00   1.10
529          0     0      0.00    1   0.00    0.4   0.00    1.3   0.00 -----   1.00
0.00   0.00 100.00 100.00
```

```
> showstats -g
statistics initialized Wed Dec 31 17:00:00
         |------ Active ------|--------------------------------- Completed -----------
------------------------|
group     Jobs Procs ProcHours Jobs    %    PHReq    %    PHDed    %   FSTgt  AvgXF
MaxXF  AvgQH  Effic  WCAcc
503          1    32    239.73    1   0.00  432.0   0.00  277.1   0.00 -----   0.02
0.00   0.12 100.00   1.10
501          1    32    265.07 ---- ------ ------ ------   22.9   0.00 ----- ------ --
---- ------ 100.00 ------
519          9    92    922.54   24   0.00 11204.9   0.00 1238.6   0.00 -----   0.11
0.00   0.15 100.00  10.33
```

15.3.4.6 Taking the System Down for Maintenance

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
```

16.0 Green computing

16.1 About green computing

To conserve energy, Moab can automatically turn power off idle nodes that have no reservations or running workload on them. Conversely, Moab can automatically power on additional nodes when jobs require such. For Moab to automatically perform these power management functions, you must configure Moab for green computing operation.

Using the MAXGREENSTANDBYPOOLSIZE parameter, you can specify a "green pool" size, which is the number of idle nodes Moab keeps turned powered on and ready to run jobs (even if some nodes are idle). Moab turns off idle nodes that exceed the number specified with the MAXGREENSTANDBYPOOLSIZE parameter. Thus, Moab automatically powers nodes on and off using a power provisioning resource manager to keep the green pool of idle nodes at the configured size.

Moab can work with various power management solutions such as IPMI, iLO (HP), DRAC (Dell), xCAT (IBM), and others. Adaptive Computing has provided some IPMI-based reference scripts you can use to deploy a green computing solution. The examples in this section will generally refer to our reference scripts and to IPMI power management. You can modify our supplied scripts to use your own power management system's commands or you can create your own scripts.

If you intentionally power off a node, a green policy might try to turn it back on automatically. If you want the node to remain powered off, you must associate a reservation with the node before you power it off. After you finish with the node, you can return it to service by deleting the reservation.

Tasks associated with green computing:

The following sections include information about the configurations and settings needed to use green computing.

- **Enabling green computing** on page 644
- **Deploying Adaptive Computing IPMI scripts** on page 647
- **Choosing which nodes Moab powers on or off** on page 648
- **Adjusting green pool size** on page 649
- **Handling power-related events** on page 649
- **Maximizing scheduling efficiency** on page 650
- **Troubleshooting green computing** on page 651

16.2 How-to's

16.2.1 Enabling green computing

Context

To enable green computing, follow the steps below. These steps are generic for all green computing configurations. It doesn't matter what power management solution you employ, these steps are what enables green computing in Moab.

To enable green computing

1. Edit `moab.cfg` to enable green computing. There are four things you must configure for basic functionality of green computing:

   a. Configure the POWERPOLICY attribute of the NODECFG parameter. The default value is *STATIC*. Set it to *OnDemand*.

   b. Configure a power provisioning resource manager to be TYPE=*NATIVE* and RESOURCETYPE=*PROV*. The resource type of *PROV* means the RM works only with node hardware and not workloads.

   c. Configure a CLUSTERQUERYURL attribute of the power provisioning RM to point to the power query script you'd like to use. Moab uses this script to query the current power state of the nodes. **CLUSTERQUERYURL** is traditionally used as a workload query but is also used by green computing for the node power state query. Adaptive Computing provides a reference IPMI script you can use.

   d. Configure a NODEPOWERURL attribute of the power provisioning RM to point to the power action script you'd like to use. Moab uses this script to turn nodes on or off. Adaptive Computing provides a reference IPMI script you can use.

```
NODECFG[DEFAULT] POWERPOLICY=OnDemand
RMCFG[ipmi] TYPE=NATIVE RESOURCETYPE=PROV
RMCFG[ipmi] CLUSTERQUERYURL=exec://$TOOLSDIR/ipmi/ipmi.mon.py
RMCFG[ipmi] NODEPOWERURL=exec://$TOOLSDIR/ipmi/ipmi.power.py
```

Sample moab.cfg for green computing

Below is a sample `moab.cfg` configuration file of a green computing setup using the Adaptive Computing IPMI scripts.

```
#######################################################################################
#
#  Use 'mdiag -C' to validate config file parameters
#
#######################################################################################

SCHEDCFG[Moab]         SERVER=myhostname:5150
ADMINCFG[1]            USERS=myusername,root
TOOLSDIR               /$HOME/tools
LOGLEVEL               1
```

```
#############################################################################
#
#  Basic Resource Manager configuration
#
#  For more information on configuring a Resource Manager, see:
#  docs.adaptivecomputing.com
#
#############################################################################

RMCFG[local]   TYPE=NATIVE
RMCFG[local]   CLUSTERQUERYURL=exec://$HOME/scripts/query.resource
RMCFG[local]   WORKLOADQUERYURL=exec://$HOME/scripts/query.workload

RMCFG[local]   JOBSUBMITURL=exec://$HOME/scripts/submit.pl
RMCFG[local]   JOBSTARTURL=exec://$HOME/scripts/job.start
RMCFG[local]   JOBCANCELURL=exec://$HOME/scripts/job.cancel
RMCFG[local]   JOBMODIFYURL=exec://$HOME/scripts/job.modify
RMCFG[local]   JOBREQUEUEURL=exec://$HOME/scripts/job.requeue
RMCFG[local]   JOBSUSPENDURL=exec://$HOME/scripts/job.suspend
RMCFG[local]   JOBRESUMEURL=exec://$HOME/scripts/job.resume

################################
# GREEN configuration:
################################
# Turn on "green" policy. (This is the policy that enables green computing).
# Here we are doing it for all nodes, but it can be controlled on a node-by-node basis
# Default is STATIC, which means green computing is disabled.
#NODECFG[DEFAULT]  POWERPOLICY=STATIC
NODECFG[DEFAULT]  POWERPOLICY=OnDemand

# Configure the power provisioning and power state query scripts for the power
# management system.
# Note that this is an entirely different RM (with a name of power in this case
# and a type of 'PROV').
# The PROV type RM is the only one that uses a NODEPOWERURL.  Additionally, the
# output of the CLUSTERQUERYURL for this type of RM is different.  (See docs)
RMCFG[power]  TYPE=NATIVE RESOURCETYPE=PROV
RMCFG[power]  NODEPOWERURL=exec://$TOOLSDIR/ipmi/ipmi.power.py
RMCFG[power]  CLUSTERQUERYURL=exec://$TOOLSDIR/ipmi/ipmi.mon.py

# We want green policy to work so it allocates jobs to compute nodes already
# powered on and will power on powered-off compute nodes only when there are
# no powered-on compute nodes available. This requires using the PRIORITY
# node allocation policy with a PRIORITYF function that has the POWER variable
# as the greatest contributing factor to the function (1 = powered-on,
# 0 = powered-off).
# If we want all compute nodes to operate under green policy, we can assign
# the PRIORITYF function to the default node configuration, which is easier
# than assigning it to individual compute nodes. If only some compute nodes
# should operate under green policy, then the PRIORITYF function must be
# configured for the individual nodes. Note the POWER variable must be the
# largest factor in the function below; it is assigned the largest multiplier,
# which should be greater than the sum of all other factors! Doing so forces
# Moab to use all eligible powered-on nodes for workload placement before
# powering on any eligible powered-off nodes.

# Enable PRIORITYF functionality
NODEALLOCATIONPOLICY  PRIORITY

# Use a priority function that uses power as the major factor (plus some other
imaginary factors)
#NODECFG[DEFAULT]      PRIORITYF='1000000*POWER + 1000*factor2 + 100*factor3...'
```

```
# Use a priority function where power is the only factor.
#NODECFG[DEFAULT]      PRIORITYF='10000*POWER'
# Use a priority function that adds some randomness but uses power as the major
factor.
NODECFG[DEFAULT]      PRIORITYF='10000*POWER + 10*RANDOM'

# Set a priority function that specifies the order nodes should be chosen to power
# up/down. By default, Moab will start at the top of the node list and go down. Some
# installations want to rotate power cycles among nodes in a different order.
# The configuration below forces Moab to power on/off random nodes, which
# eventually guarantees all nodes occasionally go through a power cycle.
GREENPOOLPRIORITYF '10*RANDOM'

# Ensure we are recording power management events
# (powering on and off nodes are recorded as "node modification" events).
RECORDEVENTLIST +NODEMODIFY

# Set the size of the standby pool. This is the number of idle nodes that will
# be powered on and idle. As the workload changes, Moab turns nodes on
# or off to try to meet this goal.
# Default value is 0
MAXGREENSTANDBYPOOLSIZE 5

# Set the length of time that it takes to power a node on/off. This will be the
# walltime of the system job that performs the power operation and should be the
# maximum expected time. If Moab detects (via the power RM) that the power
# operations have all completed, the system job will finish early.
# Default value is 10 minutes (600)
#PARCFG[ALL]      NODEPOWEROFFDURATION=600
#PARCFG[ALL]      NODEPOWERONDURATION=600

# Set the length of time a node should remain idle before it is powered off.
# This prevents Moab from immediately powering off nodes that have just finished
# a job. Increasing this number should decrease power on/off thrashing
# This should be set higher than NODEPOWEROFFDURATION and/or NODEPOWERONDURATION
#NODEIDLEPOWERTHRESHOLD 660

# If a node fails to power on, we need to remove it from the available nodes so
# Moab won't keep [re-]trying to power it on. Do this by setting a reservation
# on the failed node to give time for manual investigation.
#RMCFG[torque] NODEFAILURERSVPROFILE=failure
#RSVPROFILE[failure] DURATION=3600
```

Related topics

- **Deploying Adaptive Computing IPMI scripts** on page 647
- **Choosing which nodes Moab powers on or off** on page 648
- **Adjusting green pool size** on page 649
- **Handling power-related events** on page 649
- **Maximizing scheduling efficiency** on page 650
- **Troubleshooting green computing** on page 651

16.2.2 Deploying Adaptive Computing IPMI scripts

Context

If you want to enable green computing on your system using the Adaptive Computing supplied IPMI reference scripts, follow the steps here. The IPMI scripts provided are meant as a reference for you to configure the solution to your environment, but can also be used as-is.

Prerequisites

- OpenIPMI and ipmitool must be installed and working.
- All nodes must have the same IPMI username and password.
- You must know the IPMI host names and/or IPMI IP addresses of your nodes.
- Python must be installed. The provided IPMI scripts were developed using Python 2.6.5.
- You must identify your Moab home directory. These instructions assume the default Moab home directory of `/opt/moab`.
- You must identify your Moab tools directory. These instructions assume the default Moab tools directory of `/opt/moab/tools`.

To deploy the Adaptive Computing IPMI scripts

1. Edit the `/opt/moab/tools/ipmi/config.py` script:
   a. Set **self.ipmiuser** to the IPMI username for your nodes.
   b. Set **self.ipmipass** to the location of the IPMI password file (`/opt/moab/passfile.txt` by default).

      The permissions for the directory and the password file itself should be set so that they can be read only by root or the Moab user running the script.

   c. Set **self.homeDir** to your Moab home directory.
   d. If desired, change the **self.pollInterval** value. This is the interval, in seconds, between polls from the IPMI monitoring script.
   e. The **self.ipmifile** value is the name of a temporary file where the cluster query information is stored. You can change this or leave it alone.
   f. The **self.bmcaddrmap** value is the filename for the Moab node name/IPMI mapping. The file must exist in the Moab home directory and will be created in the next step.
2. Create a `node-bmc.txt` file in the Moab home directory. The file must contain a space-delimited list of Moab node names that map to the IPMI host names or IP address. For Example:

```
node01 node01_ipmi   # For all three of these entries, the first value is the
node02 node02_ipmi   # node name as Moab knows it. The second value is either
node03 10.1.1.1      # the node IPMI name or IPMI IP address.
```

3. Configure the `moab.cfg` file for green computing as described in Enabling green computing. Use the `ipmi.mon.py` script for the `CLUSTERQUERYURL` and the `ipmi.power.py` script for the `NODEPOWERURL`.
4. Restart Moab and verify green computing is working correctly. If you encounter trouble, see the Troubleshooting green computing topic for help.

Related topics

- **Enabling green computing** on page 644
- **Troubleshooting green computing** on page 651
- **Adjusting green pool size** on page 649
- **Handling power-related events** on page 649
- **Maximizing scheduling efficiency** on page 650

16.2.3 Choosing which nodes Moab powers on or off

Context

Moab can use the GREENPOOLPRIORITYF function to determine which nodes to power on or off. The PRIORITY node allocation policy is used to determine which nodes to allocate workload to. When Moab can no longer allocate workload to available nodes, it begins to power nodes on in the order specified by the GREENPOOLPRIORITYF function.

To choose which nodes Moab powers on or off

1. Set a GREENPOOLPRIORITYF function to describe which order nodes should be selected for power on/off actions. GREENPOOLPRIORITYF uses the PRIORITY node allocation policy options and syntax.

```
GREENPOOLPRIORITYF '10*RANDOM'
```

This tells Moab to randomly choose a node to power on to meet workload demands, and to randomly choose an idle node to power off to meet the MAXGREENSTANDBYPOOLSIZE goal.

To choose which nodes Moab allocates jobs to

1. Set a PRIORITY node allocation policy that uses power as the major factor. This causes Moab to allocate jobs to nodes that are already powered on. When no nodes are available to meet this policy, Moab uses the GREENPOOLPRIORITYF function to turn on nodes that are powered off.

```
NODEALLOCATIONPOLICY  PRIORITY
NODECFG[DEFAULT] PRIORITYF='10000*POWER + 10*RANDOM'
```

The nodes with the highest priority for workload are the nodes that are powered on. After that, Moab randomly allocates workload.

Related topics

- **Adjusting green pool size** on page 649
- **Maximizing scheduling efficiency** on page 650

16.2.4 Adjusting green pool size

Context

The MAXGREENSTANDBYPOOLSIZE parameter allows you to allocate the number of nodes to keep powered on in the standby pool. This is the number of idle nodes that are allowed be powered on and idle. As the workload changes, Moab turns nodes on or off to try to meet this goal. The default value is 0.

To adjust the green pool size

1. Modify the MAXGREENSTANDBYPOOLSIZE parameter with the number of nodes you want Moab to keep powered on for the standby pool.

```
MAXGREENSTANDBYPOOLSIZE 10
```

Moab keeps up to 10 idle nodes powered on to be kept on standby.

Related topics

- **Maximizing scheduling efficiency** on page 650
- **Choosing which nodes Moab powers on or off** on page 648

16.2.5 Handling power-related events

Context

Power actions are considered NODEMODIFYURL events and are not recorded by default, but you can configure Moab to include power-related events in the logs. Also, if a node fails to turn on (or off), it's best to associate a reservation on the failed node so that Moab won't keep trying to perform the power action over and over.

To configure Moab to record power-related events

1. Modify the **RECORDEVENTLIST** parameter.

```
RECORDEVENTLIST +NODEMODIFY
```

Power-related events are logged to the Moab log file.

To put a reservation on a node that fails to perform a power action

1. Configure the NODEFAILURERSVPROFILE attribute of **RMCFG** and create an **RSVPROFILE** with a high duration.

```
RMCFG[torque] NODEFAILURERSVPROFILE=failure
RSVPROFILE[failure] DURATION=3600
```

Nodes that fail to power on or off have a 1-hour reservation placed on them.

Related topics

- **RECORDEVENTLIST** on page 898
- **Event Logs** on page 591

16.2.6 Maximizing scheduling efficiency

Context

When considering whether to power a node on or off, Moab can take into account the amount of time that it takes to power on or power off the node. With this information, Moab can keep an idle node powered on if it knows that workload in the queue will be ready for the node in less time that it takes to power off/power on the node.

Moab can also wait to shut down nodes after they've been idle for a specific amount of time.

To specify node power on/power off duration

1. Modify the **NODEPOWERONDURATION** and **NODEPOWEROFFDURATION** attributes of **PARCFG** with the maximum amount of time it takes for your nodes to power on/power off. Make sure to use the keyword `ALL` for the resource manager name to avoid cases where Moab won't consider the power on/off duration for a node before making a power action decision.

```
PARCFG[ALL] NODEPOWERONDURATION=2:00
PARCFG[ALL] NODEPOWEROFFDURATION=2:00
```

If a node goes idle and has to wait for workload, Moab will not power off the node if the workload will be available within 4 minutes or less.

To shut down on nodes after they've been idle for a specified time

1. Modify the NODEIDLEPOWERTHRESHOLD parameter with the duration (in seconds) you want Moab to wait before shutting down an idle node. The default value is 60 seconds. Increasing the number should decrease power on/off thrashing. This should be set higher than **NODEPOWERONDURATION** and/or **NODEPOWEROFFDURATION**.

```
NODEIDLEPOWERTHRESHOLD 300
```

Moab will wait 5 minutes before shutting down a node that has become idle.

Related topics

- **Adjusting green pool size** on page 649
- **Choosing which nodes Moab powers on or off** on page 648

16.2.7 Troubleshooting green computing

Context

If you've enabled green computing and are having trouble, here are some tips that can help you determine the cause of the issues you encounter. These tips are specifically for the Adaptive Computing supplied IPMI scripts, but can be generalized for whatever power management solution you use. Simply substitute your power management system, power query script (as specified by **CLUSTERQUERYURL**), and power action script (as specified by **NODEPOWERURL**) where appropriate.

Verify your IPMI access

1. Use the ipmitool command to verify you have access to the IPMI interface of your nodes. Try getting the current power state of a node. The syntax is `ipmitool -I lan -H <host> -U <IPMI username> -P <IPMI password> chassis power status`.

```
$ ipmitool -I lan -H qt06 -U ADMIN -P ADMIN chassis power status

Chassis Power is off
```

Verify the power query (CLUSTERQUERYURL) script is working

1. Execute the `impi.mon.py` script (should be found in `/<MOABHOMEDIR>/tools/ipmi`) to start the monitor.

```
$ cd /opt/moab/tools/ipmi
$ ./ipmi.mon.py
```

2. Execute the script again. The following is an example of the expected output:

```
$ ./ipmi.mon.py

qt09  GMETRIC[System_Temp]=27 GMETRIC[CPU_Temp]=25 POWER=on State=Unknown
qt08  GMETRIC[System_Temp]=31 GMETRIC[CPU_Temp]=25 POWER=on State=Unknown
qt07  GMETRIC[System_Temp]=30 GMETRIC[CPU_Temp]=29 POWER=on State=Unknown
qt06  GMETRIC[System_Temp]=Disabled GMETRIC[CPU_Temp]=Disabled POWER=off
State=Unknown
```

If the POWER attribute is not present the script is not working correctly.

Verify the power action (NODEPOWERURL) script is working

1. Execute the `ipmi.power.py` script (should be found in `/<MOABHOMEDIR>/tools/ipmi`) to see if you can force a node to power on or off. The syntax is `ipmi.power.py <node>,<node>,<node>... [off|on]`

```
$ /opt/moab/tools/ipmi/ipmi.power.py qt06 off
```

This example is trying to power off a node named qt06.

2. Verify the machine's power state was changed to what you attempted in the previous step. You can do this remotely via two methods:

a. If the cluster query script is working, you can use that to verify the current power state of the node.

b. If you have IPMI access, you can use the ipmitool command to verify the current power state of the node.

Verify the scripts are configured correctly

1. Run the mdiag -R command to verify your IPMI resource manager configuration.

```
$ mdiag -R -v
RM[ipmi]      State: Active  Type: NATIVE  ResourceType: PROV
Timeout:            30000.00 ms
Cluster Query URL:  exec://$TOOLSDIR/ipmi/ipmi.mon.py
Node Power URL:     exec://$TOOLSDIR/ipmi/ipmi.power.py
Objects Reported:   Nodes=3 (0 procs)  Jobs=0
Nodes Reported:     3 (N/A)
Partition:          SHARED
Event Management:   (event interface disabled)
RM Performance:     AvgTime=0.05s  MaxTime=0.06s  (176 samples)
RM Languages:       NATIVE
RM Sub-Languages:   NATIVE
```

2. Run the mdiag -G command to verify that power information is being reported correctly.

```
$ mdiag -G

NodeID      State      Power   Watts  PWatts
 qt09       Idle       On      0.00   0.00
 qt08       Idle       On      0.00   0.00
 qt07       Idle       Off     0.00   0.00
```

Verify the scripts are running

1. Once green is configured and Moab is running, Moab should start the power query script automatically. Use the ps command to verify the script is running.

```
$ ps -ef | grep <CLUSTERQUERYURL script name>
```

If this command does not show the power query script running then your settings in `moab.cfg` aren't working.

Verify Moab can power nodes on or off

1. Use the mnodectl command to turn a node on or off. The syntax is `mnodectl -m power=[off|on] <node>`.

```
mnodectl -m power=off qt06
```

Moab should turn off the node named qt06.

a. Moab generates a system job called `poweron-<num>` or `poweroff-<num>` job as shown in showq. The system job calls the `ipmi.power.py` (**NODEPOWERURL**) script to execute the command.

b. Moab waits until the cluster query reports the correct data. In this case, the `ipmi.power.py` script reports that the power attribute has changed.

c. Moab does not change the power status based on the power script return code. Rather, Moab completes the system power job when it detects the power attribute has changed as indicated by the cluster query script.

Related topics

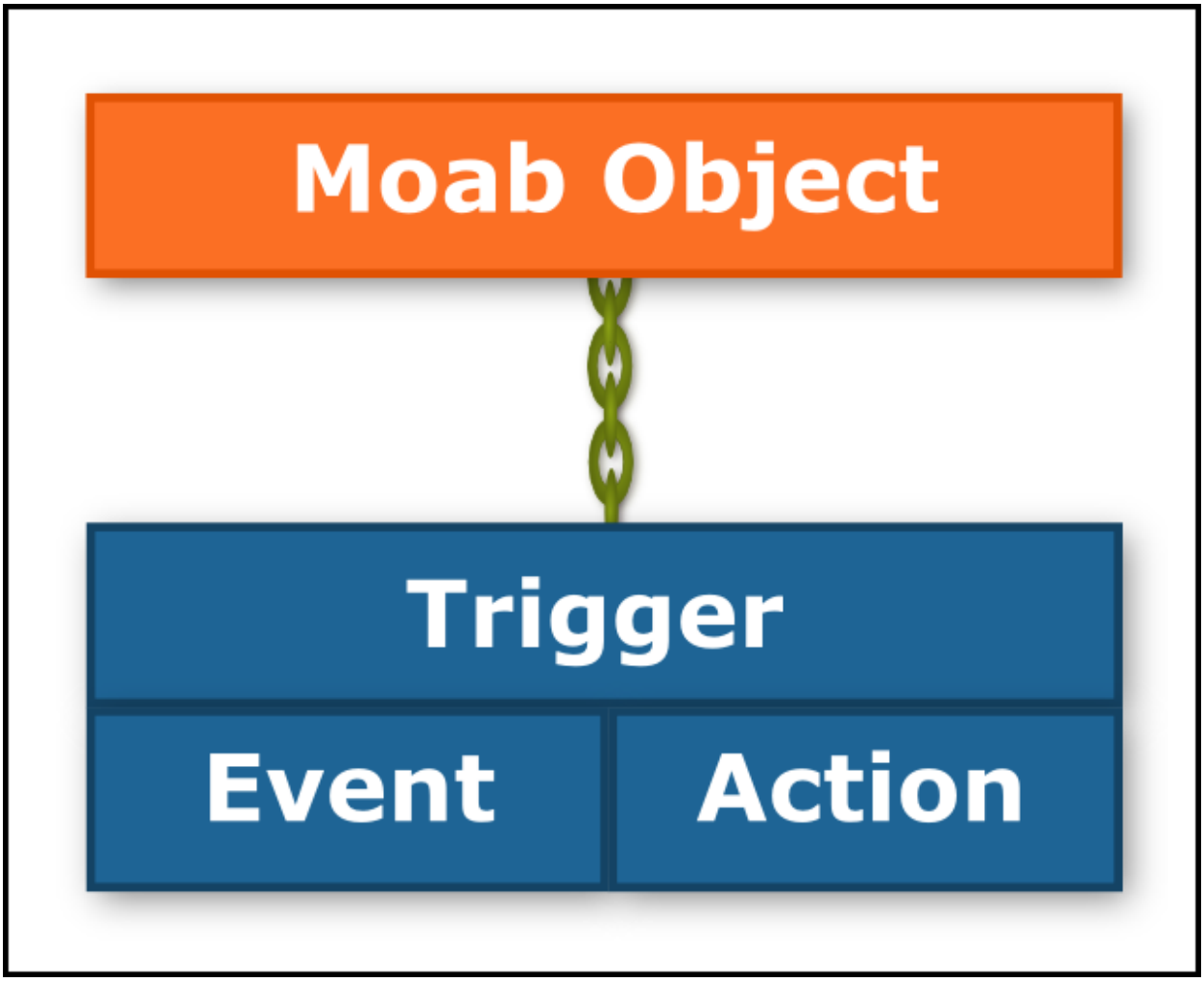
- **Enabling green computing** on page 644
- **Deploying Adaptive Computing IPMI scripts** on page 647

17.0 Object triggers

17.1 About object triggers

Moab triggers are configurable actions that respond to an event occurring on a Moab object. A trigger is attached to an object and consists of both an event that may take place on the object and the action that the trigger will take.

Image 17-1: Trigger attachment



Triggers are a powerful tool. Extreme caution should be taken when using them. They are useful in creating automatic responses to well-understood Moab events; however, by default triggers run as root and do exactly as they are told, meaning they require great thought and consideration to ensure that they act appropriately in response to the event.

Use case

An administrator wants to create the following setup in Moab:

When a node's temperature exceeds 34°C, Moab reserves it. If the temperature increases to more than 40°C, Moab requeues all jobs on the node. If the node's temperature exceeds 50°C, Moab shuts it down. Moab removes the node's reservation and unsets the variables when the node cools to less than 25°C.

The administrator wants to receive an email whenever any of these events occur. All of this can be configured in Moab using triggers. To see a full example for this use case, see **Node maintenance example** on page 682.

Sub content

- **About trigger variables** on page 684

How-to's

- **Creating a trigger** on page 658
- **Using a trigger to send email** on page 662
- **Using a trigger to execute a script** on page 663
- **Using a trigger to perform internal Moab actions** on page 663
- **Requiring an object threshold for trigger execution** on page 664
- **Enabling job triggers** on page 664
- **Modifying a trigger** on page 665
- **Viewing a trigger** on page 666
- **Checkpointing a trigger** on page 667

References

- **Job triggers** on page 667
- **Node triggers** on page 668
- **Reservation triggers** on page 670
- **Resource manager triggers** on page 671
- **Scheduler triggers** on page 672
- **Threshold triggers** on page 673
- **Trigger components** on page 674
- **Trigger exit codes** on page 682
- **Node maintenance example** on page 682
- **Environment creation example** on page 683

17.2 How-to's

17.2.1 Creating a trigger

Context

Three methods exist for attaching a trigger to an object:

- Directly to the object via the command line
- Directly to the object via the configuration file
- As part of a template via the configuration file

<attr>=<val> pair delimiters, quotation marks, and other elements of the syntax may differ slightly from one method/object combination to another, but creating any trigger follows the same basic format:

<attr>=<val>[[{&,}*<attr>=<val>*]...]

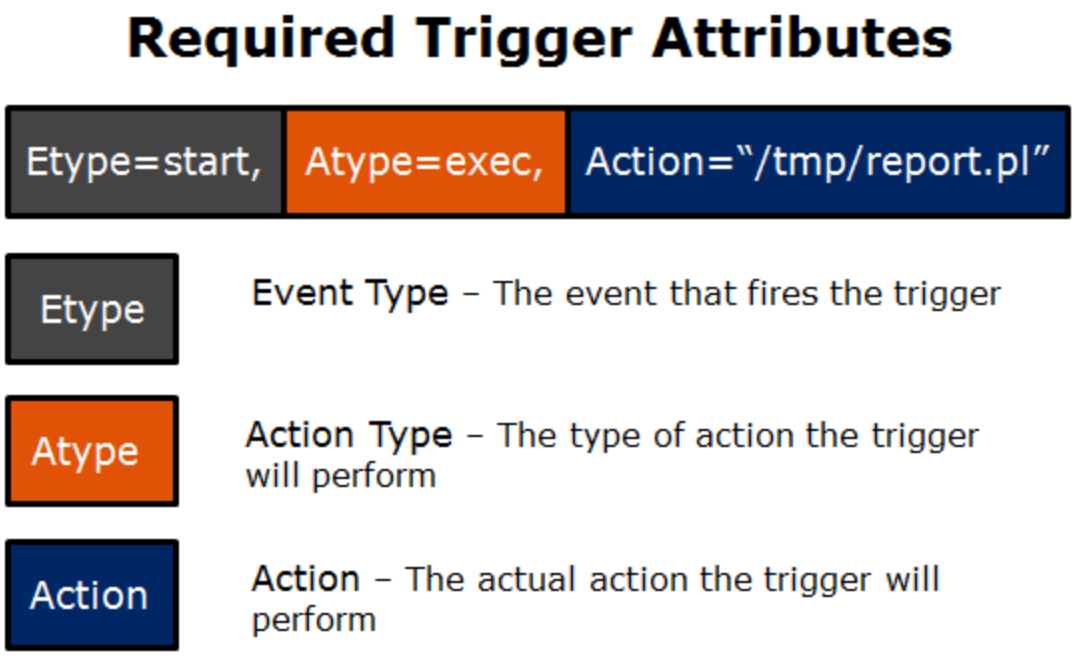
The beginning of the trigger is set off by the keyword *trigger*. It is followed by a delimited list (typically by commas) of *<attr>=<val>* pairs.

Each method of trigger creation can only be used for certain Moab objects. The following table displays which objects can receive triggers via each method. The links contain examples.

Method	Objects
Command line	job, reservation; a trigger can be attached to any existing object using mschedctl -c
Configuration file	node, reservation, RM, scheduler
Template	job, reservation

Triggers are composed of attributes. Only three are required for each trigger: an EType (event type), an AType (action type), and an Action.

Image 17-2: Required trigger attributes



Other attributes exist to further customize triggers. See **Trigger components** on page 674 for more information.

To create a Moab trigger

1. Choose an object to which, and a method by which, you will attach the trigger. Use the format and examples described in its corresponding documentation:
   - **Job triggers** on page 667
   - **Node triggers** on page 668
   - **Reservation triggers** on page 670
   - **Resource manager triggers** on page 671
   - **Scheduler triggers** on page 672

   If the trigger is to be attached to a job, you must first enable job triggers (see **Enabling job triggers** on page 664 for more information.). Please carefully review the warning before doing so.

2. Decide whether to attach the trigger via the command line or configuration file. Verify the correct syntax.

3. Set the **EType** equal to whichever event will launch the trigger if and when it occurs on the object.

   Each object has a different lifecycle, so not every event type will occur on every object. For a list of valid **ETypes** for your selected object, see the corresponding object reference page linked in step 1.

   a. To modify the timing of the trigger in any of the following ways, see **Event-modifying trigger components** on page 678.

      - To set the trigger as rearmable and specify the amount of time the trigger must wait before firing again.
      - To set an amount of time before or after the event that the trigger will fire (See **Offset** on page 678 for restrictions).
      - To set a specific threshold and the amount of time that the object must meet that threshold before the trigger will fire.

4. Configure the action that the trigger will take when the event happens. To do so, you must set the **AType** to a valid value for your object and specify the action. For instance, to execute a script, set the **AType** to *exec* and the **Action** to the location of the script in quotation marks. Include the name of the object on which the script will run.

   ```
   NODECFG[node01] TRIGGER=EType=fail,AType=exec,Action="node.fail.sh node01"
   ```

   a. To modify the action in any of the following ways, see **Action-modifying trigger components** on page 680.

      - To specify environment variables available to the trigger
      - To set a flag on the trigger
        - To attach any stderr output generated by the trigger to the parent object
        - To destroy the trigger if its object ends or cancels
        - To tell Moab to checkpoint the trigger
        - To set the trigger as periodic
        - To pass the object's XML information to the trigger's stdin
        - To set the trigger to reset if its object is modified
        - To set the trigger to fire under the user ID of the object's owner
      - To specify an amount of time that Moab will suspend normal operation to wait for the trigger to execute
      - To allot an amount of time that the trigger will attempt to run before it is marked as unsuccessful and the process, if any exists, is killed
      - Set a maximum number of times that a trigger will attempt to fire before it fails

   b. To give the trigger a name or description, see **Organizational trigger components** on page 681.

   c. To configure the trigger to set or unset a variable when it fires or to require a variable to fire, see **Setting and receiving trigger variables** on page 685.

17.2.2 Using a trigger to send email

Context

Mail triggers can be attached to nodes, jobs, reservations, and the scheduler. The recipient of the email depends on the object to which the trigger is attached. To select different recipient(s) and add flexibility to formatting, send email via a script using an exec trigger.

To use a trigger to send email

1. For objects that send mail to the primary user, you must configure the Moab administrator email using the **MAILPROGRAM** on page 867 parameter.
2. Create a trigger on one of the four valid objects listed below, setting the **AType** to mail and the **Action** to the body of the message inside of quotation marks.

Object	Recipient
Node	The primary user (the first user listed in ADMINCFG[1], typically root)
Job	The job's owner
Reservation	The primary user
Scheduler	The primary user

3. When attaching a mail trigger to all objects of a certain type, use internal variables in the **Action** to add information that is specific to an object, such as the ID, owner, time the event occurred, etc. A variable must be preceded by a dollar sign ($).

Variable	Description
$OID	Name of the object to which the trigger is attached
$OTYPE	The type of object to which the trigger is attached
$TIME	Time the trigger launched
$HOSTLIST	Host list of the trigger's object (jobs and reservations)
$OWNER	Owner of the trigger's object (jobs and reservations)
$USER	User (jobs and reservations)

The variable is replaced with the information described above. For example, the following trigger is configured on all nodes:

```
NODECFG[DEFAULT] EType=fail,AType=mail,Action="node $OID failed at $TIME"
```

When, for example, node `node03` fails, an email is sent to the primary user with a message with the subject line "node node03 started on Sat Aug 18 11:42:00".

17.2.3 Using a trigger to execute a script

Context

Exec triggers launch a program or script when the event occurs. A few examples of what a script might do in response to an event include:

- Execute an external program
- Send a complex email to any desired recipient(s)
- Collect diagnostics

It is important to note that when a script runs via a trigger, Moab forks and performs a direct OS exec, meaning there will be no pre-processing of the command by the shell. In addition, the script runs in a new, reduced environment without the same settings and variables as the environment from which it stemmed. The script must be able to run in the reduced environment.

To use a trigger to execute a script

1. Create or locate the script and note its location.
2. Create a trigger on the desired object, setting the **AType** to *exec* and the **Action** to location of the script or program.

```
JOBCFG[temp1] TRIGGER=EType=start,AType=exec,Offset=03:00,Action="/tmp/monitor.pl"
```

Jobs with the `temp1` template receive a trigger that executes `monitor.pl` three minutes after the job starts.

17.2.4 Using a trigger to perform internal Moab actions

To perform internal actions in Moab with a trigger

- Create a trigger on a job, node, or reservation, setting the **AType** to *internal* and the **Action** to one of the following:
  - *node:-:reserve* - reserves the node to which the trigger is attached
  - *job:-:cancel* - cancels the job to which the trigger is attached
  - *reservation:-:cancel* - cancels the reservation to which the trigger is attached

The specified object reserves or cancels itself once the event occurs. See **Internal Action** on page 676 for examples.

17.2.5 Requiring an object threshold for trigger execution

Context

Threshold triggers allow sites to configure triggers to launch based on internal scheduler statistics, such as generic metrics. For example, you might configure a trigger to warn the administrator when the percentage of nodes available is less than 25.

To configure a threshold trigger

1. Create a trigger. Set its **EType** to *threshold*. Configure the **AType**, **Action**, and **Threshold** attributes' values based on the valid thresholds per object listed in the table found in **Threshold triggers** on page 673.

```
NODECFG[node04] TRIGGER=EType=threshold,AType=exec,Action="$HOME/hightemp.py
$OID",Threshold=gmetric
```

2. Insert the gmetric name between brackets (such as `gmetric[temp]`). Provide a comparison operator. For valid options, see the comparison operators table.
3. Provide a number or string to match against the threshold.

```
NODECFG[node04] TRIGGER=EType=threshold,AType=exec,Action="$HOME/hightemp.py
$OID",Threshold=gmetric[TEMP]>70,RearmTime=5:00
```

Moab launches a script that warns the administrator when `node04`'s gmetric `temp` exceeds 70. Moab rearms the trigger five minutes after it fires.

17.2.6 Enabling job triggers

Context

By default common users cannot create most objects, and as a result, common users also cannot create triggers. The exception, however, is jobs. Because common users can create jobs and triggers generally run as root, additional security is necessary to ensure that not all users can create triggers. For this reason, job triggers are disabled by default.

Because triggers generally run as root, any user given the power to attach triggers has the power to run scripts and commands as root. It is recommended that you only enable job triggers on closed systems in which human users do not have access to directly submit jobs.

To give specific users permission to create job triggers, you must create a QoS, set the *trigger* flag, and add users to it.

To enable job triggers

1. In the `moab.cfg` file, create a QoS and set the *trigger* flag.

```
QOSCFG[triggerok]   QFLAGS=trigger
```

2. Add users to the QoS who should be allowed to add triggers to jobs.

```
USERCFG[joe]        QDEF=triggerok
```

User `joe` is added to the *triggerok* QoS, giving him both the power to create job triggers and root access to the machine.

17.2.7 Modifying a trigger

Context

You can modify a trigger at any time by updating its settings in the Moab configuration file (`moab.cfg`). This will update most triggers at the beginning of the next Moab iteration; however, modifying template triggers (configured using RSVPROFILE or JOBCFG) will not update the instances of the trigger that were attached to individual reservations or jobs on creation. The modification will only affect the triggers that the template attaches to future objects.

Any trigger with a specified name can be modified using the mschedctl -m command in the following format:

`mschedctl -m trigger: <triggerID><attr1>=<val1><attr2>=<val2>`

> Modifying triggers on the command line does not change their configuration in `moab.cfg`. Except for reservations that are checkpointed, changes made dynamically are lost when Moab restarts.

For example, the procedure below demonstrates how to modify the following trigger so that the offset is 10 minutes instead of 5 and so that Moab will attempt to fire the trigger up to 10 times if it fails. Assume your trigger currently looks like this:

```
NODECFG[DEFAULT] EType=fail,AType=exec,Action="/scripts/node_
fail.pl",Name=nodeFailTrig,Offset=00:05:00,MultiFire=TRUE,RearmTime=01:00:00
```

To modify a trigger

1. Type mschedctl -m into the command line and set off the trigger modification with `trigger:<id>`. Use the trigger's assigned ID or specified name to state which trigger will receive the modification.

```
> mschedctl -m trigger:nodeFailTrig
```

2. Type any changing attributes equal to the new value. Separate multiple modifications with a space between each *<attr>=<val>* pair. In this case, set the **Offset** and **MaxRetry** attributes the following way:

```
> mschedctl -m trigger:nodeFailTrig Offset=00:10:00 MaxRetry=10
```

The newly-specified attributes replace the original ones. Trigger `nodeFailTrig` now has an offset of 10 minutes and will try to fire a maximum of 10 times if it fails. The new trigger has the following attributes:

```
EType=fail,AType=exec,Action="/scripts/start_
rsv.pl",Name=nodeFailTrig,Offset=00:10:00,MultiFire=TRUE,RearmTime=01:00:00,MaxRetr
y=10
```

17.2.8 Viewing a trigger

Context

Moab provides a list of triggers when you run the mdiag -T command. You can view a specific trigger by running mdiag -T in the following format:

`mdiag -T [<triggerID>|<objectID>|<triggerName>|<objectType>]`

To view a trigger

1. Type mdiag -T in the command line.
2. Specify either the trigger ID, the trigger name, the name of the object to which the trigger is attached, or the type of object to which the trigger is attached. For example, if you wanted to view information about a trigger with ID `trigger.34` and name `jobFailTrigger`, which is attached to job `job.493`, you could run any of the following commands:

```
> mdiag -T trigger.34

> mdiag -T job.493

> mdiag -T jobFailTrigger

> mdiag -T job
```

The output of the first command would provide basic information about `trigger.34`; the second command, information about all triggers attached to `job.493` that the user can access; the third command, basic information about `jobFailTrigger`; and the fourth command, basic information about all triggers attached to jobs that the user can access.

3. Optional: to view additional information about the trigger, run the same command with the `-v` flag specified after `-T`.

```
> mdiag -T -v job.493
```

This mode outputs information in multiple lines.

4. Optional: to view detailed information about all triggers available to you, use the mdiag -T -v command. This outputs all triggers available to the user in a single line for each trigger. It provides additional state information about triggers, including reasons triggers are currently blocked.

```
> mdiag -T -v
```

17.2.9 Checkpointing a trigger

Context

Checkpointing is the process of saving state information when Moab is shut down. In general, triggers defined in the `moab.cfg` file are not checkpointed but are recreated when Moab starts. The exception is the JOBCFG parameter, which attaches triggers to jobs as they are created. There are two cases in which you may want to tell Moab to checkpoint a trigger:

- If a trigger is defined in the `moab.cfg` file but was created at the command line
- When creating a trigger using the **mschedctl** on page 184 command

To checkpoint a trigger

1. Locate the trigger to be checkpointed in the `moab.cfg` file, create one on the command line, or modify a trigger dynamically (See **Modifying a trigger** on page 665 for more information). Attach the *checkpoint* flag using the **FLAGS** attribute. For more information about flags, see **Flags** on page 680.

   ```
   FLAGS=checkpoint
   ```

2. If you are working in the configuration file, save the changes. Moab will now checkpoint your trigger.

17.3 References

17.3.1 Job triggers

For security reasons, job triggers are disabled by default. They must be enabled in order to successfully attach triggers to jobs (See **Enabling job triggers** on page 664 for more information.).

Triggers attached to jobs follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. Jobs, like other objects, have a unique set of trigger rules. The table below details the methods, options, and other notable details associated with attaching triggers to jobs.

Creation methods

Method	Format	Example
Command line on job creation: msub -l	`msub <jobName> -l 'trig=<trigSpec>'` Attributes are delimited by backslash ampersand (\&).	`> msub my.job -l 'trig=EType=create\&AType=exec\&Action="/jobs/my_trigger.pl"\&Offset=10:00'`

Method	Format	Example
Command line on existing job: mschedctl -c	`mschedctl -c trigger <trigSpec> -o job:<jobID>`	`> mschedctl -c trigger EType=end,AType=mail,Action="Job moab.54 has ended" -o job:moab.54`
Job template in moab.cfg : JOBCFG	`JOBCFG[<templateName>] TRIGGER=<trigSpec>`	`JOBCFG[vmcreate] TRIGGER=,EType=end,AType=exec,Action="/tmp/jobEnd.sh"`

Valid event types

- **cancel** on page 677
- **checkpoint** on page 677
- **create** on page 677
- **end** on page 677
- **hold** on page 678
- **modify** on page 678
- **preempt** on page 678
- **start** on page 678

Valid action types

- changeparam
- exec
- internal
- mail

Mail recipient

The job's owner

See **Using a trigger to send email** on page 662 for more information.

17.3.2 Node triggers

Triggers attached to nodes follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. Nodes, like the other objects, have a unique set of trigger rules. The table below details the methods, options, and other notable details that come with attaching triggers to nodes.

Creation methods

Method	Format	Example
Command line on existing node: msched-ctl - c	`mschedctl -c trigger <trigSpec> -o node:<nodeID>`	`> mschedctl -c trigger EType=fail,AType=exec,Action="/tmp/nodeFailure.sh" -o node:node01`
Node configuration in moab.cfg: NODECFG	`NODECFG [<name>] TRIGGER= <trigSpec>`	`NODECFG[node04] TRIGGER=EType=threshold,AType=exec,Action="$HOME/hightemp.py $OID",Threshold=gmetric[TEMP]>70`

Valid event types

- **create** on page 677
- **discover** on page 677
- **end** on page 677
- **fail** on page 677
- **standing** on page 678
- **threshold** on page 678

Valid action types

- changeparam
- exec
- internal
- mail

Thresholds

Node threshold settings	
Valid ETypes	threshold
Valid Threshold types	gmetric

Mail recipient

The user listed first in ADMINCFG[1] (usually `root`)

See **Using a trigger to send email** on page 662 for more information.

17.3.3 Reservation triggers

Triggers attached to reservations follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. Reservations, like the other objects, have a unique set of trigger rules. The table below details the methods, options, and other notable details that come with attaching triggers to reservations.

Creation methods

Method	Format	Example
Command line on reservation creation: mrsvctl -T	`mrsvctl -c -h <host-list> -T <trigSpec>`	`> mrsvctl -c -h node01 -T EType=start,AType=exec, Action="/scripts/node_start.pl"`
Command line on existing reservation: mschedctl - c	`mschedctl -c trigger <trigSpec> -o rsv:<rsvID>`	`> mschedctl -c trigger EType=modify,AType=mail,Action="Reservation system.4 has been modified" -o rsv:system.4`
Standing reservation configuration in moab.cfg : SRCFG	`SRCFG [<name>] TRIGGER= <trigSpec>`	`SRCFG[Mail2] TRIGGER=EType=start,Offset=200,AType=exec,Action="/tmp/email.sh"`
Reservation template in moab.cfg : RSVPROFILE	`RSVPROFILE [<name>] TRIGGER= <trigSpec>`	`RSVPROFILE[rsvtest] TRIGGER=EType=cancel,AType=exec,Action="$HOME/logdate.pl TEST CANCEL $OID $HOSTLIST $ACTIVE"`

Valid event types

- **create** on page 677
- **end** on page 677
- **modify** on page 678
- **standing** on page 678
- **start** on page 678
- **threshold** on page 678

Valid action types

- cancel
- changeparam
- exec
- internal
- jobpreempt
- mail

Thresholds

Node threshold settings	
Valid ETypes	threshold
Valid Threshold types	usage

Mail recipient

The owner of the reservation. If the owner is unknown or not a user, the first user listed first in ADMINCFG (usually `root`).

See **Using a trigger to send email** on page 662 for more information.

17.3.4 Resource manager triggers

Triggers attached to the resource manager follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. The resource manager, like other objects, has a unique set of trigger rules. The table below details the methods, options, and other notable details that come with attaching triggers to RMs.

Creation methods

Method	Format	Example
Command line on existing RM: mschedctl -c	`mschedctl -c trigger <trigSpec> -o rm:<rmID>`	`> mschedctl -c trigger EType=start,AType=exec,Action="/tmp/rmStart.sh" -o rm:torque`
RM configuration in moab.cfg: RMCFG	`RMCFG [<name>] TRIGGER= <trigSpec>`	`RMCFG[base] TRIGGER=EType=fail,AType=exec,Action="/opt/moab/tools/diagnose_rm.pl $OID"`

Valid event types

- **fail** on page 677
- **threshold** on page 678

Valid action types

- changeparam
- exec
- internal

17.3.5 Scheduler triggers

Triggers attached to the scheduler follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. The scheduler, like the other objects, has a unique set of trigger rules. The table below details the methods, options, and other notable details associated with attaching triggers to the scheduler.

Creation methods

Method	Format	Example
Command line on existing scheduler: msched-ctl - c	`mschedctl -c trigger <trigSpec> -o sched:<schedID>`	`> mschedctl -c trigger EType=end,AType=exec,Action="/tmp/startRsvs.sh" -o sched:moab`
Scheduler configuration in `moab.cfg` : SCHEDCFG	`SCHEDCFG[<name>] TRIGGER=<trigSpec>`	`SCHEDCFG[MyCluster] TRIGGER=EType=fail,AType=mail,Action="scheduler failure detected on $TIME",RearmTime=15:00`

Valid event types

- **create** on page 677
- **end** on page 677
- **fail** on page 677
- **modify** on page 678
- **standing** on page 678
- **start** on page 678

Valid action types

- changeparam
- exec
- internal
- mail

Mail recipient

The user listed first in ADMINCFG (usually `root`)

See **Using a trigger to send email** on page 662 for more information.

17.3.6 Threshold triggers

The following table identifies the object event, and usage types with which the threshold event/action type feature works.

Object type	Event Type	Usage types
Node	Threshold	gmetric
Reservation	Threshold	usage

The following table defines each of the usage types:

Usage type	Description
gmetric	Generic performance metrics configured in Moab (See Enabling Generic Metrics for more information).
usage	The percentage of the resource being used (not idle).

The following table defines each of the threshold trigger comparison operators:

Comparison operator	Value
>	Greater than
>=	Greater than or equal to
<	Less than

Comparison operator	Value
<=	Less than or equal to
==	Equal to

Examples

Example 17-1: Reservation usage threshold

```
SRCFG[res1] TRIGGER=EType=threshold,AType=mail,Action="More than 75% of reservation
res1 is being used",Threshold=usage>75,FailOffset=1:00
```

When more than 75% of the reservation has been in use for at least a minute, Moab fires a trigger to notify the primary user.

17.3.7 Trigger components

Required trigger components

AType

Action type	Description
cancel	Cancels the object
changeparam	Causes Moab to give a parameter to a new value
exec	Launches an external program or script on the command line when the dependencies are fulfilled. See **Using a trigger to execute a script** on page 663 for more information.
internal	Modifies Moab without using the command line. See **Using a trigger to perform internal Moab actions** on page 663 for more information.
jobpreempt	Indicates the preempt policy to apply to all jobs currently allocated resources assigned to the trigger's parent reservation
mail	Causes Moab to send mail. See **Using a trigger to send email** on page 662 for more information.

Action

Cancel Action	
Format	NONE
Description	Indicates that Moab should cancel the reservation when the event occurs. No action should be specified.
Example	`Etype=threshold,Threshold=usage<10,FailOffset=1:00,AType=cancel` *When less than 10% of the reservation has been in use for a minute, Moab cancels it.*

Changeparam Action	
Format	`Action="<STRING>"`
Description	Specifies the parameter to change and its new value (using the same syntax and behavior as the **changeparam** on page 272 command)
Example	`Atype=changeparam,Action="JOBCPURGETIME 02:00:00"` *Moab maintains detailed job information for two hours after a job has completed.*

Jobpreempt Action	
Format	`Action="cancel|checkpoint|requeue|suspend"`
Description	Signifies PREEMPTPOLICY to apply to jobs that are running on allocated resources
Example	`RSVPROFILE[adm1] TRIGGER=EType=start,Offset=-240,AType=jobpreempt,Action="cancel"` *40 minutes after the reservation `adm1` starts, all jobs using the reservation's resources adopt a PREEMPTPOLICY of cancel.*

Mail Action	
Format	`Action="<MESSAGE>"`

Mail Action

Description

When **AType**=*mail*, the **Action** parameter contains the message body of the email. This can be configured to include certain variables. See **Using a trigger to send email** on page 662 for details.

Mail triggers can be configured to launch for node failures, reservation creation or release, scheduler failures, and even job events. In this way, site administrators can keep track of scheduler events through email.

The email comes from *moabadmin*, has a subject of *moab update*, and has a body of whatever you specified in the **Action** attribute. The recipient list depends on the type of object the trigger is attached to.

- **Node** - The primary user (first listed in ADMINCFG[1]), typically `root`
- **Scheduler** - The primary user
- **Job** - The user who owns the job
- **Reservation** - The primary user

Example

```
NODECFG[DEFAULT] TRIGGER=EType=fail,AType=mail,Action="node $OID will
failed.",Offset=05:00:00
```

This example sends an email to the primary administrator informing him/her that the node (including the node ID) has failed.

Exec Action

Format

`Action="<script>"`

Description

Exec triggers will launch an external program or script when their dependencies are fulfilled. The following example will submit `job.cmd` and then execute `monitor.pl` three minutes after the job is started. See **Using a trigger to execute a script** on page 663 for more information.

Example

```
> msub -l trig=EType=start\&AType=exec\&Action="/tmp/monitor.pl"
job.cmd\&Offset=03:00
```

Internal Action

Format

`Action="<objectType>:-:<cancel|reserve>"`

Internal Action	
Description	A couple different actions are valid depending on what type of object the internal trigger is acting upon. The following list shows the available actions: • Reserve a node • Cancel a job • Cancel a reservation See **Using a trigger to perform internal Moab actions** on page 663 for more information.

Example	`NODECFG[node01] TRIGGER=EType=start,AType=internal,Action="node:-:reserve"`
	When `node01` starts, it becomes a reservation.
	`> msub moab.3 -l 'trig=EType=fail\&AType=internal\&Action="job:-:cancel"`
	If `moab.3` fails, Moab cancels it.
	`> mrsvctl -c -a user==joe -h node50 -T EType=start,AType=internal,Action="reservation:-:cancel",Offset=10:00`
	User `joe`'s jobs are given a ten-minute window to start, then the reservation cancels.

EType

Event type	Description
cancel	The event is triggered when the parent object is either canceled or deleted.
checkpoint	Triggers fire when the job is checkpointed. *checkpoint* triggers can only be attached to jobs.
create	Triggers fire when the parent object is created. *create* triggers can be attached to nodes, jobs, reservations, and the scheduler (when attached to the scheduler, triggers fire when Moab starts).
discover	Triggers fire when the node is loaded from a resource manager and Moab cannot recorgnize it nor find it in the checkpoint file.
end	Triggers fire when the parent object ends. *end* triggers can be attached to nodes, jobs, reservations, and the scheduler (when attached to the scheduler, triggers fire when Moab shuts down).
fail	Triggers fire when the resource manager is in a corrupt or down state for longer than the configured fail time, or when Moab detects a corruption in a node's reservation table. *fail* triggers can be attached to jobs, nodes, resource managers, and the scheduler.

Event type	Description
hold	Triggers fire when the job is put on hold. *hold* triggers can only be attached to jobs.
modify	Triggers fire when the parent object is modified. *modify* triggers can be attached to jobs and reservations
preempt	Triggers fire when the job is preempted. *preempt* triggers can only be attached to jobs.
standing	Triggers fire multiple times based on a certain period. They can be used with Period and Offset attributes. *standing* triggers can be attached to nodes and the scheduler.
start	Triggers fire when the parent object or Moab starts. *start* triggers can be attached to jobs, reservations, resource managers, and the scheduler (when Moab starts and at the beginning of Moab's first iteration).
threshold	Triggers fire when a threshold, such as usage or a gmetric comparison, is true. *threshold* triggers can be attached to nodes and reservations. Triggers with **EType**s set to *threshold* must include the Threshold attribute.

Event-modifying trigger components

The following trigger attributes modify the event that causes the trigger to fire.

RearmTime	
Possible Values	`[[HH:]MM:]SS`
Description	The amount of time that must pass before a trigger can fire again. **RearmTime** is enforced from the trigger event time.
Usage Notes	---

Offset	
Possible Values	`[-][[HH:]MM:]SS`
Description	The relative time offset from event when trigger can fire

Offset	
Usage Notes	• Only end triggers can have a negative value for **Offset**. • **Offset** cannot be used with cancel.

Period	
Possible Values	*Minute*, *Hour*, *Day*, *Week*, *Month*, *Infinity*
Description	The period at which the trigger will regularly fire
Usage Notes	---

Threshold	
Possible Values	`Threshold={<metric>[<metricName>]}{> >= < <= ==}<FLOAT>` Where *<metric>* is one of: • **gmetric** • **usage**
Description	When the object meets, drops below, or increases past the configured **Threshold**, the trigger will fire.
Usage Notes	Threshold triggers allow sites to configure triggers to launch based on internal scheduler statistics, such as the usage of a reservation.

FailOffset	
Possible Values	`[[HH:]MM:]SS`
Description	The time that the threshold condition must exist before the trigger fires
Usage Notes	Use with fail triggers to avoid transient triggers.

Action-modifying trigger components

Flags	
Possible Values	`Flags=<flag>[:<flag>]` or `Flags=[<flag>][[<flag>]]` *attacherror* - If the trigger outputs anything to stderr, Moab attaches it as a message to the trigger object. *cleanup* - If the trigger is still running when the parent object completes or is canceled, Moab kills the trigger. *checkpoint* - Moab always checkpoints this trigger. For more information, see **Checkpointing a trigger** on page 667. *objectxmlstdin* - Trigger passes its parent's object XML information into the trigger's stdin. This only works for exec triggers with reservation type parents. *resetonmodify* - The trigger resets if its object is modified, even if **RearmTime** is not set. *user* - The trigger executes under the user ID of the object's owner. If the parent object is the scheduler, you may explicitly specify the user using the format `user+<username>`. For example: `Flags=user+john`.
Description	Specifies various trigger behaviors and actions
Usage Notes	When specifying multiple flags, each flag can be delimited by colons (:) or with square brackets; for example: `Flags=[user][cleanup]` or `Flags=user:cleanup`

BlockTime	
Possible Values	`[[HH:]MM:]SS`
Description	The amount of time Moab will suspend normal operation to wait for trigger execution to finish
Usage Notes	Use caution; Moab will completely stop normal operation until **BlockTime** expires.

ExpireTime	
Possible Values	*<INTEGER>*
Description	The time at which trigger should be terminated if it has not already been activated
Usage Notes	---

Timeout	
Possible Values	`[+|-][[HH:]MM:]SS`
Description	The time allotted to this trigger before it is marked as unsuccessful and its process (if any) killed
Usage Notes	---

MaxRetry	
Possible Values	`MaxRetry=<INTEGER>`
Description	The number of times **Action** will be attempted before the trigger is designated a failure
Usage Notes	If **Action** fails, the trigger will restart immediately (up to **MaxRetry** times). If it fails more than **MaxRetry** times, the trigger has failed. This restart ignores **FailOffset** and **RearmTime**.

Organizational trigger components

Name	
Possible Values	`Name=<STRING>`
Description	Name of the trigger
Usage Notes	Because Moab uses its own internal ID to distinguish triggers, the **Name** need not be unique. Only the first 16 characters of **Name** are stored by Moab.

Description	
Possible Values	`Description=<STRING>`
Description	Description of the trigger
Usage Notes	---

17.3.8 Trigger exit codes

By default Moab considers any non-zero exit code as a failure and marks the trigger as having failed. If a trigger is killed by a signal outside of Moab, Moab treats the signal as the exit code and (in almost all cases) marks the trigger as having failed. Only exec triggers that exit with an exit code of 0 are marked as successful.

17.3.9 Node maintenance example

Example scenario

An administrator wants to create the following setup in Moab:

When a node's temperature exceeds 34°C, Moab reserves it. If the temperature increases to more than 40°C, Moab requeues all jobs on the node. If the node's temperature exceeds 50°C, Moab shuts it down. Moab removes the node's reservation and unsets the variables when the node cools to less than 25°C. The administrator wants to receive an email whenever any of these events occur.

The first trigger reserves the node when its reported temperature exceeds 34°C. Note that the gmetric name in the trigger must match the name of the configured gmetric exactly, including its case (See **Enabling Generic Metrics** on page 492 for more information.).

```
NODECFG[DEFAULT] TRIGGER=Description="ThresholdA",EType=threshold,Threshold=gmetric
[temp]>34,AType=internal,Action="node:-:reserve",RearmTime=30,Offset=2:00,Sets=temp_
rsv
```

The administrator wants the trigger to fire any time a node overheats, so it must be rearmable. It also needs to specify that the node must be over 34°C for at least two minutes for Moab to reserve it. If the trigger succeeds, it will set a variable to be received by the next trigger in order to make them sequential.

The administrator wants to know when this trigger has fired, so another trigger will send an email once the first trigger has fired and the `temp_rsv` variable is set. This one does so via a script:

```
NODECFG[DEFAULT] Trigger=Description="Email on
Reservation",EType=start,AType=exec,Action="$TOOLSDIR/node_temp_emailReserve.pl
$OID",RearmTime=3:00,Requires=temp_rsv
```

The second threshold trigger requeues the node's jobs if the node exceeds 40°C and the `temp_rsv` variable is set. It uses a script to do so. It sets `node_evac` variable when it fires, regardless of whether it succeeds or fails.

```
NODECFG[DEFAULT] Trigger=Description="Threshold B",EType=threshold,Threshold=gmetric
[temp]>40,Atype=exec,Action="$TOOLSDIR/node_evacuate.pl
$OID",RearmTime=3:00,requires=temp_rsv,Sets=node_evac,!node_evac
```

The administrator wants another email to inform him that the node is still overheating and has been evacuated. Another email trigger fires once it receives the `node_evac` variable.

```
NODECFG[DEFAULT] Trigger=Description="Email on
Evacuation",EType=start,AType=exec,Action="$TOOLSDIR/node_temp_emailEvac.pl
$OID",RearmTime=3:00,Requires=node_evac
```

The third threshold trigger uses a script to shut down the node if the temp gmetric exceeds 50 and the node_evac variable is set. It sets a node_shutdown variable to be received by the notification email.

```
NODECFG[DEFAULT TRIGGER=Description="Threshold C",EType=threshold,Threshold=gmetric
[temp]>50,AType=exec,Action="$TOOLSDIR/node_shutdown.pl
$OID",RearmTime=3:00,Requires=node_evac,Sets=node_shutdown

NODECFG[DEFAULT] Trigger=Description="Email on
Shutdown",EType=start,AType=exec,Action="$TOOLSDIR/node_temp_emailShutdown.pl
$OID",RearmTime=3:00,Requires=node_shutdown
```

The final trigger removes the reservation and unsets the variables once the node's temp gmetric is less than 25.

```
NODECFG[DEFAULT] Trigger=Description="Remove
Reservation",EType=threshold,Threshold=gmetric[temp]
<25,AType=exec,Action="opt/moab/bin/mrsvctl -r r:$OID",RearmTime=3:00,Requires=temp_
rsv,unsets=temp_rsv.node_evac.node_shutdown
```

17.3.10 Environment creation example

Example scenario

An administrator wants to create the following setup in Moab:

If a user requests an environment, she must have the permission of her two managers and the administrator. If all three approve, then the environment builds. The user is sent email messages informing her of the environment's end date in case she would like an extension. These are sent 7, 3, and 1 days prior to the environment's ending.

The administrator wants to require his and the managers' approval of any modifications the user makes to her environment so that it cannot be extended without consent.

The first trigger requests manager and administrator approval in response to the user's environment request. So in the event of a reservation's creation, a script is used to send messages to the administrator and manager. The internal variable OWNER is used to indicate to the recipients (via the script) which user is requesting the environment.

```
RSVPROFILE[envSetup] TRIGGER=EType=create,AType=exec,Action="envRequest.sh $OWNER"
```

The managers and administrator use an external program to approve or reject the request. On approval, a variable is sent back to Moab (to the reservation specifically). Once all three variables are set, the environment can start. In this example, the variables are called approval1, approval2, and approval3.

```
RSVPROFILE[envSetup]
TRIGGER=EType=start,AType=exec,Action="buildScript",Requires=approval1.approval2.appro
val3
```

As it is configured now, the reservation will continue to reserve the requested resources regardless of whether all three approvals are given. So, in case approval is not given, the next trigger cancels the reservation 7 days after its creation if the three variables are not set.

```
RSVPROFILE[envSetup]
TRIGGER=EType=create,Offset=7:00:00,AType=internal,Action="rsv:-:cancel",Requires=!app
roval1.!approval2.!approval3
```

Every remaining trigger in this series is meant to fire for an approved environment and must require the approval variables. Otherwise these notifications would be sent to users who do not have the environment they requested. The next triggers must be rearmable so that it can fire again if necessary; however, they should be set to just over the amount of time left on the reservation so that it doesn't fire again for the same environment. The notification triggers use the **Offset** attribute to fire at the administrator's requested times (7, 3, and 1 day(s) prior to the environment's end).

```
RSVPROFILE[envSetup] TRIGGER=EType=end,Offset=-
7:00:00,AType=exec,Action="weekNotification.sh",RearmTime=7:00:00:02,Requires=approval
1.approval2.approval3

RSVPROFILE[envSetup] TRIGGER=EType=end,Offset=-
3:00:00,AType=exec,Action="3dayNotification.sh",RearmTime=3:00:00:02,Requires=approval
1.approval2.approval3

RSVPROFILE[envSetup] TRIGGER=EType=end,Offset=-
1:00:00,AType=exec,Action="dayNotification.sh",RearmTime=1:00:00:02,Requires=approval1
.approval2.approval3
```

The next trigger requests administrator and manager approval when the environment is modified. The problem is that the trigger must be rearmable in case of multiple modifications and each time the **RearmTime** on page 678 is reached, Moab will fire the trigger based on the *first* instance of modification. To resolve this issue, this modification trigger requires a *modify* variable. When the reservation is modified, the *modify* variable is set.

```
RSVPROFILE[envSetup]
TRIGGER=EType=modify,AType=exec,Action="modify.sh",RearmTime=1:00:00,Requires=approval
1.approval2.approval3.!modify,Sets=modify
RSVPROFILE[envSetup]
TRIGGER=EType=modify,AType=exec,Action="modificationRequest.sh",RearmTime=5:00,Require
s=approval1.approval2.approval3.modify,Unsets=modify
```

The final triggers notify the user of the end of the environment.

```
RSVPROFILE[envSetup]
TRIGGER=EType=end,AType=exec,Action="end.sh",Requires=approval1.approval2.approval3
```

The same trigger is repeated for the *cancel***EType** in case the environment ends unexpectedly.

```
RSVPROFILE[envSetup]
TRIGGER=EType=cancel,AType=exec,Action="end.sh",Requires=approval1.approval2.approval3
```

17.4 Trigger variables

17.4.1 About trigger variables

Trigger variables are pieces of information that pass from trigger to trigger. They allow triggers to fire based on another trigger's behavior, state, and/or output. A variable can be a required condition for a trigger to fire; for instance, a trigger might be set to launch when a reservation starts, but only if it has

received a variable from another trigger indicating that a specific node has started first. Variables give greater flexibility and power to a site administrator who wants to automate certain tasks and system behaviors.

Variables can be used to define under what circumstances the trigger will fire. Many Moab objects have their own variables and each object's variable name space is unique. Triggers can use their own variables or the variables attached to their parent objects. A trigger's variable name space is limited to itself and its parent object. Variables do not have to be unique across all objects.

How-to's

- **Setting and receiving trigger variables** on page 685
- **Externally injecting variables into job triggers** on page 686
- **Exporting variables to parent objects** on page 686
- **Requiring variables from generations of parent objects** on page 687
- **Requesting name space variables** on page 687

References

- **Dependency trigger components** on page 688
- **Internal variables** on page 689

17.4.2 How-to's

17.4.2.1 Setting and receiving trigger variables

Context

Following is an example of how comparative dependencies can be expressed when creating a trigger.

To set and require variables

1. Create a trigger.

```
EType=start,AType=exec,Action="/tmp/trigger1.sh"
```

2. Use the **Sets** attribute to set a variable if the trigger succeeds. You can precede the variable with "!" to indicate that the variable should be set if the trigger fails. You can specify more than one variable by separating them with a period.

```
AType=exec,Action="/tmp/trigger1.sh",EType=start,Sets=!Var1.Var2
```

The trigger sets variable `Var2` when it succeeds and variable `Var1` when it fails.

3. Set up the recipient trigger(s). Use the Requires attribute to receive the variable(s). Note that preceding the variable with "!" means that the variable must not be set in order for the trigger to fire.

```
AType=exec,Action="/tmp/trigger1.sh",EType=start,Sets=!Var1.Var2
AType=exec,Action="/tmp/trigger2.sh",EType=start,Requires=Var1
AType=exec,Action="/tmp/trigger3.sh",EType=start,Requires=Var2
```

The second trigger will launch if `Var1` has been set (the first trigger failed), and the third trigger will launch if `Var2` is set (the first trigger succeeded).

4. Refine the requirement with comparisons.
   a. Use the following format:
   `<varID>[:<type>[:<varVal>]]`
   b. Change *<varID>* to the variable name.
   c. Use any of the comparisons found on the **Trigger variable comparison types** on page 689 page in place of *<type>*:
   d. Set the value that the variable will be compared against.

```
AType=exec,Action="/tmp/trigger2.sh",EType=start,Requires=Var1:eq:45
AType=exec,Action="/tmp/trigger3.sh",EType=start,Requires=Var2:ne:failure1
```

The first trigger fires if `Var1` exists and has a value of 45. The second trigger fires if `Var2` does not have a string value of `failure1`.

17.4.2.2 Externally injecting variables into job triggers

Context

Job triggers are able to see the variables in the job object to which it is attached. This means that, for triggers that are attached to job objects, another method for supplying variables exists. Updating the job object's variables effectively updates the variable for the trigger.

To externally inject variables into job triggers

1. Use the mjobctl -m command to set a variable to attach to a job.

```
> mjobctl -m var=Flag1=TRUE 1664
```

The variable *Flag1* is set. This will be available to any trigger attached to job `1664`.

17.4.2.3 Exporting variables to parent objects

To export variables to parent objects

1. When setting a variable, indicate that the variable is to be exported to the parent object by using a caret (^).

```
AType=exec,Action="/tmp/trigger1.sh",EType=start,Sets=Var1.!^Var2
Atype=exec,Action="/tmp/trigger2.sh",EType=start,Requires=Var1
AType=exec,Action="/tmp/trigger3.sh",EType=start,Requires=Var2
```

`Var2` is exported to the parent object if the trigger fails. It can be used by job and reservation triggers at the same level or by parent objects.

2. Optional: if running a script, you can set a variable as a string to pass up to the parent object.

   a. Set the variable to pass up to the parent object with the caret (^). Use the *exec* **AType** to run a script.

   ```
   AType=exec,Action="/tmp/trigger.sh",EType=start,Sets=^Var1
   ```

   The trigger sets *Var1* when it completes successfully. Because the trigger launches a script, a string value can be set for *Var1*.

   b. Declare the variable's string value on its own line in the trigger stdout.

   ```
   EXITCODE=15
   Var1=linux
   ```

   Var1 has the value of `linux` and is passed up to the parent object. This is useful in workflows in which a trigger may depend on the value given by a previous trigger.

   To return multiple variables, simply print out one per line.

17.4.2.4 Requiring variables from generations of parent objects

Context

By default, triggers look for variables to fulfill dependencies in the object to which they are directly attached. If they are attached to a job object, they will also look in the job group, if defined. However, it is not uncommon for objects to have multiple generations of parent objects. If the desired behavior is to search through all parent objects, do the following task.

To require variables from generations of parent objects

- Set the **Requires** attribute in the trigger to the required variable, preceded by a caret (^).

```
EType=start,AType=exec,Action="/tmp/trigger2.sh",Requires=^Var1
```

The trigger searches through the parent objects in which it resides for the variable Var1.

17.4.2.5 Requesting name space variables

To request a name space variable in a trigger

1. Configure the trigger. If it is attached to a generic system job, verify that it meets all the generic system job trigger requirements.
2. Create an argument list in the `Action` attribute (after the script path and before the closing quotes) and request the desired variable with an asterisk (*) in place of the name space.

```
...Action="$HOME/myTrig.py $*.IPAddr"...
```

Each applicable name space variable is added to the argument list in the format `<varName>=<val>`.

For instance, the example above would cause the script to run the following way:

```
> myTrig.py vc1.IPAddr=/tmp/dir1 vc2.IPAddr=/tmp/dir2 vc4.IPAddr=/tmp/dir3
```

Any other arguments provided here without name spaces will not change.

3. Filter which name spaces are passed down to a job trigger by setting `trigns` when you submit the job. Its value is a comma-delimited list of the desired name spaces.

```
msub -l ... -W x="trigns=vc2,vc4"
```

If the new job is applied to the example in step 2, the script's arguments include *vc2.IPAddr* and *vc4.Addr* and exclude *vc1.IPAddr*. The script runs as follows:

```
> myTrig.py vc2.IPAddr=/tmp/dir1 vc4.IPAddr=/tmp/dir2
```

17.4.3 References

17.4.3.1 Dependency trigger components

Sets	
Possible values	'.' delimited string
Description	Variable values this trigger sets upon success or failure
Usage notes	Preceding the string with an exclamation mark (!) indicates this variable is set upon trigger failure. Preceding the string with a caret (^) indicates this variable is to be exported to the parent object when the trigger completes and satisfies all its set conditions. Used in conjunction with **Requires** on page 689 to create trigger dependencies.

Unsets	
Possible values	'.' delimited string
Description	Variable this trigger destroys upon success or failure.
Usage notes	Preceding the string with an exclamation mark (!) indicates this variable is unset upon trigger failure. Used in conjunction with **Requires** on page 689 to create trigger dependencies.

Requires	
Possible values	'.' delimited string
Description	Variables this trigger requires to be set or not set before it will fire.
Usage notes	Preceding the string with an exclamation mark (!) indicates this variable must not be set. Preceding the string with a caret (^) indicates that the variable may come from a parent object (See **Requiring variables from generations of parent objects** on page 687 for more information.). Used in conjunction with **Sets** on page 688 to create trigger dependencies.

17.4.3.2 Trigger variable comparison types

The following table describes the valid types of comparisons you can use to express the relationship of a trigger variable to its value:

Type	Comparison	Notes
set	is set (exists)	Default
notset	not set (does not exist)	Same as specifying '!' before a variable
eq	equals	
ne	not equal	
gt	greater than	Integer values only
lt	less than	Integer values only
ge	greater than or equal to	Integer values only
le	less than or equal to	Integer values only

17.4.3.3 Internal variables

Several internal variables are available for use in trigger scripts. These can be accessed using `$<varName>`.

Internal Variables	
ETYPE	The type of event that signals that the trigger can fire. ETYPE values include cancel, checkpoint, create, end, fail, hold, migrate, preempt, standing, start, and threshold.
OID	The name of the object to which the trigger was attached
OTYPE	The type of object to which the trigger is attached; can be rsv, job, node, vm, or sched
OWNERMAIL	A variable that is populated only if the trigger's parent object has a user associated with it and that user has an email address associated with it
TIME	The time of the trigger launch in the following format: `Wed Mar 10 12:35:12 2012`
USER	The user (when applicable)

Object-specific internal variables

Job Variables	
MASTERHOST	The primary node for the job
HOSTLIST	The entire host list of the job

Reservation Variables	
HOSTLIST	The entire host list for the reservation
OBJECTXML	The XML representation of an object output is the same that is generated by mdiag -r --xml
OS	The operating system on the first node of the reservation
OWNER	The owner of the reservation

Example 17-2: Internal variable example

```
AType=exec,Action="/tmp/trigger.sh $OID $HOSTLIST",EType=start
```

The object ID (`$OID`) and host list (`$HOSTLIST`) will be passed to `/tmp/trigger.sh` as command line arguments when the trigger executes the script. The script can then process this information as needed.

18.0 Miscellaneous

- **User Feedback Overview** on page 691
- **Enabling High Availability Features** on page 692
- **Malleable Jobs** on page 694
- **Identity Managers** on page 695
- **Generic System Jobs** on page 699
- **Implementing Guaranteed Start Time** on page 702

18.1 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. messages associated with the job
18. host list (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 18-1:

```
FEEDBACKPROGAM  /opt/moab/tools/fb.pl
```

18.2 Enabling High Availability Features

- Moab High Availability Overview
  - Configuring High Availability via a Networked File System
  - Confirming High Availability on a Networked File System
- Other High Availability Configuration

High Availability Overview

High availability allows Moab to run on two different machines: a primary and secondary server. The configuration method to achieve this behavior takes advantage of a networked file system to configure two Moab servers with only one operating at a time.

If you use a shared file system for High Availability and Moab is configured to use a database, Moab must be an ODBC build, not SQLite.

When configured to run on a networked file system — any networked file system that supports file locking is supported — the first Moab server that starts locks a particular file. The second Moab server

waits on that lock and only begins scheduling when it gains control of the lock on the file. This method achieves near instantaneous turnover between failures and eliminates the need for two Moab servers to synchronize information periodically as the two Moab servers access the same database/checkpoint file.

As Moab uses timestamping in the lock file to implement high availability, the clocks on both servers require synchronization; all machines in a cluster must be synchronized to the same time server.

Moab high availability and TORQUE high availability operate independently of each other. If a job is submitted with msub and the primary Moab server is down, msub tries to connect to the fallback Moab server. Once the job is given to TORQUE, if TORQUE can't connect to the primary pbs_server, it tries to connect to the fallback pbs_server. For example:

A job is submitted with msub, but Moab is down on *server01*, so msub contacts Moab running on *server02*.

A job is submitted with msub and Moab hands it off to TORQUE, but pbs_server is down on *server01*, so qsub contacts pbs_server running on *server02*.

When you shut down or restart Moab on both servers, you must run the command twice. A single shutdown (mschedctl -k) or restart (mschedctl -R) command will go to the primary server and kill it, causing the secondary server to fall back and start operating. To kill the secondary server, resubmit the command.

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** attributes of the **SCHEDCFG** parameter. File lock is turned on using the FLAGS=filelockha flag. Specify the lock file with the **HALOCKFILE** attribute. The following example illustrates a possible configuration:

```
SCHEDCFG[Moab]  SERVER=host1:42559
SCHEDCFG[Moab]  FBSERVER=host2
SCHEDCFG[Moab]  FLAGS=filelockha
SCHEDCFG[Moab]  HALOCKFILE=/opt/moab/.moab_lock
```

Use the **HALOCKUPDATETIME** parameter to specify how frequently the primary server updates the timestamp on the lock file. Use the **HALOCKCHECKTIME** parameter to specify how frequently the secondary server checks the timestamp on the lock file.

```
HALOCKCHECKTIME 9
HALOCKUPDATETIME 3
```

*In the preceding example, the secondary server checks the lock file for updates every 9 seconds. The **HALOCKUPDATETIME** parameter is set to 3 seconds, permitting the primary server three opportunities to update the timestamp for each time the secondary server checks the timestamp on the lock file.*

FBSERVER does not take a port number. The primary server's port is used for both the primary server and the fallback server.

Confirming High Availability on a Networked File System

Administrators can run the mdiag -S -v command to view which Moab server is currently scheduling and responding to client requests.

Other High Availability Configuration

Moab has many features to improve the availability of a cluster beyond the ability to automatically relocate to another execution server. The following table describes some of these features.

Feature	Description
JOBACTIONONNODEFAILURE	If a node allocated to an active job fails, it is possible for the job to continue running indefinitely even though the output it produces is of no value. Setting this parameter allows the scheduler to automatically preempt these jobs when a node failure is detected, possibly allowing the job to run elsewhere and also allowing other allocated nodes to be used by other jobs.
SCHEDCFG[] RECOVERYACTION	If a catastrophic failure event occurs (SIGSEGV or SIGILL signal is triggered), Moab can be configured to automatically restart, trap the failure, ignore the failure, or behave in the default manner for the specified signal. These actions are specified using the values *RESTART*, *TRAP*, *IGNORE*, or *DIE*, as in the following example: `SCHEDCFG[bas] MODE=NORMAL RECOVERYACTION=RESTART`

18.3 Malleable Jobs

Malleable jobs are jobs that can be adjusted in terms of resources and duration required, and which allow the scheduler to maximize job responsiveness by selecting a job's resource shape or footprint prior to job execution. Once a job has started, however, its resource footprint is fixed until job completion.

To enable malleable jobs, the underlying resource manager must support dynamic modification of resource requirements prior to execution (i.e., TORQUE) and the jobs must be submitted using the TRL (task request list) resource manager extension string. With the **TRL** attribute specified, Moab will attempt to select a start time and resource footprint to minimize job completion time and maximize overall effective system utilization (i.e., *<AverageJobEfficiency> * <AverageSystemUtilization>*).

Example 18-2:

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
```

18.4 Identity Managers

- Identity Manager Overview
- Basic Configuration
- Importing Credential Fairness Policies
- Identity Manager Data Format
- Identity Manager Conflicts
- Refreshing Identity Manager Data

The Moab identity manager interface can be used to coordinate global and local information regarding users, groups, accounts, and classes associated with compute resources. The identity manager interface may also be used to allow Moab to automatically and dynamically create and modify user accounts and credential attributes according to current workload needs.

Only one identity manager can be configured at a time.

Identity Manager Overview

Moab allows sites extensive flexibility when it comes to defining credential access, attributes, and relationships. In most cases, use of the USERCFG, GROUPCFG, ACCOUNTCFG, CLASSCFG, and QOSCFG parameters is adequate to specify the needed configuration. However, in certain cases such as the following, this approach may not be ideal or even adequate:

- Environments with very large user sets
- Environments with very dynamic credential configurations in terms of fairshare targets, priorities, service access constraints, and credential relationships
- Enterprise environments with fairness policies based on multi-cluster usage

Moab addresses these and similar issues through the use of an identity manager. An identity manager is configured with the IDCFG parameter and allows Moab to exchange information with an external identity management service. As with Moab resource manager interfaces, this service can be a full commercial package designed for this purpose, or something far simpler such as a web service, text file, or database.

Basic Configuration

Configuring an identity manager in basic read-only mode can be accomplished by simply setting the **SERVER** attribute. If Moab is to interact with the identity manager in read/write mode, some additional configuration may be required.

BLOCKCREDLIST	
Format	One or more comma-delimited object types from the following list: *acct*, *group*, or *user*

BLOCKCREDLIST

Details

If specified, Moab will block all jobs associated with credentials not explicitly reported in the most recent identity manager update. If the credential appears on subsequent updates, resource access will be immediately restored.

Jobs will only be blocked if fairshare is enabled. This can be accomplished by setting the FSPOLICY parameter to any value such as in the following example:

```
FSPOLICY DEDICATEDPS
```

Example

```
IDCFG[test01] BLOCKCREDLIST=acct,user,groups
```

Moab will block any jobs associated with accounts, users, or groups not in the most recent identity manager update.

CREATECRED

Format

<*BOOLEAN*> (default is *FALSE*)

Details

Specifies whether Moab should create credentials reported by the identity manager that have not yet been locally discovered or loaded via the resource manager. By default, Moab will only load information for credentials which have been discovered outside of the identity manager.

Example

```
IDCFG[test01] CREATECRED=TRUE
```

Moab will create credentials from `test01` that have not been previously loaded.

REFRESHPERIOD

Format

minute, *hour*, *day*, or *infinity* (default is *infinity*).

Details

If specified, Moab refreshes identity manager information once every specified iteration. If *infinity* is specified, the information is updated only at Moab start up.

Example

```
IDCFG[test01] REFRESHPERIOD=hour
```

Moab queries the identity manager every hour.

RESETCREDLIST	
Format	One or more comma-delimited object types from the following list: *acct*, *group*, or *user*.
Details	If specified, Moab will reset the account access list and fairshare cap and target for all credentials of the specified type(s) regardless of whether they are included in the current info manager report. Moab will then load information for the specified credentials.
Example	`IDCFG[test01] RESETCREDLIST=group` *Moab will reset the account access list and fairshare target for all groups.*

SERVER	
Format	*<URL>*
Details	Specifies the protocol/interface to use to contact the identity manager.
Example	`IDCFG[test01] SERVER=exec://$HOME/example.pl` *Moab will use `example.pl` to communicate with the identity manager.*

UPDATEREFRESHONFAILURE	
Format	*<BOOLEAN>* (default is *FALSE*)
Details	When an IDCFG script fails, it retries almost immediately and continuously until it succeeds. When **UPDATEREFRESHONFAILURE** is set to *TRUE*, a failed script does not attempt to rerun immediately, but instead follows the specified **REFRESHPERIOD** schedule. When set to *TRUE*, **UPDATEREFRESHONFAILURE** updates the script execution timestamp, even if the script does not end successfully.
Example	`IDCFG[info] SERVER=exec:///home/tshaw/test/1447/bad_script.pl REFRESHPERIOD=hour UPDATEREFRESHONFAILURE=TRUE`

Importing Credential Fairness Policies

One common use for an identity manager is to import fairness data from a global external information service. As an example, assume a site needed to coordinate Moab group level fairshare targets with an

allocation database that constrains total allocations available to any given group. To enable this, a configuration like the following might be used:

```
IDCFG[alloc] SERVER=exec://$TOOLSDIR/idquery.pl
...
```

The `tools/idquery.pl` script could be set up to query a local database and report its results to Moab. Each iteration, Moab will then import this information, adjust its internal configuration, and immediately respect the new fairness policies.

Identity Manager Data Format

When an identity manager outputs credential information either through an `exec` or `file` based interface, the data should be organized in the following format:
<CREDTYPE>:<CREDID> <ATTR>=<VALUE>

where

- *<CREDTYPE>* is one of `user`, `group`, `account`, `class`, or `qos`.
- *<CREDID>* is the name of the credential.
- *<ATTR>* is one of `adminlevel`, `alist`, `chargerate`, comment, emailaddress, `fstarget`, `globalfstarget`, `globalfsusage`, maxjob, maxmem, maxnode, maxpe, maxproc, maxps, maxwc, `plist`, `priority`, `qlist`, or `role`. Multi-dimensional policies work here as well.
- *<VALUE>* is the value for the specified attribute.

To clear a comment, set its value to ""; for example: `comment=""`.

Example 18-3:

The following output may be generated by an `exec` based identity manager:

```
group:financial fstarget=16.3 alist=project2
group:marketing fstarget=2.5
group:engineering fstarget=36.7
group:dm fstarget=42.5
user:jason adminlevel=3
account:sales maxnode=128 maxjob=8,16
```

The following example limits user `bob` to 8 `matlab` generic resources.

```
user:bob MAXGRES[matlab]=8
```

To specify unlimited use of generic resources, set the value to `-1`.

Identity Manager Conflicts

When local credential configuration (as specified via `moab.cfg`) conflicts with identity manager configuration, the identity manager value takes precedence and the local values are overwritten.

Refreshing Identity Manager Data

By default, Moab only loads identity manager information once when it is first started up. If the identity manager data is dynamic, then you may want Moab to periodically update its information. To do this, set the **REFRESHPERIOD** attribute of the **IDCFG** parameter. Legal values are documented in the following table:

Value	Description
minute	Update identity information once per minute
hour	Update identity information once per hour
day	Update identity information once per day
infinity	Update identity information only at start-up (default)

Example 18-4:

```
IDCFG[hq] SERVER=exec://$TOOLSDIR/updatepolicy.sh REFRESHPERIOD=hour
```

Job credential feasibility is evaluated at job submission and start time.

Related topics

- Credential Overview
- Usage Limits/Throttling Policies

18.5 Generic System Jobs

Generic system jobs are system jobs with a trigger. They are useful for specifying steps in a workflow.

- Creating a Generic System Job
  - The Trigger
- Workflows Using Job Template Dependencies
  - Inheriting Resources in Workflows

Creating a Generic System Job

Generic system jobs are specified via a job template. The template can be selectable and you must use the **GENERICSYSJOB** attribute to let Moab know that this job template describes a generic system job and to specify a trigger, as shown in the following example:

```
JOBCFG[gen]
GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/genericTrig.py",Timeout=5:00
```

The Trigger

The generic system job's trigger that meets certain criteria. This trigger must have a timeout, an `Atype=Exec`, and the `EType` must equal "start". The timeout of the trigger will be used as the walltime for the job. The trigger will begin when the system job begins and the job will be considered completed when the trigger completes. The job will have the same completion code as the trigger. The walltime on the job template is not applicable in this case since the timeout of the trigger will be the walltime.

If the trigger fails, an error message will be attached to all of the job's parent VCs. You can view this in the `--xml` output of the VC query. The message includes the location of STDIN, STDOUT, and STDERR files. For example:

```
mvcctl -q ALL --xml

<Data>
<vc CREATETIME="1320184350" DESCRIPTION="Moab.1"
  FLAGS="DESTROYOBJECTS,DESTROYWHENEMPTY,HASSTARTED,WORKFLOW"
  JOBS="Moab.1" NAME="vc1" OWNER="user:frank">
<ACL aff="positive" cmp="%=" name="frank" type="USER"></ACL>
<MESSAGES>
<message COUNT="1" CTIME="1320184362"
     DATA="Trigger 10 failed on job Moab.1.setup- STDIN:
/tmp/ByLLl2wv/spool/vm.py.ieWPPS5 STDOUT:
/tmp/ByLLl2wv/spool/vm.py.oDMIXAW STDERR /tmp/ByLLl2wv/spool/vm.py.e2jD5iN"
     EXPIRETIME="1322776362" OWNER="frank" PRIORITY="0"
     TYPE="other" index="0"></message>
</MESSAGES>
<Variables>
<Variable name="VMID">vm1</Variable>
<Variable name="HV">TRUE</Variable>
</Variables>
</vc>
</Data>
```

You can specify other triggers on a generic system job using the **TRIGGER** attribute and delimiting them with semicolons. For example:

```
JOBCFG[gen]  GENERICSYSJOB=<genericSystemJobTriggerSpecs>
JOBCFG[gen]  TRIGGER=<triggerSpecs>;TRIGGER=<triggerSpecs>
```

Workflows Using Job Template Dependencies

To create workflows, use the following format:

```
JOBCFG[gen]  TEMPLATEDEPEND=AFTERANY:otherTemplate
```

This will create a job based on the template `otherTemplate`. The generic job will run after the otherTemplate job has finished. Afterany in the example means after all other jobs have completed, regardless of success.

Inheriting Resources in Workflows

The *INHERITRES* flag can be used to cause the same resources in one step of a workflow to be passed to the next step:

```
JOBCFG[gen]  TEMPLATEDEPEND=AFTERANY:otherTemplate
JOBCFG[otherTemplate]  INHERITRES=TRUE
```

This example forces the job based on `otherTemplate` to have the same resource requirements as its parent. When the `otherTemplate` job is finished, the INHERITRES flag will cause the parent to run on the same resources as the child.

The job that finishes first will pass its allocation up.

Any variables on the original job will be passed to the other jobs in the workflow. Variables can be added by other jobs in the workflow via the sets attribute in the generic system job's trigger. Other triggers must then request that variable name in the command line options.

You will need to set the carat (^) in order for the variable to be sent up to the job group.

If you set the variable, you need to set it in the STDOUT of the trigger script. See the example below:

```
JOBCFG[W1] GENERICSYSJOB=...,action='$HOME/W1.py $ipaddress' TEMPLATEDEPEND=AFTER:W2
JOBCFG[W2] TRIGGER=...,action='$HOME/W2.py',sets=^ipaddress
```

If a variable value is not set in STDOUT, it will be set to TRUE.

To set the variable to a specific value, the `W2.py` script must set the value in its STDOUT:

```
print "ipaddress=10.10.10.1" #This will be parsed by Moab and set as the value of the
"ipaddress" variable
```

Example 18-5:

To create a VM with a workflow using job template dependencies and generic system jobs, use the following format:

```
#The job template that is "gate" to the workflow
JOBCFG[CreateVMWithSoftware] TEMPLATEDEPEND=AFTEROK:InstallSoftware SELECT=TRUE

JOBCFG[InstallSoftware]
GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/setupSoftware.py
$IPAddr",Timeout=30:00
JOBCFG[InstallSoftware]  INHERITRES=TRUE
JOBCFG[InstallSoftware]  TEMPLATEDEPEND=AFTEROK:CreateVM

JOBCFG[CreateVM]  GENERICSYSJOB=EType=start,AType=exec,Action=$HOME/installVM.py
$HOSTLIST",Timeout=1:00:00,sets=^IPAddr
JOBCFG[CreateVM]  INHERITRES=TRUE
```

The user will then submit the job requesting what they need in the VM:

```
msub -l walltime=2:00:00,template=CreateVMWithSoftware,nodes=1:ppn=4,mem=1024
ActualWorkload.py
```

The job will have the `CreateVMWithSoftware` template applied to it and will create the `InstallSoftware` job. The `InstallSoftware` job, because of INHERITRES, will have the same resource request (4 procs, 1GB of memory). This job then has its template applied to it which will do the same thing in creating the `CreateVM` job. The `CreateVM job` will then run, the trigger script will return the IP address of the new VM and pass its allocation up to the `InstallSoftware` job. The `InstallSoftware` job will use the IPAddr variable to find the VM and install the software. It will then return its resources up to the parent job, which will run the actual workload.

18.6 Implementing Guaranteed Start Time

Guaranteed start time allows you to specify when your environment will start. There are limitations on the number of jobs that can be combined for a guaranteed start time. Also, each job must be a single requirement job. The process of guaranteeing a start time requires that jobs be attached to a virtual container (VC). Only jobs or workflows belonging directly to the VC will be scheduled.

To guarantee start time, do the following:

1. Create a virtual container (VC).

```
> mvcctl -c
```

2. Set the holdjobs flag on the VC to prevent jobs from immediately running.

```
> mvcctl -m flags+=holdjobs <VCNAME>
```

3. Submit the job(s).

```
> msub -W x=vc=<VCNAME> . . .
```

4. (Optional Step) Specify a start time as epoch or relative (+) via the reqstarttime attribute.

```
> mvcctl -m reqstarttime=<epoch | +{mm:ss}> <VCNAME>
```

5. Schedule the VC.

```
> mvcctl -x schedulevc <VCNAME>
```

There is a default limit of five jobs that can be combined for guaranteed start time.

19.0 Database Configuration

Moab supports connecting to a database via native SQLite3, and it can also connect to other databases using the ODBC driver. Oracle support is forthcoming. These optional external databases store some additional information that the MongoDB database does not and allow you to query them directly using SQL. These databases are slower, however, and only SQLite3, which does not allow external queries, is supported.

The SQLite3 connection is for storing statistics. Consider reviewing the SQLite web page Appropriate Uses for SQLite for information regarding the suitability of using SQLite3 on your system.

While the ODBC connection is useful for storing statistics, it also stores events, nodes, and jobs. You can further configure Moab to store checkpoint information to a database rather than to the flat text file (`.moab.ck`) if you set the CHECKPOINTWITHDATABASE parameter to *TRUE*.

Connecting to an external database makes Moab more searchable, allowing you to run queries for statistics and events rather than using regular expressions to drawn the information from the Moab flat files.

- **SQLite3** on page 703
- **Connecting to a MySQL Database with an ODBC Driver** on page 704
- **Connecting to a PostgreSQL Database with an ODBC Driver** on page 707
- **Migrating Your Database to Newer Versions of Moab** on page 709
- **Importing Statistics from stats/DAY.* to the Moab Database** on page 714

Moab must use an ODBC-compliant database to report statistics with Viewpoint reports.

19.1 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.

19.2 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 the MySQL ODBC driver. This document assumes the necessary MySQL and ODBC drivers have already been installed and configured.

To set up and configure Moab to connect to a MySQL database using the MySQL ODBC driver, do the following:

This solution has been tested and works with these versions:

- libmyodbc - 5.1.5
- unixodbc - 2.3.2
- MySQL 5.1

For a Debian-based system, unixodbc-dev is required, but it might not be required 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`):

```
USEDATABASE             ODBC
# Turn on stat profiling
USERCFG[DEFAULT]        ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]       ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]         ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]       ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]     ENABLEPROFILING=TRUE
```

```
NODECFG[DEFAULT]        ENABLEPROFILING=TRUE
```

2. Create the database in MySQL using the MySQL database dump contained in the `moab-db.sql` file. This file is located in the `contrib/sql` directory in the root of the binaries.

   - Run the following command:

     ```
     mysql -u root -p < moab-db-mysql-create.sql
     ```

3. Configure the MySQL and ODBC driver. The `/etc/odbcinst.ini` file should contain content similar to what follows:

   ```
   [MySQL]
   Description = ODBC for MySQL
   Driver = /usr/lib/odbc/libmyodbc.so
   ```

   Run `updatedb && locate libmyodbc` to find the MySQL ODBC client driver. You could also check the `libmyodbc` package that was installed.

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 `<username>`, password `<password>`, and to connect to MySQL running on the localhost on port `3306`. ODBC uses this information to connect to the database called `Moab`.

5. Test the ODBC to MySQL connection by running the isql command, which reads the `/etc/odbc.ini` file:

   ```
   $ isql -v ODBC
   +---------------------------------------+
   | Connected! |
   | |
   | sql-statement |
   | help [tablename] |
   | quit |
   | |
   +---------------------------------------+
   ```

```
SQL> show tables;
+-----------------------------------------------------------------+
| Tables_in_Moab |
+-----------------------------------------------------------------+
| EventType |
| Events |
| GeneralStats |
| GenericMetrics |
| Moab |
| NodeStats |
| NodeStatsGenericResources |
| ObjectType |
| mcheckpoint |
+-----------------------------------------------------------------+
SQLRowCount returns 10
10 rows fetched
SQL>
```

If you encounter any errors using the isql command, there was a problem setting up the ODBC to MySQL connection. Try the following debugging steps to resolve the issue:

a. The `odbcinst.ini` and `odbc.ini` files are usually assumed to be located in `/etc`, but that is not always true. Use the `odbcinst -j` command to determine the assumed location of the files in your configuration.

```
[root#] odbcinst -j
unixODBC 2.2.12
DRIVERS............: /etc/unixODBC/odbcinst.ini
SYSTEM DATA SOURCES: /etc/unixODBC/odbc.ini
USER DATA SOURCES..: /home/adaptive/.odbc.ini
```

b. Because `odbcinst.ini` and `odbc.ini` are expected in `/etc/unixODBC`, not `/etc`, move them from `/etc` to `/etc/unixODBC`.

c. Use the `strace` command to determine where `isql` expects the `odbc.ini` and `odbcinst.ini` files.

```
$ strace isql -v ODBC noting the location in which isql expects the odbc.ini and
odbcinst.ini files.
```

6. With the ODBC driver configured, the database created, and Moab configured to use the database, start Moab for it to begin storing information in the created database.

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

Related topics

- Importing Statistics to the Moab Database

19.3 Connecting to a PostgreSQL Database with an ODBC Driver

This documentation shows how to set up and configure Moab to connect to a PostgreSQL database using the ODBC driver. This document assumes the necessary ODBC drivers have already been installed and configured.

Occasionally vacuuming your PostgresSQL database could improve Moab performance. See the PostgresSQL documentation for information on how to vacuum your database.

To set up and configure Moab to connect to a PostgreSQL database using the ODBC driver, do the following:

This solution has been tested and works with these file versions:

- odbc-postgresql - 1:08.03.0200-1.2
- unixodbc - 2.2.14

For a Debian-based system, unixodbc-dev is required, but it might not be required for Red Hat flavors (such as CentOS and RHEL).

1. Configure the PostgreSQL and ODBC driver. The `/etc/odbcinst.ini` file should contain content similar to what follows:

```
[PostgreSQL]
Description = PostgreSQL ODBC driver
Driver = /usr/lib/odbc/psqlodbca.so
Setup = /usr/lib/odbc/libodbcpsqlS.so
Debug = 0
CommLog = 1
UsageCount = 2
```

   Run `updatedb && locate libodbcpsql` to find the PostgreSQL ODBC client driver. You could also check the `libodbcpsql` package that was installed.

2. Configure Moab to use the PostgreSQL ODBC driver. Moab uses an ODBC datastore file to connect to PostgreSQL using ODBC. This file must be located in the Moab home directory (`/opt/moab` by default) and be named `dsninfo.dsn`, which is used by Moab. If the following content, which follows the standard ODBC driver file syntax, is not already included in the `/etc/odbc.ini` file, make sure that you include it. Also, include the same content in the `dsninfo.dsn` file.

```
[ODBC]
Driver = PostgreSQL
Description = PostgreSQL Data Source
Servername = localhost
Port = 5432
```

```
Protocol = 8.4
UserName = postgres
Password = moab
Database = Moab
```

The user should have read/write privileges on the Moab database.

The preceding example file tells ODBC to use the PostgreSQL driver, `postgres` user, `moab` password, and to connect to PostgreSQL running on the localhost on port 5432. ODBC uses this information and connects to the database called `Moab`.

3. Test the ODBC to PostgreSQL connection by running the isql command, which reads the `/etc/odbc.ini` file. If connected, you should be able to run the help command.

   If you encounter any errors using the isql command, there was a problem setting up the ODBC to MySQL connection. Try the following debugging steps to resolve the issue:

   a. The `odbcinst.ini` and `odbc.ini` files are usually assumed to be located in `/etc`, but that is not always true. Use the `odbcinst -j` command to determine the assumed location of the files in your configuration.

   ```
   [root#] odbcinst -j
   unixODBC 2.2.12
   DRIVERS............: /etc/unixODBC/odbcinst.ini
   SYSTEM DATA SOURCES: /etc/unixODBC/odbc.ini
   USER DATA SOURCES..: /home/adaptive/.odbc.ini
   ```

   b. Because `odbcinst.ini` and `odbc.ini` are expected in `/etc/unixODBC`, not `/etc`, move them from `/etc` to `/etc/unixODBC`.

   c. Use the `strace` command to determine where `isql` expects the `odbc.ini` and `odbcinst.ini` files.

   ```
   $ strace isql -v ODBC noting the location in which isql expects the odbc.ini and
   odbcinst.ini files.
   ```

4. Create the database in PostgreSQL using the `moab-db-postgresql.sh` setup script contained in the `contrib/sql` directory at the root of the binary.

   - Run the script and provide the DB username that will attach to the Moab database (you must supply a DB username or the script will exit). The default admin user is `postgres`, but you can make a new user at this time:

   ```
   > ./moab-db-postgresql.sh postgres
   Create db user "postgres" in postgreSQL? (y/n)>
   ```

   - The script asks if you want to create the DB user you specified in postgreSQL. If the DB user already exists, answer 'n'. Otherwise, the DB user is created and it asks for the new user's password.
   - The script then creates the database "Moab".

- Finally, as the DB user you provided, the script imports the DB schema from `moab-db-postgresql-create.sql` into the Moab database.

5. Download and install the ODBC version of Moab. Install and configure Moab as normal but add the following in the Moab configuration file (`moab.cfg`):

```
USEDATABASE             ODBC
# Turn on stat profiling
USERCFG[DEFAULT]        ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]       ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]         ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]       ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]     ENABLEPROFILING=TRUE
NODECFG[DEFAULT]        ENABLEPROFILING=TRUE
```

6. With the ODBC driver configured, the database created, and Moab configured to use the database, start Moab for it to begin storing information in the created database.

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

Related topics

- Importing Statistics to the Moab Database

19.4 Migrating Your Database to Newer Versions of Moab

Sometimes when upgrading from an older version of Moab to a newer version, you must update your database schema. If the schema Moab expects to operate against is different from the actual schema of the database Moab is connected to, Moab might not be able to use the database properly and data might be lost.

When upgrading the Moab database schema from an old version, you must perform each version upgrade in order. You cannot skip versions. For example, to migrate from version 6.1 to version 7.2, you must follow the steps in **Migrating from Moab 6.1 to Moab 7.0** on page 710, **Migrating from Moab 7.0 to Moab 7.1** on page 710, **Migrating from Moab 7.1 to Moab 7.2** on page 710, then **Migrating from Moab 7.2.x (where "x" is 5 or lower) to Moab 7.2.6** on page 709.

Migrating from Moab 7.2.x (where "x" is 5 or lower) to Moab 7.2.6

In Moab 7.2.6, several columns were extended and the primary key on the Triggers table changed. To upgrade your database with these changes, use the `moab-db-<database>-upgrade7_2_6.sql` file located in the `contrib/sql` directory in the root of the binaries. For example, to migrate your MySQL database from the 7.2.x (pre-7.2.6) schema to the 7.2.6 schema, run the following:

```
mysql -u root -p < moab-db-mysql-upgrade7_2_6.sql
```

Similar migration scripts exist for Oracle and PostgreSQL.

Migrating from Moab 7.1 to Moab 7.2

In Moab 7.2, several events in the event table related to the Accounting Manager were renamed. To upgrade your database with these changes, use the `moab-db-<database>-upgrade7_2.sql` file located in the `contrib/sql` directory in the root of the binaries. For example, to migrate your MySQL database from the 7.1 schema to the 7.2 schema, run the following:

```
mysql -u root -p < moab-db-mysql-upgrade7_2.sql
```

Similar migration scripts exist for Oracle and PostgreSQL.

Migrating from Moab 7.0 to Moab 7.1

In Moab 7.1, Offset was renamed TriggerOffset in the Triggers table. To upgrade your database with these changes, use the `moab-db-<database>-upgrade7_1.sql` file located in the root of the binaries. For example, to migrate your MySQL database from the 7.0 schema to the 7.1 schema, run the following:

```
mysql -u root -p < moab-db-mysql-upgrade7_1.sql
```

Similar migration scripts exist for Oracle and PostgreSQL.

Migrating from Moab 6.1 to Moab 7.0

In Moab 7.0, the Moab table has been removed from the database, and a MoabInfo and JobHistory table have been added to it. To upgrade your database with these changes, use the `moab-db-mysql-upgrade6_1.sql` file located in the `contrib/sql` directory in the root of the binaries. This is done by running the following command:

```
mysql -u root -p < moab-db-mysql-upgrade6_1.sql
```

Your MySQL database is updated for Moab 7.0.

Migrating from Moab 6.0 to Moab 6.1

An Events table has been added to the database in Moab 6.1. Update the `contrib/sql/moab-db.sql` file with the following table:

```
CREATE TABLE Events (
ID INTEGER,
ObjectType INTEGER,
EventType INTEGER,
EventTime INTEGER UNSIGNED,
ObjectName VARCHAR(64),
Name VARCHAR(64),
Description TEXT,
PRIMARY KEY (ID)
);
```

Use the mdiag -e `--xml` command in the following format to query the events table.

```
mdiag -e [-w <starttime>|<endtime>|<eventtypes>|<oidlist>|<eidlist>|<objectlist>] --
xml
```

The table is then displayed with all specified events configured with the RECORDEVENTLIST parameter.

If the command could return a large of data, redirect the output. `mdiag -e --xml > outputfile`

Migrating from Moab 5.4 to Moab 6.0

The ODBC database schema has been updated for Moab 6.0. When updating Moab to version 6.0, the changes below must be applied to the database for database functionality to work. Below are the SQL statements required to update the schema for Moab 6.0.

These changes are only necessary for an ODBC database. An SQLite database does not require an update.

```
ALTER TABLE Events ADD COLUMN Name VARCHAR(64);
ALTER TABLE Events MODIFY Description TEXT;

CREATE TABLE Nodes (
  ID VARCHAR(64),
  State VARCHAR(64),
  OperatingSystem VARCHAR(64),
  ConfiguredProcessors INTEGER UNSIGNED,
  AvailableProcessors INTEGER UNSIGNED,
  ConfiguredMemory INTEGER UNSIGNED,
  AvailableMemory INTEGER UNSIGNED,
  Architecture VARCHAR(64),
  AvailGres VARCHAR(64),
  ConfigGres VARCHAR(64),
  AvailClasses VARCHAR(64),
  ConfigClasses VARCHAR(64),
  ChargeRate DOUBLE,
  DynamicPriority DOUBLE,
  EnableProfiling INTEGER UNSIGNED,
  Features VARCHAR(64),
  GMetric VARCHAR(64),
  HopCount INTEGER UNSIGNED,
  HypervisorType VARCHAR(64),
  IsDeleted INTEGER UNSIGNED,
  IsDynamic INTEGER UNSIGNED,
  JobList VARCHAR(64),
  LastUpdateTime INTEGER UNSIGNED,
  LoadAvg DOUBLE,
  MaxLoad DOUBLE,
  MaxJob INTEGER UNSIGNED,
  MaxJobPerUser INTEGER UNSIGNED,
  MaxProc INTEGER UNSIGNED,
  MaxProcPerUser INTEGER UNSIGNED,
  OldMessages VARCHAR(64),
  NetworkAddress VARCHAR(64),
  NodeSubstate VARCHAR(64),
  Operations VARCHAR(64),
  OSList VARCHAR(64),
  Owner VARCHAR(64),
  ResOvercommitFactor VARCHAR(64),
  Partition VARCHAR(64),
  PowerIsEnabled INTEGER UNSIGNED,
  PowerPolicy VARCHAR(64),
  PowerSelectState VARCHAR(64),
  PowerState VARCHAR(64),
```

```
  Priority INTEGER UNSIGNED,
  PriorityFunction VARCHAR(64),
  ProcessorSpeed INTEGER UNSIGNED,
  ProvisioningData VARCHAR(64),
  AvailableDisk INTEGER UNSIGNED,
  AvailableSwap INTEGER UNSIGNED,
  ConfiguredDisk INTEGER UNSIGNED,
  ConfiguredSwap INTEGER UNSIGNED,
  ReservationCount INTEGER UNSIGNED,
  ReservationList VARCHAR(64),
  ResourceManagerList VARCHAR(64),
  Size INTEGER UNSIGNED,
  Speed DOUBLE,
  SpeedWeight DOUBLE,
  TotalNodeActiveTime INTEGER UNSIGNED,
  LastModifyTime INTEGER UNSIGNED,
  TotalTimeTracked INTEGER UNSIGNED,
  TotalNodeUpTime INTEGER UNSIGNED,
  TaskCount INTEGER UNSIGNED,
  VMOSList VARCHAR(64),
  PRIMARY KEY (ID)
);


CREATE TABLE Jobs (
  ID VARCHAR(64),
  SourceRMJobID VARCHAR(64),
  DestinationRMJobID VARCHAR(64),
  GridJobID VARCHAR(64),
  AName VARCHAR(64),
  User VARCHAR(64),
  Account VARCHAR(64),
  Class VARCHAR(64),
  QOS VARCHAR(64),
  OwnerGroup VARCHAR(64),
  JobGroup VARCHAR(64),
  State VARCHAR(64),
  EState VARCHAR(64),
  SubState VARCHAR(64),
  UserPriority INTEGER UNSIGNED,
  SystemPriority INTEGER UNSIGNED,
  CurrentStartPriority INTEGER UNSIGNED,
  RunPriority INTEGER UNSIGNED,
  PerPartitionPriority TEXT,
  SubmitTime INTEGER UNSIGNED,
  QueueTime INTEGER UNSIGNED,
  StartTime INTEGER UNSIGNED,
  CompletionTime INTEGER UNSIGNED,
  CompletionCode INTEGER,
  UsedWalltime INTEGER UNSIGNED,
  RequestedMinWalltime INTEGER UNSIGNED,
  RequestedMaxWalltime INTEGER UNSIGNED,
  CPULimit INTEGER UNSIGNED,
  SuspendTime INTEGER UNSIGNED,
  HoldTime INTEGER UNSIGNED,
  ProcessorCount INTEGER,
  RequestedNodes INTEGER,
  ActivePartition VARCHAR(64),
  SpecPAL VARCHAR(64),
  DestinationRM VARCHAR(64),
  SourceRM VARCHAR(64),
```

```
  Flags TEXT,
  MinPreemptTime INTEGER UNSIGNED,
  Dependencies TEXT,
  RequestedHostList TEXT,
  ExcludedHostList TEXT,
  MasterHostName VARCHAR(64),
  GenericAttributes TEXT,
  Holds TEXT,
  Cost DOUBLE,
  Description TEXT,
  Messages TEXT,
  NotificationAddress TEXT,
  StartCount INTEGER UNSIGNED,
  BypassCount INTEGER UNSIGNED,
  CommandFile TEXT,
  Arguments TEXT,
  RMSubmitLanguage TEXT,
  StdIn TEXT,
  StdOut TEXT,
  StdErr TEXT,
  RMOutput TEXT,
  RMError TEXT,
  InitialWorkingDirectory TEXT,
  UMask INTEGER UNSIGNED,
  RsvStartTime INTEGER UNSIGNED,
  BlockReason TEXT,
  BlockMsg TEXT,
  PSDedicated DOUBLE,
  PSUtilized DOUBLE,
  PRIMARY KEY (ID)
);

CREATE TABLE Requests (
  JobID VARCHAR(64),
  RIndex INTEGER UNSIGNED,
  AllocNodeList VARCHAR(1024),
  AllocPartition VARCHAR(64),
  PartitionIndex INTEGER UNSIGNED,
  NodeAccessPolicy VARCHAR(64),
  PreferredFeatures TEXT,
  RequestedApp VARCHAR(64),
  RequestedArch VARCHAR(64),
  ReqOS VARCHAR(64),
  ReqNodeSet VARCHAR(64),
  ReqPartition VARCHAR(64),
  MinNodeCount INTEGER UNSIGNED,
  MinTaskCount INTEGER UNSIGNED,
  TaskCount INTEGER UNSIGNED,
  TasksPerNode INTEGER UNSIGNED,
  DiskPerTask INTEGER UNSIGNED,
  MemPerTask INTEGER UNSIGNED,
  ProcsPerTask INTEGER UNSIGNED,
  SwapPerTask INTEGER UNSIGNED,
  NodeDisk INTEGER UNSIGNED,
  NodeFeatures TEXT,
  NodeMemory INTEGER UNSIGNED,
  NodeSwap INTEGER UNSIGNED,
  NodeProcs INTEGER UNSIGNED,
  GenericResources TEXT,
  ConfiguredGenericResources TEXT,
  PRIMARY KEY (JobID,RIndex)
);
```

```
INSERT INTO ObjectType (Name,ID) VALUES ("Rsv",13);
INSERT INTO ObjectType (Name,ID) VALUES ("RM",14);
INSERT INTO ObjectType (Name,ID) VALUES ("Sched",15);
INSERT INTO ObjectType (Name,ID) VALUES ("SRsv",16);
INSERT INTO ObjectType (Name,ID) VALUES ("Sys",17);
INSERT INTO ObjectType (Name,ID) VALUES ("TNode",18);
INSERT INTO ObjectType (Name,ID) VALUES ("Trig",19);
INSERT INTO ObjectType (Name,ID) VALUES ("User",20);
INSERT INTO ObjectType (Name,ID) VALUES ("CJob",23);
INSERT INTO ObjectType (Name,ID) VALUES ("GRes",30);
INSERT INTO ObjectType (Name,ID) VALUES ("Gmetric",31);
INSERT INTO ObjectType (Name,ID) VALUES ("Stats",39);
INSERT INTO ObjectType (Name,ID) VALUES ("TJob",42);
INSERT INTO ObjectType (Name,ID) VALUES ("Paction",43);
INSERT INTO ObjectType (Name,ID) VALUES ("VM",45);
INSERT INTO ObjectType (Name,ID) VALUES ("JGroup",48);

INSERT INTO EventType (Name,ID) VALUES ("TRIGTHRESHOLD",41);
INSERT INTO EventType (Name,ID) VALUES ("VMCREATE",42);
INSERT INTO EventType (Name,ID) VALUES ("VMDESTROY",43);
INSERT INTO EventType (Name,ID) VALUES ("VMMIGRATE",44);
INSERT INTO EventType (Name,ID) VALUES ("VMPOWERON",45);
INSERT INTO EventType (Name,ID) VALUES ("VMPOWEROFF",46);
INSERT INTO EventType (Name,ID) VALUES ("NODEMODIFY",47);
INSERT INTO EventType (Name,ID) VALUES ("NODEPOWEROFF",48);
INSERT INTO EventType (Name,ID) VALUES ("NODEPOWERON",49);
INSERT INTO EventType (Name,ID) VALUES ("NODEPROVISION",50);
INSERT INTO EventType (Name,ID) VALUES ("ALLSCHEDCOMMAND",51);
INSERT INTO EventType (Name,ID) VALUES ("AMCANCEL",52);
INSERT INTO EventType (Name,ID) VALUES ("AMDEBIT",53);
INSERT INTO EventType (Name,ID) VALUES ("AMQUOTE",54);
INSERT INTO EventType (Name,ID) VALUES ("AMRESERVE",55);
INSERT INTO EventType (Name,ID) VALUES ("RMPOLLEND",56);
INSERT INTO EventType (Name,ID) VALUES ("RMPOLLSTART",57);
INSERT INTO EventType (Name,ID) VALUES ("SCHEDCYCLEEND",58);
INSERT INTO EventType (Name,ID) VALUES ("SCHEDCYCLESTART",59);
INSERT INTO EventType (Name,ID) VALUES ("JOBCHECKPOINT",60);

ALTER TABLE GeneralStats ADD COLUMN TotalConfiguredProcCount INTEGER;
```

19.5 Importing Statistics from stats/DAY.* to the Moab Database

The `contrib/stat_converter` folder contains the files to build `mstat_converter`, an executable that reads file-based statistics in a Moab stats directory and dumps them into a database. It also reads the Moab checkpoint file (`.moab.ck`) and dumps that to the database as well. It uses the `$MOABHOMEDIR/moab.cfg` file to connect to the appropriate database and reads the statistics files from `$MOABHOMEDIR/stats`.

To run, execute the program `mstat_converter` with no arguments.

The statistics converter program does not clear the database before converting. However, if there are statistics in the database and the statistics files from the same period, the converter overwrites the database information with the information from the statistics files.

20.0 Accelerators

Moab can integrate with the TORQUE resource manager to discover, report, schedule, and submit workload to various accelerator architectures (such as NVIDIA GPUs or Intel® Xeon Phi™ co-processor architecture) for parallel processing. See the topics below for specific information.

- Scheduling GPUs
  - Using GPUs with NUMA
  - NVIDIA GPUs
  - GPU Metrics
- Configuring Intel® Xeon Phi™ Co-processor Architecture
  - Intel® Xeon Phi™ Co-processor Metrics

20.1 Scheduling GPUs

In TORQUE 2.5.4 and later, users can request GPUs on a node at job submission by specifying a nodes resource request, using the qsub -l option. The number of GPUs a node has must be specified in the nodes file. The GPU is then reported in the output of pbsnodes:

```
napali
state = free
np = 2
ntype = cluster
status = rectime=1288888871,varattr=,jobs=,state=free,netload=1606207294,gres=tom:!
/home/dbeer/dev/scripts/dynamic_
resc.sh,loadave=0.10,ncpus=2,physmem=3091140kb,availmem=32788032348kb,
totmem=34653576492kb,idletime=4983,nusers=3,nsessions=14,sessions=3136 1805 2380 2428
1161 3174 3184
3191 3209 3228 3272 3333 20560 32371,uname=Linux napali 2.6.32-25-generic #45-Ubuntu
SMP Sat Oct 16 19:52:42
UTC 2010 x86_64,opsys=linux
mom_service_port = 15002
mom_manager_port = 15003
gpus = 1
```

The $PBS_GPUFILE has been created to include GPU awareness. The GPU appears as a separate line in $PBS_GPUFILE and follows this syntax:

```
<hostname>-gpu<index>
```

If a job were submitted to run on a server called "napali" (the submit command would look something like: qsub test.sh -l nodes=1:ppn=2:gpus=1), the $PBS_GPUFILE would contain:

```
napali-gpu0
```

It is left up to the job's owner to make sure that the job executes properly on the GPU. By default, TORQUE treats GPUs exactly the same as ppn (which corresponds to CPUs).

Related topics

- Using GPUs with NUMA
- NVIDIA GPUs

20.2 Using GPUs with NUMA

The pbs_server requires awareness of how the MOM is reporting nodes since there is only one MOM daemon and multiple MOM nodes. Configure the `server_priv/nodes` file with the `num_node_boards` and `numa_gpu_node_str` attributes. The attribute `num_node_boards` tells pbs_server how many NUMA nodes are reported by the MOM. If each NUMA node has the same number of GPUs, add the total number of GPUs to the nodes file. Following is an example of how to configure the nodes file with `num_node_boards`:

```
numahost gpus=12 num_node_boards=6
```

This line in the nodes file tells pbs_server there is a host named numahost and that it has 12 GPUs and 6 nodes. The pbs_server divides the value of GPUs (12) by the value for `num_node_boards` (6) and determines there are 2 GPUs per NUMA node.

In this example, the NUMA system is uniform in its configuration of GPUs per node board, but a system does not have to be configured with the same number of GPUs per node board. For systems with non-uniform GPU distributions, use the attribute `numa_gpu_node_str` to let pbs_server know where GPUs are located in the cluster.

If there are equal numbers of GPUs on each NUMA node, you can specify them with a string. For example, if there are 3 NUMA nodes and the first has 0 GPUs, the second has 3, and the third has 5, you would add this to the nodes file entry:

```
numa_gpu_node_str=0,3,5
```

In this configuration, pbs_server knows it has three MOM nodes and the nodes have 0, 3s, and 5 GPUs respectively. Note that the attribute gpus is not used. The gpus attribute is ignored because the number of GPUs per node is specifically given.

In TORQUE 3.0.2 or later, qsub supports the mapping of `-l gpus=X` to `-l gres=gpus:X`. This allows users who are using NUMA systems to make requests such as `-l ncpus=20,gpus=5` (or `-l ncpus=20:gpus=5`)indicating they are not concerned with the GPUs in relation to the NUMA nodes they request; they only want a total of 20 cores and 5 GPUs.

Related topics

- Scheduling GPUs
- NVIDIA GPUs

20.3 NVIDIA GPUs

The pbs_mom file can now query for GPU hardware information and report status to the pbs_server. gpustatus will appear in pbsnodes output. New commands allow for setting GPU modes and for resetting GPU ECC error counts.

This feature is only available in TORQUE 2.5.6, 3.0.2, and later.

This document assumes that you have installed the NVIDIA CUDA ToolKit and the NVIDIA development drivers on a compute node with an NVIDIA GPU. (Both can be downloaded from http://developer.nvidia.com/category/zone/cuda-zone).

You will want to download the latest version if you run into problems compiling.

If the pbs_server does not have GPUs, it only needs to be configured with --enable-nvidia-gpus. All other systems that have NVIDIA GPUs will need:

- --enable-nvidia-gpus
- --with-nvml-include=DIR (include path for nvml.h)

  nvml.h is only found in the NVIDIA CUDA ToolKit.

- --with-nvml-lib=DIR (*lib path for libnvidia-ml)

Systems that have NVIDIA GPUs require the following:

Server

```
./configure --with-debug --enable-nvidia-gpus
```

Compute nodes (with NVIDIA GPUs)

```
./configure --with-debug --enable-nvidia-gpus --with-nvml-lib=/usr/lib64 --with-nvml-
include=/cuda/NVML
```

If all of the compute nodes have the same hardware and software configuration, you can choose to compile on one compute node and then run make packages.

```
> make packages
Building ./torque-package-clients-linux-x86_64.sh ...
Building ./torque-package-mom-linux-x86_64.sh ...
Building ./torque-package-server-linux-x86_64.sh ...
Building ./torque-package-gui-linux-x86_64.sh ...
Building ./torque-package-devel-linux-x86_64.sh ...
Done.
```

The package files are self-extracting packages that can be copied and executed on your production machines. (Use --help for options.)

When updating, it is good practice to stop the pbs_server and make a backup of the TORQUE home directory. You will also want to back up the output of qmgr -c 'p s'. The update will only overwrite the binaries.

If you move GPU cards to different slots, you must restart pbs_server in order for TORQUE to recognize the drivers as the same ones in different locations rather than 2 new, additional drivers.

For further details, see these topics:

- **TORQUE configuration** on page 718
- **GPU modes for NVIDIA 260.x driver** on page 718
- **GPU Modes for NVIDIA 270.x driver** on page 718
- **gpu_status** on page 719
- **New NVIDIA GPU support** on page 719

TORQUE configuration

There are three configuration (./configure) options available for use with Nvidia GPGPUs:

- --enable-nvidia-gpus
- --with-nvml-lib=DIR
- --with-nvml-include=DIR

`--enable-nvidia-gpus` is used to enable the new features for the Nvidia GPGPUs. By default, the pbs_moms use the `nvidia_smi` command to interface with the Nvidia GPUs.

```
./configure --enable-nvidia-gpus
```

To use the NVML (NVIDIA Management Library) API instead of nvidia-smi, configure TORQUE using `--with-nvml-lib=DIR` and `--with-nvml-include=DIR`. These commands specify the location of the `libnvidia-ml` library and the location of the `nvml.h` include file.

```
./configure -with-nvml-lib=/usr/lib
--with-nvml-include=/usr/local/cuda/Tools/NVML
server_priv/nodes:
node001 gpus=1
node002 gpus=4
…
pbsnodes -a
node001
    …
    gpus = 1
...
```

By default, when TORQUE is configured with `--enable-nvidia-gpus` the `$TORQUE_HOME/nodes` file is automatically updated with the correct GPU count for each MOM node.

GPU modes for NVIDIA 260.x driver

- 0 – Default - Shared mode available for multiple processes
- 1 – Exclusive - Only one COMPUTE thread is allowed to run on the GPU
- 2 – Prohibited - No COMPUTE contexts are allowed to run on the GPU

GPU Modes for NVIDIA 270.x driver

- 0 – Default - Shared mode available for multiple processes
- 1 – Exclusive Thread - Only one COMPUTE thread is allowed to run on the GPU (v260 exclusive)

- 2 – Prohibited - No COMPUTE contexts are allowed to run on the GPU
- 3 – Exclusive Process - Only one COMPUTE process is allowed to run on the GPU

gpu_status

```
root@gpu:~# pbsnodes gpu
gpu
...
     gpus = 2
     gpu_status = gpu[1]=gpu_id=0:6:0;gpu_product_name=Tesla
     C2050;gpu_display=Disabled;gpu_pci_device_id=6D110DE;gpu_pci_location_id=0:6:0;
     gpu_fan_speed=54 %;gpu_memory_total=2687 Mb;gpu_memory_used=74
Mb;gpu_mode=Default;gpu_state=Unallocated;gpu_utilization=96
%;gpu_memory_utilization=10
%;gpu_ecc_mode=Enabled;gpu_single_bit_ecc_errors=0;gpu_double_bit_ecc_errors=
0;gpu_temperature=88 C,gpu[0]=gpu_id=0:5:0;gpu_product_name=Tesla
C2050;gpu_display=Enabled;gpu_pci_device_id=6D110DE;gpu_pci_location_id=0:5:0;
gpu_fan_speed=66 %;gpu_memory_total=2687 Mb;gpu_memory_used=136
Mb;gpu_mode=Default;gpu_state=Unallocated;gpu_utilization=96
%;gpu_memory_utilization=10
%;gpu_ecc_mode=Enabled;gpu_single_bit_ecc_errors=0;
gpu_double_bit_ecc_errors=0;gpu_temperature=86 C,driver_ver=270.41.06,timestamp=Wed
May 4 13:00:35
2011
```

New NVIDIA GPU support

qsub allows specifying required compute mode when requesting GPUs. If no GPU mode is requested, it will default to "exclusive" for Nvidia driver version 260 or "exclusive_thread" for NVIDIA driver version 270 and above.

- qsub -l nodes=1:ppn=1:gpus=1
- qsub -l nodes=1:gpus=1
- qsub -l nodes=1:gpus=1:exclusive_thread
- qsub -l nodes=1:gpus=1:exclusive_process
- qsub -l nodes=1:gpus=1:reseterr
- qsub -l nodes=1:gpus=1:reseterr:exclusive_thread (exclusive_thread:reseterr)
- qsub -l nodes=1:gpus=1:reseterr:exclusive_process

Related topics

- **Scheduling GPUs** on page 715
- **Using GPUs with NUMA** on page 716

20.4 GPU Metrics

GPU metrics can be collected for nodes that:

- Have one or more GPUs.
- Run TORQUE 2.5.x or later.

- Use NVIDIA drivers v260.x or v270.x.

GPU metric tracking must be enabled in `moab.cfg`:

```
RMCFG[torque]  flags=RECORDGPUMETRICS
```

There is one GPU metric for all GPU devices within a node (gpu_timestamp) and nine GPU metrics for each GPU device within a node. If the maximum GPU devices within a node is 4, you must increase the MAXGMETRIC value in `moab.cfg` by (maxgpudevices x gpumetrics) + 1. In this case, the formula is (4 x 9) + 1 = 37, so whatever the MAXGMETRIC value is, it must be increased by 37. This way, when enabling GPU metrics recording, Moab has enough GMETRIC types to accommodate the GPU metrics.

GPU Metrics Map

The GPU metric names map is as follows (where *X* is the GPU number):

Metric name as returned by pbsnodes	GMETRIC name as stored in Moab	Metric output
timestamp	gpu_timestamp The gpu_timestamp metric is global to all GPUs on the node and indicates the last time the driver collected information on the GPUs.	The time data was collected in epoch time
gpu_fan_speed	gpuX_fan	The current fan speed as a percentage
gpu_memory_total	gpuX_mem	The total GPU memory in megabytes
gpu_memory_used	gpuX_usedmem	The total used GPU memory in megabytes
gpu_utilization	gpuX_util	The GPU capability currently in use as a percentage
gpu_memory_utilization	gpuX_memutil	The GPU memory currently in use as a percentage
gpu_ecc_mode	gpuX_ecc	Whether ECC is enabled or disabled

Metric name as returned by pbsnodes	GMETRIC name as stored in Moab	Metric output
gpu_single_bit_ecc_errors	gpuX_ecc1err	The total number of EEC single-bit errors since the last counter reset
gpu_double_bit_ecc_errors	gpuX_ecc2err	The total number of EEC double-bit errors since the last counter reset
gpu_temperature	gpuX_temp	The GPU current temperature in Celsius

Example 20-1: GPU example

```
$ mdiag -n -v --xml

<Data>
<node AGRES="GPUS=2;"
AVLCLASS="[test 8][batch 8]"
CFGCLASS="[test 8][batch 8]"
GMETRIC="gpu1_fan:59.00,gpu1_mem:2687.00,gpu1_usedmem:74.00,gpu1_util:94.00,gpu1_
memutil:9.00,gpu1_ecc:0.00,gpu1_ecc1err:0.00,gpu1_ecc2err:0.00,gpu1_temp:89.00,gpu0_
fan:70.00,gpu0_mem:2687.00,gpu0_usedmem:136.00,gpu0_util:94.00,gpu0_memutil:9.00,gpu0_
ecc:0.00,gpu0_ecc1err:0.00,gpu0_ecc2err:0.00,gpu0_temp:89.00,gpu_
timestamp:1304526680.00"
GRES="GPUS=2;"
LASTUPDATETIME="1304526518" LOAD="1.050000"
MAXJOB="0" MAXJOBPERUSER="0" MAXLOAD="0.000000" NODEID="gpu"
NODEINDEX="0" NODESTATE="Idle" OS="linux" OSLIST="linux"
PARTITION="makai" PRIORITY="0" PROCSPEED="0" RADISK="1"
RAMEM="5978" RAPROC="7" RASWAP="22722" RCDISK="1" RCMEM="5978"
RCPROC="8" RCSWAP="23493" RMACCESSLIST="makai" SPEED="1.000000"
STATMODIFYTIME="1304525679" STATTOTALTIME="315649"
STATUPTIME="315649"></node>
</Data>
```

20.5 Intel® Xeon Phi™ Coprocessor Configuration

Intel Many-Integrated Cores (MIC) architecture configuration

If you use an Intel Many-Integrated Cores (MIC) architecture-based product (e.g., Intel Xeon Phi™) in your cluster for parallel processing, you must configure TORQUE to detect them.

Prerequisites

- TORQUE 4.2 or later
- If you set up TORQUE using auto-detection and intend to get the MIC-based device status report, you must build pbs_mom on a system that has the lower-level API libraries for the MIC-based device(s) installed. Additionally, every MOM built with `--enable-mics` and running on a compute node must already have the lower-level API libraries installed on the node. Note that the library is called `coi_host`. You must obtain the API libraries from Intel.

Setup Options

There are two ways to configure MIC-based devices with TORQUE: (1) manually and (2) by auto-detection.

Manual configuration

- Add `mics=X` to the nodes file for the appropriate nodes.

```
napali np=12 mics=2
```

Auto-detect

When you use auto-detection, pbs_mom discovers the MIC-based devices and reports them to pbs_server.

- At build time, add `--enable-mics` to the configure line.

```
./configure --enable-mics <other configure options>
```

Validating the configuration

TORQUE

pbsnodes

Example 20-2: pbsnodes output

```
slesmic
       state = free
       np = 100
       ntype = cluster
       status =
rectime=1347634381,varattr=,jobs=,state=free,netload=7442004852,gres=,loadave=0.00,ncp
us=32,physmem=65925692kb,availmem=66531344kb,totmem=68028984kb,idletime=59059,nusers=2
,nsessions=8,sessions=4387 4391 4392 4436 4439 4443 4459 100395,uname=Linux slesmic
3.0.13-0.27-default #1 SMP Wed Feb 15 13:33:49 UTC 2012 (d73692b) x86_64,opsys=linux
       mom_service_port = 15002
       mom_manager_port = 15003
       mics = 2
       mic_status = mic[1]=mic_id=8796;num_cores=61;num_threads=244;physmem=8065748992;free_
physmem=7854972928;swap=0;free_swap=0;max_frequency=1090;isa=COI_ISA_
KNC;load=0.000000;normalized_load=0.000000;,mic[0]=mic_id=8796;num_cores=61;num_
threads=244;physmem=8065748992;free_physmem=7872712704;swap=0;free_swap=0;max_
frequency=1090;isa=COI_ISA_KNC;load=0.540000;normalized_load=0.008852;

rhmic.ac
       state = free
```

```
       np = 100
       ntype = cluster
       status =
rectime=1347634381,varattr=,jobs=,state=free,netload=3006171583,gres=,loadave=0.00,ncp
us=32,physmem=65918268kb,availmem=66901588kb,totmem=67982644kb,idletime=59477,nusers=2
,nsessions=2,sessions=3401 29320,uname=Linux rhmic.ac 2.6.32-220.el6.x86_64 #1 SMP Tue
Dec 6 19:48:22 GMT 2011 x86_64,opsys=linux
       mom_service_port = 15002
       mom_manager_port = 15003
       mics = 1
       mic_status = mic[0]=mic_id=8796;num_cores=61;num_threads=244;physmem=8065748992;free_
physmem=7872032768;swap=0;free_swap=0;max_frequency=1090;isa=COI_ISA_
KNC;load=0.540000;normalized_load=0.008852;<mic_status>;
```

Moab

mdiag -n -v

Example 20-3: mdiag -n -v output

```
$ mdiag -n -v
compute node summary
Name                    State   Procs      Memory         Disk          Swap
Speed   Opsys   Arch Par   Load Classes                        Features

hola                     Idle    4:4      8002:8002        1:1       10236:13723
1.00   linux      - hol   0.24 [batch]                       -
GRES=MICS:2,
-----                    ---     4:4      8002:8002        1:1       10236:13723

Total Nodes: 1  (Active: 0  Idle: 1  Down: 0)
```

checknode -v

Example 20-4: checknode output

```
$ checknode slesmic
node slesmic

State:      Idle  (in current state for 00:00:16)
Configured Resources: PROCS: 100  MEM: 62G  SWAP: 64G  DISK: 1M  MICS: 2
Utilized   Resources: SWAP: 1581M
Dedicated  Resources: ---
Generic Metrics:    mic1_mic_id=8796.00,mic1_num_cores=61.00,mic1_num_
threads=244.00,mic1_physmem=8065748992.00,mic1_free_physmem=7854972928.00,mic1_
swap=0.00,mic1_free_swap=0.00,mic1_max_frequency=1090.00,mic1_load=0.12,mic1_
normalized_load=0.00,mic0_mic_id=8796.00,mic0_num_cores=61.00,mic0_num_
threads=244.00,mic0_physmem=8065748992.00,mic0_free_physmem=7872679936.00,mic0_
swap=0.00,mic0_free_swap=0.00,mic0_max_frequency=1090.00
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   0.000
Classes:    [batch]
RM[napali]* TYPE=PBS
EffNodeAccessPolicy: SHARED

Total Time: 3:45:43  Up: 3:45:43 (100.00%)  Active: 00:00:00 (0.00%)

Reservations:
  ---
```

Job submission

Syntax

Example 20-5: Request MIC-based device(s) in qsub

```
qsub .... -l nodes=X:mics=Y
```

Because these resources are delimited with a colon, this command requests a job with X nodes and Y mics per task. If you run the same command and delimit the resources with a comma (`qsub .... -l nodes=X,mics=Y`), you request a job with X nodes and Y mics per job.

qstat -f

Example 20-6: qstat -f output

```
Job Id: 5271.napali
Job_Name = STDIN
Job_Owner = dbeer@napali
job_state = Q
queue = batch
server = napali
Checkpoint = u
ctime = Fri Sep 14 08:56:33 2012
Error_Path = napali:/home/dbeer/dev/private-torque/trunk/STDIN.e5271
Hold_Types = n
Join_Path = oe
Keep_Files = n
Mail_Points = a
mtime = Fri Sep 14 08:56:33 2012
Output_Path = napali:/home/dbeer/dev/private-torque/trunk/STDIN.o5271
Priority = 0
qtime = Fri Sep 14 08:56:33 2012
Rerunable = True
Resource_List.neednodes = 1:mics=1
Resource_List.nodect = 1
Resource_List.nodes = 1:mics=1
substate = 10
Variable_List = PBS_O_QUEUE=batch,PBS_O_HOME=/home/dbeer,
       PBS_O_LOGNAME=dbeer,
       PBS_O_PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
       in:/usr/games,PBS_O_MAIL=/var/mail/dbeer,PBS_O_SHELL=/bin/bash,
       PBS_O_LANG=en_US.UTF-8,
       PBS_O_SUBMIT_FILTER=/usr/local/sbin/torque_submitfilter,
       PBS_O_WORKDIR=/home/dbeer/dev/private-torque/trunk,PBS_O_HOST=napali,
       PBS_O_SERVER=napali
euser = dbeer
egroup = company
queue_rank = 3
queue_type = E
etime = Fri Sep 14 08:56:33 2012
submit_args = -l nodes=1:mics=1
fault_tolerant = False
job_radix = 0
submit_host = napali
```

checkjob -v

Example 20-7: checkjob -v output

```
dthompson@mahalo:~/dev/moab-test/trunk$ checkjob -v 2
job 2 (RM job '2.mahalo')

AName: STDIN
State: Idle
Creds:  user:dthompson  group:dthompson  class:batch
WallTime:   00:00:00 of 1:00:00
SubmitTime: Thu Sep 13 17:06:06
(Time Queued  Total: 00:00:24  Eligible: 00:00:02)

TemplateSets:  DEFAULT
Total Requested Tasks: 1

Req[0]  TaskCount: 1  Partition: ALL
Dedicated Resources Per Task: PROCS: 1  MICS: 1

...
```

20.6 Intel® Xeon Phi™ Co-processor Metrics

Intel Many-Integrated Cores (MIC) architecture-based device (e.g., Intel Xeon Phi™) metrics can be collected for nodes that:

- Have one or more MIC-based devices.
- Run TORQUE 4.2.x or later.
- Run Moab 7.2 or later.

MIC-based device metric tracking must be enabled in `moab.cfg`:

```
RMCFG[torque]  flags=RECORDMICMETRICS
```

There are 11 metrics for each MIC-based device within a node. If the maximum MIC-based devices within a node is 4, you must increase the MAXGMETRIC value in `moab.cfg` by (*maxmicdevices* x *micmetrics*). In this case, the formula is (4 x 11) = 44, so whatever the **MAXGMETRIC** value is, it must be increased by 44. This way, when enabling MIC-based device metrics recording, Moab has enough GMETRIC types to accommodate the additional metrics.

MIC-based Metrics Map

The MIC-based metric names map is as follows (where *X* is the MIC-based device number):

Metric name as returned by pbsnodes	GMETRIC name as stored in Moab	Metric output
mic_id	micX_mic_id	The ID of the MIC-based device
num_cores	micX_num_cores	The number of cores in the MIC-based device
num_threads	micX_num_threads	The number of hardware threads on the MIC-based device
physmem	micX_physmem	The total physical memory in the MIC-based device
free_physmem	micX_free_physmem	The available physical memory in the MIC-based device
swap	micX_swap	The total swap space on the MIC-based device
free_swap	micX_free_swap	The unused swap space on the MIC-based device
max_frequency	micX_max_frequency	The maximum frequency speed of any core in the MIC-based device
isa	micX_isa	The hardware interface type of the MIC-based device
load	micX_load	The total current load of the MIC-based device
normalized_load	micX_normalized_ load	The normalized load of the MIC-based device (total load divided by number of cores in the MIC-based device)

21.0 VMs

- **Policy-based VM Migration** on page 727
- **Overcommit Factor and Threshold** on page 729
- **Overutilization Migration** on page 731
- **Green Migration and Consolidation** on page 731

21.1 Policy-based VM Migration

One of the unique features of Moab Adaptive Computing Suite is policy-based VM migration. Using information about data center applications and server load, Moab can aim to keep VMs in the data center optimally distributed across hypervisors.

There are two types of policy-based VM migration:

- Performance-based Migration
- Consolidation-based Migration (formerly known as "Green Migration")

These two types have differing goals that at times may seem at odds with one another. Fortunately, Moab's scheduling engine is, in most cases, able to resolve and satisfy both sets of goals:

- Performance goal: To equalize loads across hypervisors as migrations are queued due to overcommit conditions. This places VMs to be migrated on the least-loaded HV available.
- Consolidation goal: To load hypervisors as close to thresholds as possible, without exceeding them. This policy places VMs to be migrated on the most loaded HV possible, within these constraints. A second loop of this policy will select lightly-loaded hypervisors to be evacuated completely.

This chapter will explain each type of migration in more detail and the general steps taken by Moab to find the best VM placement. It will also discuss how to configure Moab to perform automatic migration.

- Overcommit Factor and Threshold
- Overutilization Migration
- Green Migration and Consolidation

Throttling VM Migration

When the event occurs that triggers the migration you will want to prevent your infrastructure from being overloaded with migrations. In the following example we use GRES to limit the migration of VMs to 10.

Example of throttling in the Moab.cfg:

```
# Limit for reservation-based migration throttle Moab.cfg
#Throttle the number of VMs that can be migrated at any given time to 10
VMMigrateThrottle 10
```

General Notes

- If the RM reports a VM's resources as undefined or 0, or the VM's reported resources do not meet the following criteria, Moab prevents the VM from migrating.
  - The VM's state must not be "Unknown."
  - The VM must have a reported CPULOAD greater than 0.
  - The VM must have an AMEMORY less than its CMEMORY. This indicates that some memory is currently in use and tells Moab that the RM is reporting memory correctly.
- The RM must report a hypervisor's HVType and a positive CPU and memory load for Moab to consider it for migration; however, Moab will allow migration to hypervisors without a reported CPU or memory load if VMMIGRATETOZEROLOADNODES is *TRUE*.
- The mvmctl -f command, run with the *eval* flag, does not submit migration jobs, but returns a report detailing the actions that would occur if you configured a VM migration policy.

Example `moab.cfg`

```
SCHEDCFGMoab SERVER=server:42559

#Note for this you would need to have a xcat and msm configuration. For convenience I
have removed those to keep this example short
#VM Management

HideVirtualNodes Transparent
AllowVMMigration TRUE

VMMigrateThrottle 10

PARCFG[ALL] VMMigrateDuration=0:40:00
PARCFG[ALL] VMCreateDuration=01:00:00
PARCFG[ALL] VMDeleteDuration=30:00

#List of images available

IMAGECFG[xcat] VMOSLIST=suse11,win2008,rhel54

#Set the powerpolicy for each node

NODECFG[DEFAULT] POWERPOLICY=green

#Number of systems to keep active for a resource in the pool. This example keeps one
system available if VM migration is needed.

MaxGreenStandByPoolSize 1

PARCFG[xcat] NODEPOWEROFFDURATION=20:00
PARCFG[xcat] NODEPOWERONDURATION=20:00

#Max time to wait before a hypervisor is powered off if that hypervisor is idle. This
is a green setting
NodeIdlePowerThreshold 40:00:00
```

```
#Overcommit settings and green settings

NODEAVAILABILITYPOLICY UTILIZED
VMMigrationPolicy OVERCOMMIT,CONSOLIDATION@10:00:00
VMOCThreshold PROC:0.7,MEM:0.95
NODECFG[DEFAULT] OVERCOMMIT=PROC:2.0,MEM:2.0

GEVENTCFG[disk_free] ACTION=fail SEVERITY=4
NODECFG[DEFAULT]
TRIGGER=atype=exec,etype=threshold,failoffset=1:00,threshold=gmetricdisk_
free>90,action="/opt/moab/tools/filesize_fault.py $OID $METRICTYPE"
```

21.2 Overcommit Factor and Threshold

The two main configuration settings that govern how migrations work are the Overcommit Factor and Overcommit Threshold. Both can be applied to the processors and memory of virtual machines (VM's).

The Overcommit Factor and Threshold can be defined as a global default or on a per-node basis.

```
NODECFG[DEFAULT]  OVERCOMMIT=PROC:2.0,MEM:2.0  # This is the default global policy
NODECFG[node42]   OVERCOMMIT=PROC:3.0,MEM:3.0  # This is a node-specific policy for
node42
```

Overcommit Factor defines the upper bound or maximum amount of VCPUs that can be created on any given hypervisor (HV). For example, if you have a hypervisor with 12 processors or cores (Moab sees them as 12 processors), and have an Overcommit Factor of 2.0 for procs, then Moab will not allow, under any condition, more than 24 VCPU's to be allocated on this hypervisor. Remember: a VM can have one or more VCPU's. So, in this example, the HV could only support 8 VM's if they all had 3 VPCU's each. It could support 4 VM's if they had 6 VPCU's each, and so forth.

The Overcommit Threshold defines how many VM's are allowed on a node if those VM's have a load being reported. The Overcommit Factor defines the maximum under any condition, but the Overcommit Threshold controls how many can be practically supported by a hypervisor due to load.

An Overcommit Threshold is a number between 0 and 1 and is interpreted as a percentage that is applied to the number of configured processors. It is not applied to the overcommitted processor count. For example, if we have an Overcommit Threshold of 0.7 for CPUs and a hypervisor with 12 configured processors, then that HV can support a CPU load of up to 8.4 before Moab will try to migrate VM's off of it. Moab uses the CPULOAD reported for the hypervisor to determine if the threshold is exceeded.

An example using both the Overcommit Factor and Overcommit Threshold is as follows:

- We have a hypervisor with 12 procs, an overcommit factor of 2.0 and a threshold of 0.7.

  We initially create VM's on the hypervisor. Each VM has a VCPU count of 1 and a CPULOAD of 0.1. We keep creating VM's on the hypervisor until we have 24. At this point, Moab will not any more VMs to be created because the Overcommit Factor has been reached (12x2.0=24). The hypervisor reports a load of about 2.4 (this will usually be greater than the sum of the VM loads due to overhead and VMs that are not under Moab's control). The load is well below the threshold of 0.7 (threshold) * 12 (number of processors) = 8.4, so no VM's need be migrated.

```
#Example of the above implemented in Moab.cfg
NODEAVAILABILITYPOLICY      UTILIZED
VMMigrationPolicy           OVERCOMMIT,CONSOLIDATION@10:00:00
NODECFG[DEFAULT]            VMOCThreshold=PROC:0.7,MEM:0.95
NODECFG[DEFAULT]            OVERCOMMIT=PROC:2.0,MEM:2.0
```

- The example above first specifies **NODEAVAILABILITYPOLICY** *UTILIZED*. This tells Moab to use what the resource manager is reporting as being used for resources. Next the *OVERCOMMIT* flag must be set on **VMMigrationPolicy** for the **VMOCTHRESHOLD** attribute to function. *CONSOLIDATION* is then also specified with an Overcommit migration even time of *10:00:00* or 10 minutes. Next **VMOCThreshold** is then specified to overcommit *PROC* by *.7* and *MEM* by *.95* allowing PROC utilization to reach 8.2 and MEM to reach 22.8 Gigs (.95 x 24) before action needs to be taken. Finally each node is given a default **OVERCOMMIT** for *PROC* by *2* and *MEM* by *2*. This could allow up to 24 VM's at 1 processor and 2 Gigs of memory each.
- Over time, the VMs start to get used more and their CPULOAD increases. Soon, four of the VM's load shoots up to 3. This brings the load to 12 x 3.0 = 36. This is above the Overcommit Threshold of 8.4, so Moab will now need to migrate VM's during the next Overcommit Migration event.
- When the Overcommit Migration event occurs, Moab will migrate enough of the high-load VMs off of the hypervisor to bring the load back down below 8.4. In this example, Moab would need to migrate at least TWO of the high-load VMs, bringing the total load down to 6.2. Once these two VM's have been migrated, Moab should not migrate any more, as we are now below our threshold. Moab should migrate these two VM's in the same Overcommit event--it should not take multiple events to migrate both of them.

Example of Overcommit Migration event

```
#GMETRIC threshold based triggers
# WLM Metric Threshold to check file system utilization
GEVENTCFG[disk_free] ACTION=fail SEVERITY=4
NODECFG[DEFAULT] TRIGGER=atype=exec,etype=threshold,failoffset=1:00,threshold=gmetric
[disk_free]>90,action="/opt/moab/tools/filesize_fault.py $OID $METRICTYPE"
```

In the above example, a GEVENT of type *disk_free* is created. This is one of the predefined GEVENTS provided by Moab. The action is set to fail if the event is triggered and give it an arbitrary **SEVERITY** of *4*. Next, the nodes to be defined with this event are specified. In this example this is applied to all nodes. If the threshold is over 90, Moab is informed the following action will take effect: `/opt/moab/tools/filesize_fault.py $OID $METRICTYPE`

Example of supported GMETRIC. Note this is not an exhaustive list:

```
"bytes_out"Number of net bytes out per second
"cpu_num" Number of CPUs
"cpu_speed" processor speed (in MHz)
"disk_free" Total free disk space (GB)
"disk_total" Total available disk space
"load_one" One minute load average
"machine_type" cpu architecture
"mem_free" Amount of available memory (KB)
"mem_total"  Amount of available memory
"os_name"
"os_release" operating system release
"pkts_in" NYI */ / Packets in per second (packets/sec)
"pkts_out" NYI Packets out per second
```

```
"swap_free" Amount of available swap memory (KB)
"swap_total"  Total amount of swap memory
```

Note that the Overcommit Factor and Threshold should also apply when selecting a VM destination. If a VM needs to be migrated off of a loaded hypervisor Y, but moving it to hypervisor X would cause X's load or overcommit factor to be violated, Moab cannot move it to X. It must try to find another location. Also, the example above dealt only with CPU or processor counts, but Overcommit Factor and Threshold also apply to a wide array of system resources. You can also create your own with Ganglia.

21.3 Overutilization Migration

Overutilization occurs when a hypervisor's resource usage load goes above a defined threshold. Once this occurs, Moab can be configured to migrate any VM's present on the hypervisor to other, less-used hypervisors.

Calculations

In terms of values returned by the resource managers, a hypervisor's Utilization Threshold is exceeded (and migrations should occur at the next overcommit migration event) if:

```
CPULOAD(HV) > VMOCTHRESHOLD(PROC) * CPROC(HV)
```

or if:

```
(CMEMORY(HV) - AMEMORY(HV)) / CMEMORY(HV) > VMOCTHRESHOLD(MEM) * CMEMORY(HV)
```

Moab will calculate the quantity of virtual machines to remove by subtracting the sum of the CPULOAD (VM)'s and/or the (CMEMORY(VM) - AMEMORY(VM))'s from the corresponding processor or memory hypervisor loads until the overcommit condition is improved.

As mentioned in the example in Overcommit Factor and Threshold we see the following from the configuration:

```
NODECFG[DEFAULT] OVERCOMMIT=PROC:2.0,MEM:2.0
```

Moab will monitor the resources based off the resource manager's reports and then migrate the VM to a less used hypervisor. Note there are many parameters that this can be attached to using GMETRIC.

21.4 Green Migration and Consolidation

As stated previously, the goal of consolidation is to minimize the number of hypervisors with one or more VMs on them. This is accomplished by migrating VMs from lightly utilized hypervisor onto other hypervisors. The primary goal is to completely evacuate as many hypervisors as possible. Additional migrations to make a hypervisor lightly loaded are also desirable. The VMs from a particular hypervisor can be migrated onto more than one target hypervisor.

Calculations

In terms of values returned by the resource managers, a hypervisor will not allow a VM to be provisioned or a migration to occur that violates a UTILIZATION Threshold (if set) or that violate

```
SUM(CPROC(VM)) > OVERCOMMIT(PROC) * CPROC(HV)
```

or violates:

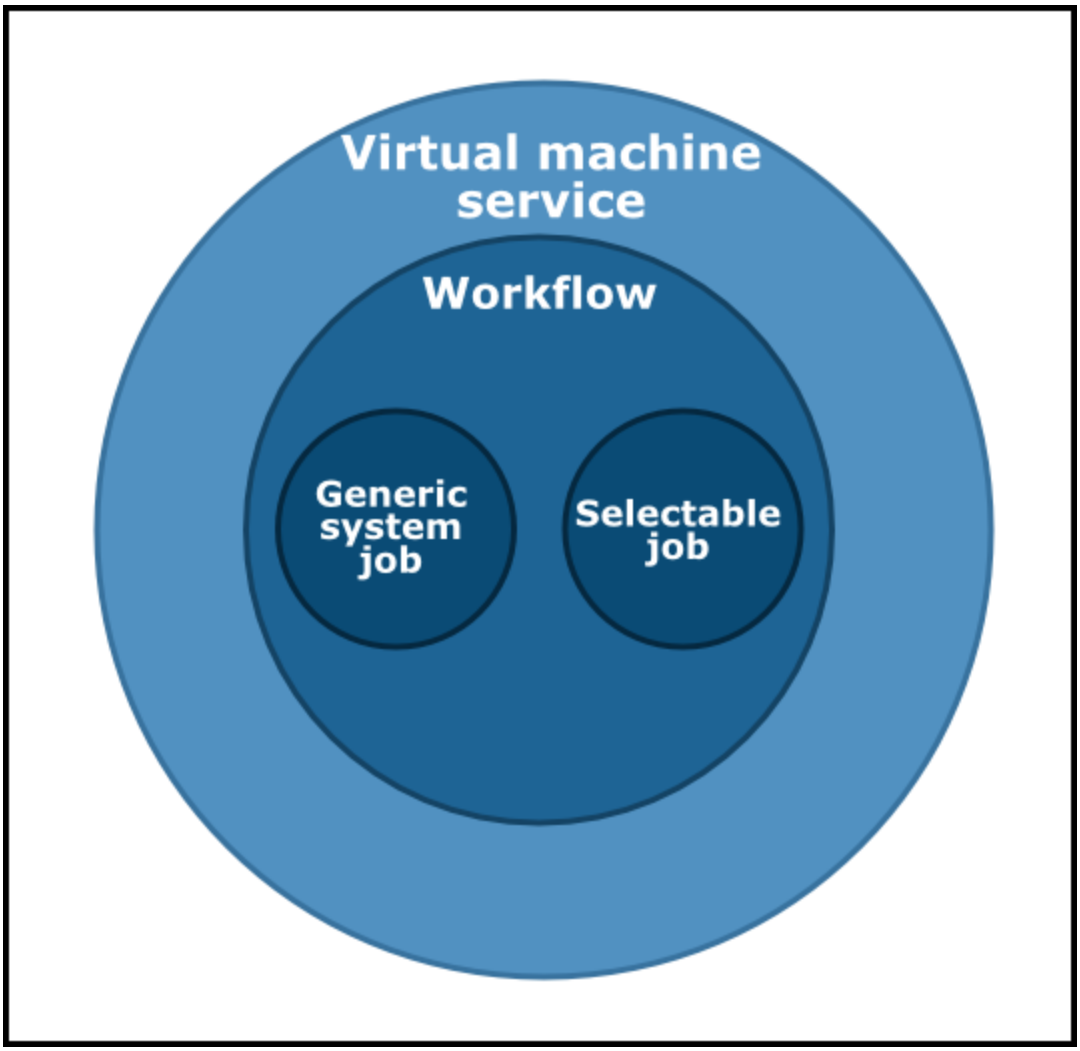
```
SUM(CMEMORY(VM)) > OVERCOMMIT(MEM) * CMEMORY(HV)
```

```
AllowVMMigration TRUE
AggregateNodeActions False
VMMigrateThrottle 10
PARCFG[ALL] VMCreateDuration=01:00:00
PARCFG[ALL] VMDeleteDuration=30:00
PARCFG[ALL] VMMigrateDuration=0:40:00
NODECFG[DEFAULT] POWERPOLICY=OnDemand
#MaxGreenStandByPoolSize considered a standby pool. Default value is 0 and when 0 is
set it disables the standbypool. Nodes can be evaluated based off idle rate to be
added to the standby pool.
MaxGreenStandByPoolSize 0
PARCFG[xcat] NODEPOWEROFFDURATION=20:00
PARCFG[xcat] NODEPOWERONDURATION=20:00
NodeIdlePowerThreshold 40:00:00 # Time a node must be idle before we shut it down
```

22.0 Workload-Driven Cloud Services

22.1 About workload-driven cloud services

A cloud service is one or more job workflows held in a virtual container — each comprising a separate but related piece of the service — that create, set up, and maintain it. Individual jobs within the workflows implement Moab triggers to run scripts that perform certain tasks and to set and receive variables. Variables correctly sequence the jobs that set up the services and allow scripts further in the workflow to locate them. The following example illustrates the composition of a simple VM service without storage:



A VC contains the job workflow, its jobs, and their triggers and shared variables. That workflow consists of a number of generic system jobs and a single selectable job that the user submits when requesting the service or service component.

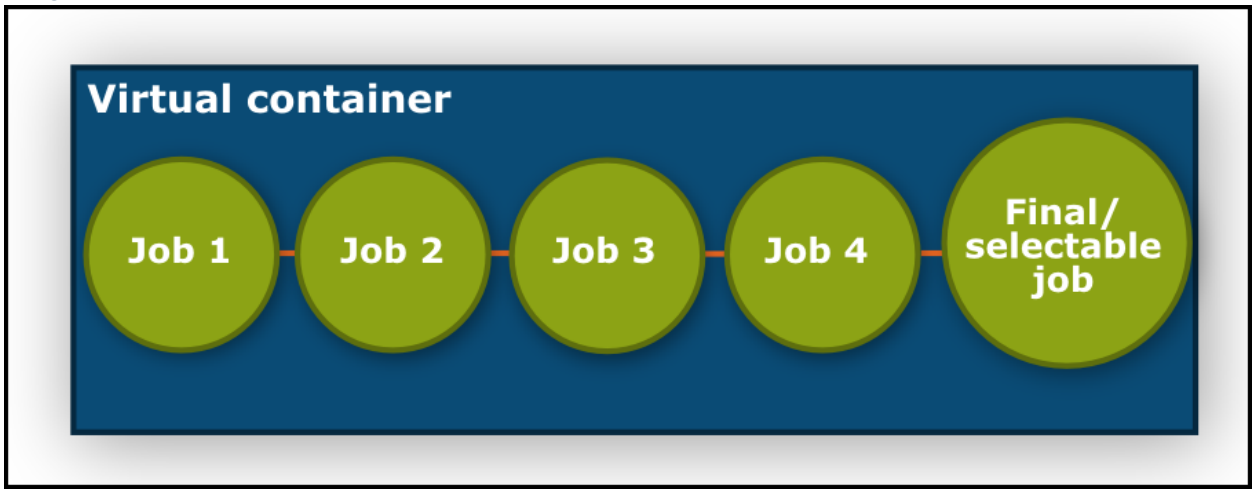
Workflows

A cloud workflow contains a series of generic system jobs, or system jobs with a trigger, with dependencies and variables. The workflow also contains a selectable job, which must meet the following criteria:

1. It is not a generic system job.
2. It is the final job in the workflow.
3. If the service is a VM, it is a VM-tracking job.

Each job, besides the first in the workflow, has a dependency on another job to ensure that they run in the correct order. Workflows must be linear; they cannot branch.

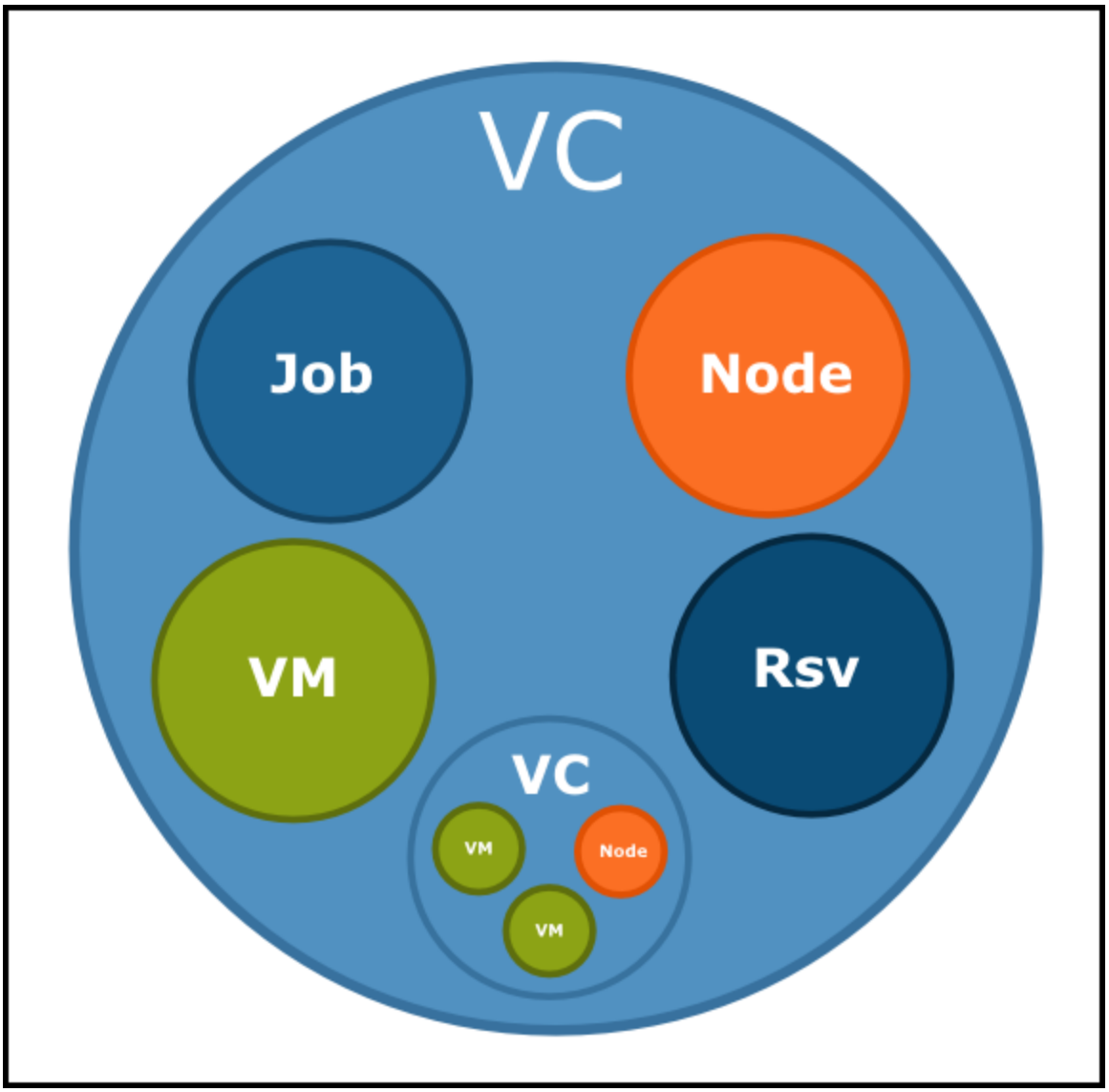
Image 22-1: Workflow example



As shown in the illustration above, the user requests the selectable job using the msub command. Moab creates it and the other jobs inside of a VC. They run in the order specified by the template dependencies, ending with the selectable job.

Virtual Containers

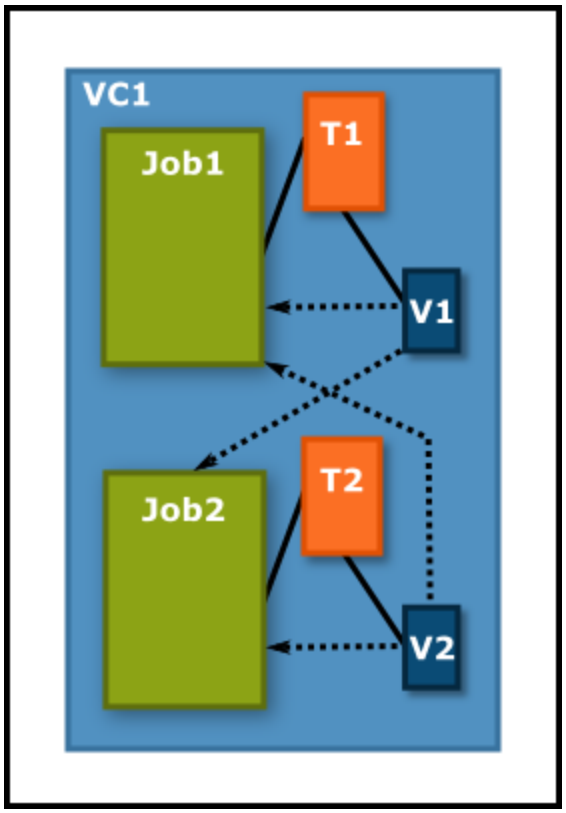
Virtual Containers (VCs) are logical groupings of objects with a shared variable space. They can hold jobs, reservations, nodes, VMs, and other VCs, including services.



This image illustrates a VC containing all possible object types, a PM workflow, and a child VC containing a VM service. The child VC contains two virtual machines and one physical machine workflow.

Variables

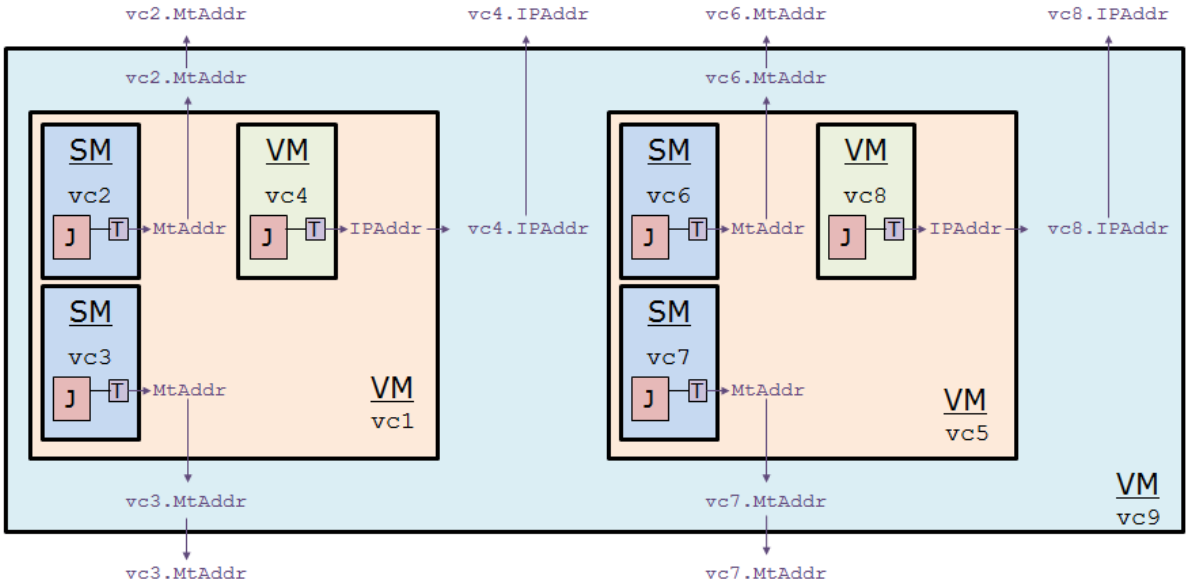
Variables are the method by which jobs in a workflow, or even jobs within separate subservices of a single service, send messages to one another. A variable could be the IP address of a new VM, an indication of script success or failure, or another piece of information that affects other jobs in the service. By default, trigger variables are only accessible to its parent object (in this case, the job to which the trigger is attached); however, If configured to do so, triggers will pass their variables up to the VC, granting all objects access to one another's variables.



The image above shows `vc1`, a container comprised of two jobs. Each job has a trigger that sets a variable and it passes it up to `vc1`. When `vc1` receives the variables, they become available to all objects. `Job2` can receive *V1*, and `Job1` can receive *V2*.

Additionally, if `vc1` is the child of another VC, it passes all those variables up to its parent. This means that, as long as they have one VC in common, an object buried deep inside of multiple VCs can access a variable set by an object deep inside another series of VCs.

When variables are to be passed further up than one VC, Moab applies a name space before sending them to the parent VC(s). This way, if variables originating from separate components of a single service have the same name, they will not overwrite each other. A variable's new name becomes the name of its immediate VC and its original name with a period between them (*<vcName>.<varName>*). For example, if *V2* in the image above were passed up to `vc1`'s parent VC, its name would become *vc1.V2*.



The VM service in `vc9` pictured above is made up of two VMs services, each containing one VM and two storage mounts. The six child containers each have a job with a trigger that sets a variable. When the variables are passed up from the first VC to the second, they inherit the name of the first VC as part of their own name. If this did not occur, `vc9` would have four *MtAddr* and two *IPAddr* variables overwriting one another. A job that requests a variable originating from a different VC from its own would most likely receive the wrong information. When configuring a trigger, you can request name spaces in the argument list so that the script only pulls in the desired variable(s).

Setting up Workload-Driven Cloud in Moab

1. Enable VMs and VM-tracking jobs in Moab. See Enabling Cloud Services.
2. Configure the setup templates using generic system jobs, where the trigger called will interact with underlying systems or resource managers. See **Creating a generic system job** on page 739.
3. Set up job templates in `moab.cfg` that define generic workflows for all create VM processes. You will likely create template workflows for other resources as well, such as storage or VLANs. Configure the workflows with job template dependencies for receiving a host name, provisioning an OS, setting up software, or other tasks. Set a single selectable job template to represent the main resource for each workflow. See **Creating a cloud workflow** on page 740 and Requesting Name Space Variables.
4. Create a virtual container to hold the workflow(s) that create the VM. See Creating a VC to Hold a Service.

Related Tasks:

- Enabling Cloud Services
- Creating a Generic System Job
- Creating a Cloud Workflow

- Creating a Service
- Canceling a Service

Reference topics:

- Cloud-Specific Job Template Attributes
- Generic System Job Trigger Requirements
- VM Service Example

22.2 Tasks

22.2.1 Enabling cloud services

To configure Moab for cloud services

1. In the Moab configuration file, set HIDEVIRTUALNODES to *TRANSPARENT* and VMTRACKING to *TRUE*.

```
HIDEVIRTUALNODES  TRANSPARENT
VMTRACKING        TRUE
```

   HIDEVIRTUALNODES enables VM management and reveals hypervisors, and **VMTRACKING** turns on Moab's ability to use VM-tracking jobs to represent VMs in the job queue.

2. Optional: By default, Moab takes no action when a VM expires or becomes stale (has not been reported by the RM for five 30-second iterations). To customize this behavior, modify or add the following parameters in `moab.cfg`:

   - ENABLEVMDESTROY - causes Moab to automatically destroy VMs when their walltime expires or, if **VMSTALEACTION** is *DESTROY*, when they become stale.
   - VMSTALEACTION - specifies what action Moab should take when a VM becomes stale.
   - VMSTALEITERATIONS - specifies how many consecutive iterations a VM must not be reported by the RM for Moab to consider it stale.
   - RMPOLLINTERVAL - sets the length of an iteration.

   In the following example, Moab destroys stale VMs after three 60-second iterations.

```
RMPOLLINTERVAL    60,60
...
VMSTALEACTION     DESTROY
ENABLEVMDESTROY   TRUE
VMSTALEITERATIONS 3
```

22.2.2 Creating a generic system job

Context

Generic system jobs, or system jobs with a trigger, are created via job templates. They can be selectable, but they should not be when used in a cloud workflow.

To create a generic system job

1. Create a job template that sets the GENERICSYSJOB attribute.
2. Create a trigger that meets the requirements detailed in **Generic system job trigger requirements on page 747** and make it the value of **GENERICSYSJOB**.

```
JOBCFG[gen]
GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/genericTrig.py",Timeout=5:00,Flags=
objectxmlstdin
```

22.2.3 Creating a cloud workflow

Context

The following procedure details the steps required to create a workflow that builds a VM. This workflow contains four job templates:

1. *createVM* - The system job that creates the VM. It starts the workflow but may not run immediately. The trigger sets the IP address of the VM as a variable when it successfully completes.
2. *installSoftware* - The system job that installs software on the new VM. It depends on the successful completion of *createVM* and uses the IP address variable to locate the correct VM.
3. *createVMWithSoftware* - The VM tracking job the user submits to request the workflow. It is the final job in the workflow and depends on the successful completion of *installSoftware*.
4. *destroy* - The job that, if *createVMWithSoftware* is canceled, runs a script to delete the VM.

Image 22-2: Sample cloud workflow in submit order

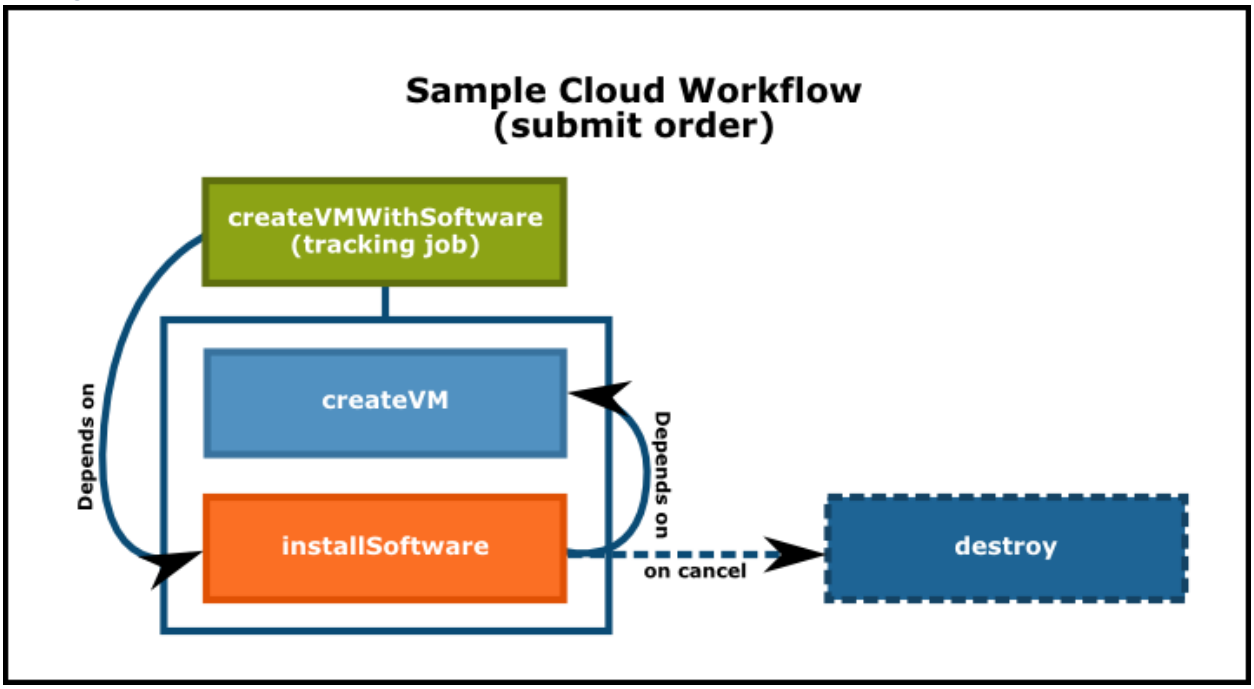
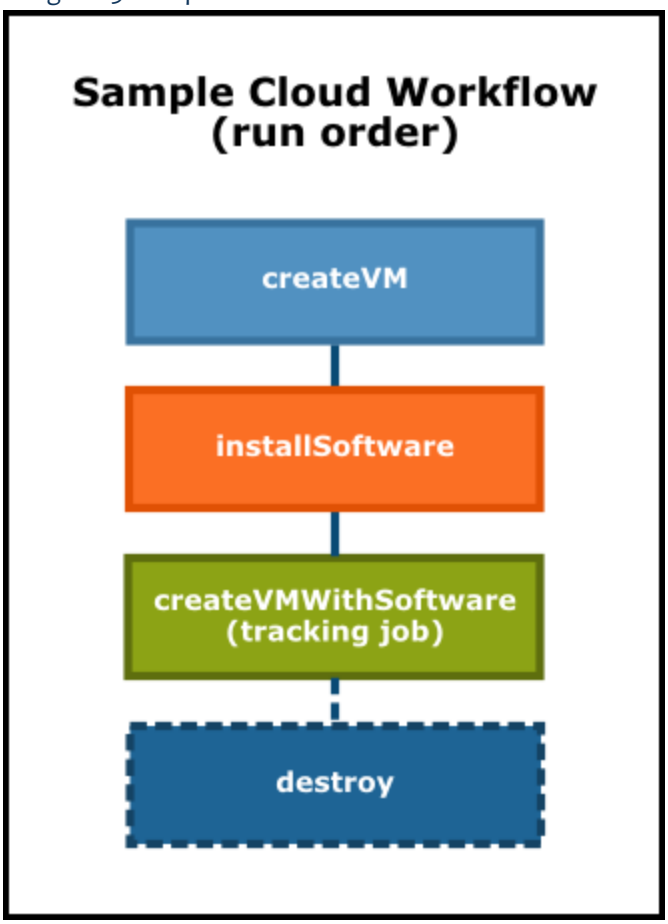


Image 22-3: Sample cloud workflow in run order



To create a generic system job workflow

1. Create a generic system job template in `moab.cfg` that will set up the VM.
   a. Give the template a unique name (*createVM*).
   b. Set the INHERITRES attribute to *TRUE*.
   c. Configure the trigger to launch a script that creates the VM (see **Generic system job trigger requirements** on page 747). Set a variable to hold the VM's IP address.

```
JOBCFG[createVM] GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/installVM.py
$HOSTLIST",Timeout=1:00:00,sets=^IPAddr
JOBCFG[createVM] INHERITRES=TRUE
```

2. Create a generic system job template to install software on the new VM.
   a. Give the template a unique name (*installSoftware*).
   b. Set the **INHERITRES** attribute to *TRUE*.

c. Specify a dependency on the `createVM` job using the TEMPLATEDEPEND attribute (see the Job Dependency Syntax table for options). In this example, the *installSoftware* job begins after the *createVM* job completes successfully.

d. Configure the trigger to launch a script that installs the software on VM, located via the IP address provided in the *IPAddr* variable.

```
JOBCFG[installSoftware]
GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/setupSoftware.py
$IPAddr",Timeout=30:00,Flags=objectxmlstdin
JOBCFG[installSoftware] INHERITRES=TRUE
JOBCFG[installSoftware] TEMPLATEDEPEND=AFTEROK:createVM
```

If you are using your workflow in Moab Web Services or Viewpoint, the trigger attached to the last setup job must pass up the *Deployed* variable upon success.

3. Create a template for the job that will act as the gate to the workflow. Do not make it a generic system job.

a. Give the template a unique name (*createVMWithSoftware*).

b. Set a dependency on the previous job. In this case, *createVMWithSoftware* should begin after *installSoftware* completes successfully.

c. Set the SELECT attribute to *TRUE* to indicate that users can request the workflow using this job (see **Applying templates based on job attributes** on page 779 for more information). The **SELECT** attribute also makes the workflow available in the Viewpoint service templates.

d. Set the *VMTRACKING* and *NORMSTART* flags. When you set the *VMTRACKING* flag on a job, Moab automatically changes the SYSTEMJOBTYPE to *vmtracking*.

e. Configure the DESTROYTEMPLATE attribute to point to the *destroy* job in the workflow. When this job is canceled, the *destroy* job runs a script to delete the service.

```
JOBCFG[createVMWithSoftware] TEMPLATEDEPEND=AFTEROK:installSoftware SELECT=TRUE
FLAGS=NORMSTART,VMTRACKING
JOBCFG[createVMWIthSoftware] DESTROYTEMPLATE=destroy
```

4. Create the *destroy* job template, specifying that it is a generic system job. Attach a trigger with a script that will take the location of the VM or other workflow service from the *IPAddr* variable and destroy the workflow.

```
JOBCFG[destroy] GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/destroy.py
$IPAddr",timeout=5:00
```

If you wish to implement an automatic VM migration policy, you must similarly configure a migration job and link to it via the MIGRATETEMPLATE attribute.

5. To request the VM, submit a job with the *createVMWithSoftware* job template and any other desired resources. You must specify a script to satisfy the msub syntax, but it will not actually run.

```
> msub -l walltime=2:00:00,template=createVMWithSoftware,nodes=1:ppn=4,mem=1024
anyScript.py
```

If ENABLEVMDESTROY is *FALSE* (default), Moab puts the VM-tracking job on hold when a VM build times out rather than deleting it. The job is visible in the showq report. Releasing the job hold causes Moab to attempt to pick up the VM again.

Moab creates a job and applies the *createVMWithSoftware* template to it. This includes the creation of the *installSoftware* and *createVM* jobs. The **INHERITRES** attribute causes the new jobs to adopt the same resource request as *createVMWithSoftware* (4 processors and 1 GB of memory). Moab then applies their own templates.

The *createVM* job runs first, the trigger script returning the IP address of the new VM and setting it as a variable. The job passes its allocation to the *installSoftware* job. *installSoftware* uses the IP address variable to locate the VM and install software on it. The job returns its resources to *createVMWithSoftware*, which is now the VM-tracking job.

22.2.4 Creating a service

Context

A VM service can contain multiple resources or subservices. For instance, the example below contains a VM and storage. To combine multiple resources in a single service, you must create a virtual container and add the services to it.

To create a virtual container and add workflows

1. Create the top layer VC using the mvcctl -c command.

```
> mvcctl -c
VC 'vc1' created
```

2. Submit the storage mount to vc1. To do so, run msub -l, requesting the storage workflow's configured submit job. You may submit multiple storage mounts if desired. You must submit a job script to satisfy the msub syntax, but Moab ignores the script.

```
> msub -l walltime=2:00:00,gres=gold:50,flags=gresonly,template=storage -W
x="vc=vc1" job.sh --xml
<Data><job JobID="Moab.4"></job><CreatedVC>vc2</CreatedVC></Data>
```

Job Moab.4 and VC vc2 have been created. Moab.4 has been placed inside of vc2, and vc2 has been placed inside of vc1.

3. Submit the VM-tracking job to vc1, setting its dependencies on the storage mount variable (*SM*) and its name space (vc2).

To request a specific name for your VM, set the VMID variable. The request will be rejected if the VM ID is not available.

```
> msub -l walltime=INFINITY,template=VMTracking,os=rhel51,depend=set:vc2.SM -W
x="vc=vc1" -W x=var=VMID=myVM" job.sh --xml
<Data><job JobID="Moab.5"></job></CreatedVC>vc4</CreatedVC></Data>
```

Job Moab.5 and VC vc4 have been created. Moab.5 has been placed inside of vc4, and vc4 has been placed inside

```
of vc1.Moab.5 depends on vc2.SM, the variable set by Moab.4's trigger.
```

22.2.5 Canceling a service

Context

A few methods exist to delete a service in Moab. If the service is a VM, you can use the mvmctl -d command and cancel any attached services (such as storage) by running canceljob on those services' tracking jobs.

For any service, you can destroy the contained objects and delete the service container using mvcctl -d as documented in the steps below, or you can delete each component of the service individually by running canceljob on the tracking jobs.

> Do not remove out-of-band VMs by deleting the placeholder reservation that Moab creates to track its resources. Instead, verify that your resource manager no longer reports the VM, then run mvmctl -d.

To cancel a service

1. When creating the service workflow, include a destroy job that will delete the service (all VMs, storage, etc.) when the main job is canceled.
2. Set the DestroyObjects flag on the service's main VC.

   ```
   > mvcctl -m flags+=DestroyObjects vc1
   ```

3. Run mvcctl -d on the VC. The *DestroyObjects* flag causes Moab to tear down the entire service, using destroy template jobs where they exist.

   ```
   > mvcctl -d vc1
   ```

   ```
   vc1 and all of its objects are destroyed.
   ```

4. Alternately, rather than doing steps 2-3, you can run canceljob on the select job of each workflow to delete the service. This will retain the main job while the destroy job runs. Once the destroy job completes successfully, the main job or VM-tracking job will cancel.

   mvmctl -d will also destroy the VM; however, Moab does not automatically delete attached resource workflows, like storage mounts. You must use canceljob on the select (tracking) job of each attached workflow to destroy the whole service.

22.3 References

22.3.1 Cloud-specific job template attributes

The table below details the job template attributes that relate specifically to workload-driven cloud services.

DESTROYTEMPLATE	
Format	*<templateName>*
Template Type	JSET
Description	When this job is canceled, Moab creates a new job and applies the specified template (must be a generic system job). The original job remains until the new job successfully completes. The job created by **DESTROYTEMPLATE** is the cancel action for the original job. By default, the destroy job runs after the tracking job; however, you can write the destroy job script so that it can be called regardless of whether the setup or tracking job ran. For a destroy job to run, its dependencies must be satisfied. You can disable dependencies by setting the **NOVMDESTROYDEPENDENCIES** on page 1140 scheduler flag.

Example	`JOBCFG[VMTracking] GENERICSYSJOB=<triggerSpecs>`
	`JOBCFG[VMTracking] DESTROYTEMPLATE=destroyVM`
	
	`JOBCFG[destroyVM] GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/destroy.py $VMID",timeout=5:00`
	
	When the VMTracking job is canceled, the destroyVM job is created. When that completes, VMTracking is removed.

GENERICSYSJOB	
Format	<triggerSpecifications>
Template Type	JSET

GENERICSYSJOB

Description	Causes the job template to create a special type of system job (a generic system job) that does the following: • runs for the duration of the trigger to which it is attached • shares an exit code with the trigger • must have one trigger attached to it that meets certain requirements (See Generic System Job Trigger Requirements). See Creating a Generic System Job for more information.
Example	`JOBCFG[test]  GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/setupSoftware.py $IPAddr",Time=5:00`

INHERITRES

Format	*<BOOLEAN>* : *TRUE* | *FALSE*
Template Type	JSET
Description	This job inherits the resource definition of the job that created it (via TEMPLATEDEPEND). The job that finishes first will pass its allocation directly to the next job.
Example	`JOBCFG[test]  INHERITRES=TRUE`

MIGRATETEMPLATE

Format	*<templateName>*
Template Type	JSET
Description	When this job is relocated, Moab creates a new job and applies the specified template (must be a generic system job). The original job remains until the new job successfully completes. The job created by **MIGRATETEMPLATE** is the migrate action for the original job.

MIGRATETEMPLATE	

Example	`JOBCFG[VMTracking] MIGRATETEMPLATE=migrateVM`
	
	`JOBCFG[migrateVM] GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/migrate.py $VMID $MASTERHOST",timeout=5:00 FLAGS=NORMSTART`
	
	When the VMTracking job is moved, the migrateVM job is created. When that completes, VMTracking migrates to the desired location.

TEMPLATEDEPEND	
Format	*<dependencyType>:<templateName>*
Template Type	JSET
Description	Specifies when the job should run based on other jobs in the workflow. See the Job Dependency Syntax table for details.
Example	`JOBCFG[test] TEMPLATEDEPEND=AFTEROK:test2`

22.3.2 Generic system job trigger requirements

A generic system job specifies one trigger that must meet the all of the following criteria:

1. The `EType` is start.
2. The `AType` is exec.
3. The `Timeout` attribute is the desired walltime of the job. Moab ignores walltime requests when you submit a generic system job, using the trigger Timeout instead.
4. The objectxmlstdin flag is set so that the job's XML, which contains the description of the VM being created, is passed to the trigger stdin for the VM creation script to access.

The trigger fires when the system job begins, and, because the trigger's Timeout doubles as the job's walltime, both complete at the same time. The job and trigger have the same completion code.

```
JOBCFG[gen] GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/installVM.py
$HOSTLIST",Timeout=1:00:00,Flags=objectxmlstdin
```

The job template gen creates a job with a walltime of 1 hour.

Sometimes the trigger will set a variable on completion or require a variable to run at all. For information about setting variables, passing them up to parent objects, and requiring variables on parent objects, see the Trigger Variables documentation.

You can attach additional triggers using the **TRIGGER** attribute and delimit them with semicolons.

```
JOBCFG[gen] GENERICSYSJOB=<genericSystemJobTriggerSpecs>
JOBCFG[gen] TRIGGER=<triggerSpecs>;TRIGGER=<triggerSpecs>;TRIGGER=<triggerSpecs>
```

The job template gen creates a job with a walltime of 1 hour.

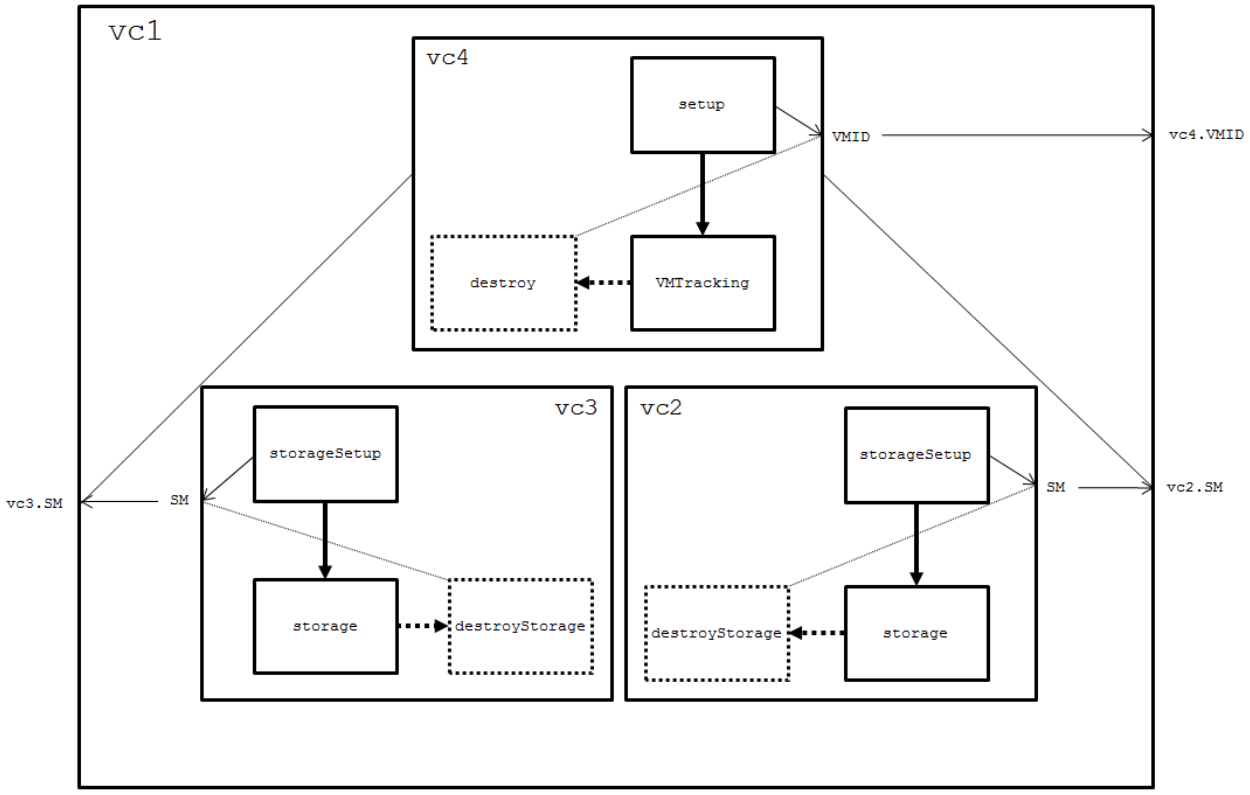
22.3.3 VM service example

This section describes how to set up a VM with two storage mounts. The scripts referenced in the code samples are examples and not shipped with Moab.

```
#The VM-tracking job
JOBCFG[VMTracking]  FLAGS=VMTRACKING SELECT=TRUE
JOBCFG[VMTracking]  TEMPLATEDEPEND=AFTEROK:VMSetup
JOBCFG[VMTracking]  DESTROYTEMPLATE=VMDestroy

JOBCFG[VMSetup]  GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/vm.py $MASTERHOST
$*.SM",flags=objectxmlstdin,timeout=5:00,sets=^VMID

JOBCFG[VMSetup]  INHERITRES=TRUE

#This is VMID, not *.VMID, because it put directly into the VMTracking workflow (same
is true for storageDestroy below)
JOBCFG[VMDestroy]  GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/destroy.py
$VMID",timeout=5:00

#Storage jobs
JOBCFG[storage]  FLAGS=NORMSTART,GRESONLY SELECT=TRUE
JOBCFG[storage]  WALLTIME=INFINITY
JOBCFG[storage]  TEMPLATEDEPEND=AFTEROK:storageSetup
JOBCFG[storage]  DESTROYTEMPLATE=storageDestroy

JOBCFG[storageSetup]
GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/storage.py",flags=objectxmlstdin,ti
meout=5:00,sets=^SM FLAGS=NORMSTART,GRESONLY
JOBCFG[storageSetup] INHERITRES=TRUE

JOBCFG[storageDestroy]
GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/removeStorage.py $SM",timeout=5:00
SELECT=TRUE
```

The `moab.cfg` configuration above contains two service workflow templates: VM and storage. Each workflow is comprised of three jobs: a setup job, a submit/tracking job, and a destroy job. The destroy jobs retrieve variables containing their respective service's location and run a script to destroy their respective workflows.

The following image illustrates how the templates above will work together when one *VMTracking* and two *storage* jobs are requested as part of a service in a single VC:



To put the workflows together like the service illustrated above, you would first need to create the top layer VC (`vc1`).

```
> mvcctl -c
VC 'vc1' created
```

You would then submit the two storage mounts by creating two of the *storage* jobs and placing them inside of `vc1`.

```
> msub -l walltime=1:00:00,gres=gold:50,flags=gresonly,template=storage -W x="vc=vc1"
job.sh --xml
vc2
> msub -l walltime=1:00:00,gres=silver:100,flags=gresonly,template=storage -W
x="vc=vc1" job.sh --xml
vc3
```

Finally, you would request the *VMTracking* job and place it into `vc1`, indicating its dependencies on the storage VCs. Since these are being pulled from the uppermost VC, their name spaces will be applied and must be specified (*vc2.SM* and *vc3.SM*).

```
> msub -l walltime=INFINITY,template=VMTracking,os=rhel51,depend=set:vc3.SM:vc2.SM -W
x="vc=vc1" -W x="trigns=vc1,vc2" -W x="var=VMID=myvm" job.sh --xml
vc4
```

23.0 Preemption

23.1 About preemption

Sites possess workloads of varying importance, and users may want to run jobs with higher priorities before jobs with lower priorities. This can be done by using preemption. Preemption is simply the process by which a higher-priority job can take the place of a lower-priority job. You can also use preemption for optimistic scheduling and development job support.

This section explains how to configure and use preemption. **Simple example of preemption** on page 771 offers a basic introduction and contains examples to help you get started using preemption. The other sections offer more explanation and information about what you can do with preemption and contain some best practices that will help you avoid the need for troubleshooting in the future.

While this section does not explain every possible preemption configuration, it does prescribe the best practices for setting up and using preemption with your system. It is recommended that you follow the established instructions contained in this section.

Preemption does not work with dynamic provisioning.

Neither *SPANEVENLY* nor *DELAY* values of the NODESETPLUS parameter will work with multi-req jobs or preemption.

Do not allow preemption with interactive jobs unless PREEMPTPOLICY is set to *CANCEL*. (For more information, see **Canceling jobs with preemption** on page 752.)

Tasks associated with preemption:

The following sections include information about each type of preemption, their different usage benefits, and any configurations and settings needed to use them.

- **Canceling jobs with preemption** on page 752
- **Checkpointing jobs with preemption** on page 755
- **Requeueing jobs with preemption** on page 757
- **Suspending jobs with preemption** on page 760
- **Using owner preemption** on page 763
- **Using QoS preemption** on page 767

Preemption references:

These sections contain information that you can use as references for the preemption tasks.

- **Manual preemption commands** on page 768
- **Preemption flags** on page 769
- **PREEMPTPOLICY types** on page 770
- **Simple example of preemption** on page 771
- **Testing and troubleshooting preemption** on page 774

Related topics

- **Optimizing Scheduling Behavior – Backfill and Node Sets** on page 433

23.2 Preemption tasks

23.2.1 Canceling jobs with preemption

Context

CANCEL is one of the **PREEMPTPOLICY** types (for more information, see **PREEMPTPOLICY types** on page 770). The *CANCEL* value cancels active jobs, regardless of any **JOBFLAGS** (such as *REQUEUEABLE* or *SUSPENDABLE*). (For more information, see **Job Flags** on page 73.)

For information about *PREEPMPTEE* and *PREEMPTOR* flags, see **Preemption flags** on page 769

> You should not allow preemption with interactive jobs unless PREEMPTPOLICY is set to *CANCEL*.

The following outlines some benefits of using *CANCEL* and also lists some things you should be aware of if you choose to use it.

Advantages:

This attribute is the easiest to configure and use.

Cautions:

Canceled jobs are not automatically restarted or requeued. Users must resubmit canceled jobs.

To preempt jobs using CANCEL

1. Make the following configurations to the `moab.cfg` file:
   a. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)

b. Make sure that **JOBNODEMATCHPOLICY** is *not* set; either comment out or remove the line from your moab.cfg file. JOBNODEMATCHPOLICY is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 774).

c. Set **PREEMPTPOLICY** to *CANCEL* (for more information, see **PREEMPTPOLICY types** on page 770).

d. Make sure that the *PREEMPTEE* job has a lower priority than the *PREEMPTOR* job (for more information, see **Preemption flags** on page 769).

For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY CANCEL

QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

2. Submit a job to the preemptee QoS (`test1`). For example:

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test1 -l procs=128
```

(Optional) Examine the following output for showq:

```
Moab.7
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.7    john        Running    128       00:01:59      Thu Nov 10 12:28:44

1 active job     128 of 128 processors in use by local jobs (100.00%)
                 2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

3. Now submit a job to the preemptor QoS (`test2`). For example:

```
[john@g06]$ echo sleep 120 | msub -l procs=128,walltime=120 -l qos=test2
```

(Optional) Examine the following output for showq:

```
Moab.8
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
```

```
Moab.7    john        Canceling  128       00:01:56      Thu Nov 10 12:28:44
Moab.8    john        Running    128       00:02:00      Thu Nov 10 12:28:48

2 active jobs 128 of 128 processors in use by local jobs (100.00%)
              2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total jobs: 2
```

Note that `test1` is canceled when `test2` is submitted.

(Optional) Examine the checkjob outputs for these two jobs:

```
[john@g06]$ checkjob Moab.9
job Moab.9

State: Removed
Completetion Code: -1 Time: Thu Nov 10 12:28:48
Creds: user:john group:john qos:test1
WallTime: 00:00:02 of 00:02:00
SubmitTime: Thu Nov 10 12:28:44
(Time Queued Total: 00:00:07 Eligible: 00:00:00)

Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses
NodeCount: 2

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.UFe8sQ

StartCount: 1
Flags: GLOBALQUEUE,PROCSPECIFIED
Attr: PREEMPTEE
StartPriority: 100
```

Note that the preempted job has been removed.

```
[john@g06]$ checkjob Moab.10
job Moab.10

State: Running
Creds: user:john group:john qos:test2
WallTime: 00:00:00 of 00:02:00
SubmitTime: Thu Nov 10 12:36:31
(Time Queued Total: 00:00:00 Eligible: 00:00:00)
```

```
StartTime: Thu Nov 10 12:28:48
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.CZavjU

StartCount: 1
Flags: HASPREEMPTED,PREEMPTOR,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 10000
Reservation 'Moab.10' (-00:00:07 -> 00:01:53 Duration: 00:02:00)
```

Related topics

- **Suspending jobs with preemption** on page 760
- **Checkpointing jobs with preemption** on page 755
- **Requeueing jobs with preemption** on page 757
- **Preemption flags** on page 769
- **About preemption** on page 751
- **PREEMPTPOLICY types** on page 770
- **Testing and troubleshooting preemption** on page 774

23.2.2 Checkpointing jobs with preemption

Context

CHECKPOINT is one of the **PREEMPTPOLICY** types (for more information, see **PREEMPTPOLICY types** on page 770). For systems that allow checkpointing, the *CHECKPOINT* value allows a job to save its current state and either terminate or continue running. A checkpointed job may restart at any time and resume execution from its most recent checkpoint.

You can tune checkpointing behavior on a per-resource manager-basis by setting the CHECKPOINTSIG and CHECKPOINTTIMEOUT attributes of the **RMCFG** parameter.

For information about *PREEPMPTEE* and *PREEMPTOR* flags, see **Preemption flags** on page 769

The following outlines some benefits of using *CHECKPOINT* and also lists some things you should be aware of if you choose to use it.

Advantages:

This attribute allows you to restart a job from its last checkpoint.

Cautions:

Jobs tend to take longer to complete when you use *CHECKPOINT*.

To preempt jobs using CHECKPOINT

Make the following configurations to the `moab.cfg` file:

1. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)
2. Make sure that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*, which is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 774).
3. Set **PREEMPTPOLICY** to *CHECKPOINT* (for more information, see **PREEMPTPOLICY types** on page 770).
4. Make sure that the *PREEMPTEE* job has a lower priority than the *PREEMPTOR* job (for more information, see **Preemption flags** on page 769).

For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY CHECKPOINT

QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

Related topics

- **Suspending jobs with preemption** on page 760
- **Requeueing jobs with preemption** on page 757
- **Canceling jobs with preemption** on page 752
- **Preemption flags** on page 769
- **About preemption** on page 751
- **PREEMPTPOLICY types** on page 770
- **Testing and troubleshooting preemption** on page 774

23.2.3 Requeueing jobs with preemption

Context

REQUEUE is one of the **PREEMPTPOLICY** types (for more information, see **PREEMPTPOLICY types** on page 770). The *REQUEUE* value terminates active jobs and returns them to the job queue in an idle state.

For information about *PREEPMPTEE* and *PREEMPTOR* flags, see **Preemption flags** on page 769

The following outlines some benefits of using *REQUEUE* and also lists some things you should be aware of if you choose to use it.

Advantages:

- Jobs are automatically resubmitted into the job queue.

Cautions:

- A job gets resubmitted in the job queue at the same priority it had when Moab originally started it (i.e., the job does not jump ahead in the queue).
- Jobs start over from the beginning.

You must mark a job as *RESTARTABLE* if you want it to requeue. If you do not, the job will be canceled when it is preempted.

If supported by the resource manager, you can set the *RESTARTABLE* job flag when submitting the job by using the msub -r option. Otherwise, use the **JOBFLAGS** attribute of the associated class or QoS credential, as in this example:

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

For more information, see **Job Flags** on page 73.

To preempt jobs using REQUEUE

1. Make the following configurations to the `moab.cfg` file:
   a. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)
   b. Make sure that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*, which is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 774).
   c. Set **PREEMPTPOLICY** to *REQUEUE* (for more information, see **PREEMPTPOLICY types** on page 770).
   d. Make sure that the *PREEMPTEE* job has a lower priority than the *PREEMPTOR* job (for more information, see **Preemption flags** on page 769).

   For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY REQUEUE

QOSCFG[test1] QFLAGS=PREEMPTEE JOBFLAGS=RESTARTABLE MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

2. Submit a job to the preemptee QoS (`test1`). For example:

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test1 -l procs=128
```

(Optional) Examine the following output for showq:

```
Moab.1
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING      STARTTIME
Moab.1    john        Running    128       00:09:59       Wed Nov 9 15:56:33

1 active job     128 of 128 processors in use by local jobs (100.00%)
                 2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

3. Now submit a job to the preemptor QoS (`test2`). For example:

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test2 -l procs=128
```

(Optional) Examine the following output for showq and checkjob:

```
Moab.2
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME     STATE      PROCS    REMAINING     STARTTIME
Moab.2    john         Running    128      00:09:59      Wed Nov 9 15:56:47

1 active job 128 of 128 processors in use by local jobs (100.00%)
             2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE       PROCS    WCLIMIT       QUEUETIME
Moab.1    john        Idle        128      00:10:00      Wed Nov 9 15:56:33

1 eligible job

blocked jobs-----------------------
JOBID     USERNAME    STATE       PROCS    WCLIMIT       QUEUETIME
```

```
0 blocked jobs

Total jobs: 2
```

```
[john@g06]# checkjob Moab.2
job Moab.2

State: Running
Creds: user:john group:john qos:test2
WallTime: 00:02:04 of 00:10:00
SubmitTime: Wed Nov 9 15:56:46
(Time Queued Total: 00:00:01 Eligible: 00:00:00)

StartTime: Wed Nov 9 15:56:47
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses
NodeCount: 2

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.ELoX5Q

StartCount: 1
Flags: HASPREEMPTED,PREEMPTOR,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 10000
Reservation 'Moab.2' (-00:02:21 -> 00:07:39 Duration: 00:10:00)
```

Related topics

- **Suspending jobs with preemption** on page 760
- **Checkpointing jobs with preemption** on page 755
- **Canceling jobs with preemption** on page 752
- **Preemption flags** on page 769
- **About preemption** on page 751
- **PREEMPTPOLICY types** on page 770
- **Testing and troubleshooting preemption** on page 774

23.2.4 Suspending jobs with preemption

Context

SUSPEND is one of the **PREEMPTPOLICY** types (for more information, see **PREEMPTPOLICY types** on page 770). The *SUSPEND* attribute causes active jobs to stop executing, but to remain in memory on the allocated compute nodes.

For information about *PREEPMPTEE* and *PREEMPTOR* flags, see **Preemption flags** on page 769

The following outlines some benefits of using *SUSPEND*, and also lists some things you should be aware of if you choose to use it.

Advantages:

- The job remains in memory on the allocated compute nodes.
- Using *SUSPEND* frees up processor resources.
- The job can restart where it left off before it was suspended.

Cautions:

- There is a possibility that having multiple suspended jobs on a compute node will crash the swap.
- Moab tracks only *requested* memory of active jobs (not *used* memory). The swap can crash if the job uses a lot of memory and Moab starts other jobs.
- Suspended jobs do not relinquish their licenses.

You must mark a job as *SUSPENDABLE* if you want it to suspend. If you do not, the job will be requeued or canceled when it is preempted.

If supported by the resource manager, you can set the job *SUSPENDABLE* flag when submitting the job by using the msub -r option. Otherwise, use the **JOBFLAGS** attribute of the associated class or QoS credential, as in this example:

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

For more information, see **Job Flags** on page 73.

To preempt jobs using SUSPEND

When you use *SUSPEND*, you must increase your **JOBRETRYTIME**. By default, JOBRETRYTIME is set to 60 seconds, but when you use *SUSPEND*, it is recommended that you increase the time to 300 seconds (5 minutes).

1. Make the following configurations to the `moab.cfg` file:
   a. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)

b. Make sure that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*, which is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 774).

c. Set **PREEMPTPOLICY** to *SUSPEND* (for more information, see **PREEMPTPOLICY types** on page 770).

d. For the *PREEMPTEE* job, set **JOBFLAGS**=*RESTARTABLE,SUSPENDABLE*.

e. Make sure that the *PREEMPTEE* job has a lower priority than the *PREEMPTOR* job (for more information, see **Preemption flags** on page 769).

For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY SUSPEND

QOSCFG[test1] QFLAGS=PREEMPTEE JOBFLAGS=RESTARTABLE,SUSPENDABLE MEMBERULIST=john
PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

2. Submit a job to the preemptee QoS (`test1`). For example:

```
[john@g06]$ echo sleep 120 | msub -l procs=128,walltime=120 -l qos=test1
```

(Optional) Examine the output for showq:

```
Moab.7
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.7    john        Running    128       00:01:59      Thu Nov 10 12:28:44

1 active job     128 of 128 processors in use by local jobs (100.00%)
                 2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

3. Now submit a job to the preemptor QoS (`test2`). For example:

```
[john@g06]$ echo sleep 120 | msub -l procs=128,walltime=120 -l qos=test2
```

(Optional) Examine the output for showq:

```
Moab.8
[john@g06]# showq
```

```
active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.7    john        Suspended  128       00:01:56      Thu Nov 10 12:28:44
Moab.8    john        Running    128       00:02:00      Thu Nov 10 12:28:48

2 active jobs 128 of 128 processors in use by local jobs (100.00%)
              2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total jobs: 2
```

Note that when a job is suspended, it stays in the output of showq. This is normal behavior for a suspended job. Moab should only suspend a job once.

4. (Optional) Examine the checkjob outputs for these two jobs.

```
[john@g06]$ checkjob Moab.9
job Moab.9

State: Suspended
Creds: user:john group:john qos:test1
WallTime: 00:00:02 of 00:02:00
SubmitTime: Thu Nov 10 12:36:29
(Time Queued Total: 00:00:07 Eligible: 00:00:00)

Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses
NodeCount: 2

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.UFe8sQ

StartCount: 1
Flags: RESTARTABLE,SUSPENDABLE,PREEMPTEE,GLOBALQUEUE,PROCSPECIFIED
Attr: PREEMPTEE
StartPriority: 100
job cannot be resumed: preemption required but job is conditional preemptor with no
targets
BLOCK MSG: non-idle state 'Running' (recorded at last scheduling iteration)
```

```
[john@g06]$ checkjob Moab.10
job Moab.10

State: Running
```

```
Creds: user:john group:john qos:test2
WallTime: 00:00:00 of 00:02:00
SubmitTime: Thu Nov 10 12:36:31
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Thu Nov 10 12:36:31
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.CZavjU

StartCount: 1
Flags: HASPREEMPTED,PREEMPTOR,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 10000
Reservation 'Moab.10' (-00:00:07 -> 00:01:53 Duration: 00:02:00)
```

Occasionally, Moab will keep a job from restarting, holding it in a suspended state for a long period of time, if it thinks the job cannot restart. For example, if a job could write to I/O before it was suspended, and now it cannot, Moab would realize the job is unable to start and would leave it in a suspended state.

Related topics

- **Checkpointing jobs with preemption** on page 755
- **Requeueing jobs with preemption** on page 757
- **Canceling jobs with preemption** on page 752
- **Preemption flags** on page 769
- **About preemption** on page 751
- **PREEMPTPOLICY types** on page 770
- **Testing and troubleshooting preemption** on page 774

23.2.5 Using owner preemption

Context

Owner preemption allows jobs submitted by a reservation owner to preempt jobs submitted by other users (for more information, see **Configuring and Managing Reservations** on page 386).

Owner preemption is enabled with the **OWNERPREEMPT** reservation flag.

For information about *PREEPMPTEE* and *PREEMPTOR* flags, see **Preemption flags** on page 769

To enable owner preemption

1. Make the following configurations to the `moab.cfg` file:

   a. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)

   b. Make sure that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*, which is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 774).

   c. Set the **PREEMPTPOLICY** type (for more information, see **PREEMPTPOLICY types** on page 770).

   d. Set the *OWNERPREEMPT* flag.

   > Optionally, if you want the owner preemption to override any PREEMPTMINTIME settings for *PREEMPTEE* jobs, you can set the **OWNERPREEMPTIGNOREMINTIME** flag as well.

   e. Specify an owner.

   > If the non-owner job does not have a *RESTARTABLE* or *REQUEUEABLE* flag set, the job will cancel.

   For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY <policy>

SRCFG[myrez] FLAGS=OWNERPREEMPT HOSTLIST=node01
SRCFG[myrez] OWNER=USER:john
SRCFG[myrez] USERLIST=jane,john PERIOD=INFINITY

QOSCFG[test1] QFLAGS=PREEMPTEE JOBFLAGS=restartable MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

2. Submit a job to a user who is not the owner (in this example, `jane`).

```
[jane@g06]$ echo sleep 600 | msub -l walltime=600 -l procs=64
```

   (Optional) Examine the following output for showq and checkjob for `jane`'s job:

```
Moab.1
[jane@g06]$ showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS      REMAINING     STARTTIME
Moab.1    jane        Running    64         00:09:57      Mon Nov 14 12:07:52

1 active job     64 of 64 processors in use by local jobs (100.00%)
                 1 of 1 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME

0 eligible jobs
```

```
blocked jobs------------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

```
root@g06]# checkjob Moab.1
job Moab.1

State: Running
Creds: user:jane group:jane
WallTime: 00:01:02 of 00:10:00
SubmitTime: Mon Nov 14 12:07:52
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Mon Nov 14 12:07:52
Total Requested Tasks: 64

Req[0] TaskCount: 64 Partition: FLEXlm
NodeCount: 1

Allocated Nodes:
[node01:64]


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.FoZfIU

StartCount: 1
Flags: GLOBALQUEUE,PROCSPECIFIED
StartPriority: 1
Reservation 'Moab.1' (-00:01:24 -> 00:08:36 Duration: 00:10:00)
```

3. Now submit a job for the owner (in this example, `john`).

```
[john@g06]$ echo sleep 600 | msub -l walltime=600 -l procs=50
```

```
[john@g06]$ echo sleep 600 | msub -l walltime=600 -l procs=50
```

(Optional) Examine the following output for showq and checkjob for `john`'s job:

```
Moab.2
[john@g06]$ showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.1    jane        Canceling  64        00:07:43      Mon Nov 14 12:07:52
Moab.2    john        Running    50        00:09:59      Mon Nov 14 12:10:08

2 active jobs    64 of 64 processors in use by local jobs (100.00%)
                 1 of 1 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs
```

```
blocked jobs------------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total jobs: 2
```

Note that `jane`'s job is canceled once `john`'s job is submitted.

```
[john@g06]$ checkjob Moab.2
job Moab.2

State: Running
Creds: user:john group:john
WallTime: 00:00:31 of 00:10:00
SubmitTime: Mon Nov 14 12:10:08
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Mon Nov 14 12:10:08
Total Requested Tasks: 50

Req[0] TaskCount: 50 Partition: FLEXlm
NodeCount: 1

Allocated Nodes:
[node01:50]


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.jf1N4a

StartCount: 1
Flags: HASPREEMPTED,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 1
Reservation 'Moab.2' (-00:00:48 -> 00:09:12 Duration: 00:10:00)
```

Note the new HASPREEMPTED flag.

(Optional) Now look at the showq for `jane`'s job (after):

```
[root@g06]# checkjob Moab.1
job Moab.1

State: Removed
Completion Code: -1 Time: Mon Nov 14 12:10:08
Creds: user:jane group:jane
WallTime: 00:02:47 of 00:10:00
SubmitTime: Mon Nov 14 12:07:52
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

Total Requested Tasks: 64

Req[0] TaskCount: 64 Partition: FLEXlm
NodeCount: 1

Allocated Nodes:
[node01:64]
```

```
IWD: /opt/native
Executable: /opt/native/spool/moab.job.FoZfIU

Execution Partition: FLEXlm
Flags: GLOBALQUEUE,PROCSPECIFIED
StartPriority: 0
```

Note that the state is now Removed.

Related topics

- **Preemption flags** on page 769
- **About preemption** on page 751
- **PREEMPTPOLICY types** on page 770
- **Testing and troubleshooting preemption** on page 774

23.2.6 Using QoS preemption

Context

This section breaks down how to configure the `moab.cfg` file to set up preemption with QoS. Using QoS, you can specify preemption rules and control access to preemption privileges by using the QFLAGS *PREEMPTEE* and *PREEMPTOR* credentials. For information about the *PREEMPTEE* and *PREEMPTOR* flags, see **Preemption flags** on page 769.

QoS-based preemption only occurs when the following three conditions are satisfied:

- The preemptor job has the *PREEMPTOR* value set.
- The preemptee job has the *PREEMPTEE* value set.
- The preemptor job has a higher priority than the preemptee job.

To configure moab.cfg for QoS preemption

1. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)
2. Make sure that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*, which is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 774).
3. If it is not already, set **NODEACCESSPOLICY** to *SHARED*.
4. Set the **PREEMPTPOLICY** policy type (for more information, see **PREEMPTPOLICY types** on page 770).
5. Set up **QFLAGS** to mark jobs as *PREEMPTEE* (a lower-priority job that can be preempted by a higher-priority job), or as *PREEMPTOR* (a higher-priority job that can preempt a lower-priority job). As in the example:

```
QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=<user> PRIORITY=100
```

```
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=<user> PRIORITY=10000
```

For more information, see **Preemption flags** on page 769.

6. Make sure that the *PREEMPTEE* job has a lower priority than the *PREEMPTOR* job. As in the example:

```
QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=<user> PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=<user> PRIORITY=10000
```

For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY <policy>

QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=<user> PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=<user> PRIORITY=10000
```

Related topics

- **About preemption** on page 751
- **Preemption flags** on page 769
- **PREEMPTPOLICY types** on page 770
- **Simple example of preemption** on page 771
- **Testing and troubleshooting preemption** on page 774

23.3 Preemption references

23.3.1 Manual preemption commands

You can use the **mjobctl** command to manually preempt jobs. The command can modify a job's execution state in the following ways:

Action	Flag	Details
Cancel	-c	Terminate job; remove from queue
Checkpoint	-C	Terminate and checkpoint job leaving job in queue
Requeue	-R	Terminate job; leave in queue
Resume	-r	Resume suspended job
Start (execute)	-x	Start idle job

Action	Flag	Details
Suspend	-s	Suspend active job

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

Related topics

- **About preemption** on page 751
- **Testing and troubleshooting preemption** on page 774

23.3.2 Preemption flags

Using QoS, you can specify preemption rules and control access to preemption privileges. This allows you to increase system throughput, improve job response time for specific classes of jobs, or enable various political policies. You enable all policies by specifying some QoS credentials with the QFLAGS *PREEMPTEE*, and others with *PREEMPTOR*.

PREEMPTEE	
Description	Indicates that the job can be preempted by a higher-priority job.
Use	Use for lower-priority jobs that can be preempted.
Notes	This may delay some node actions. When reprovisioning, the system job may expire before the provision action occurs; while the action will still occur, the job will not show it.
Example	`QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=<user> PRIORITY=100`

PREEMPTOR	
Description	Indicates that the job should take priority and preempt any *PREEMPTEE* jobs.
Use	Use for jobs that need to take precedence over lower-priority jobs.
Notes	*PREEMPTOR* jobs, either queued or running, must have a higher priority than *PREEMPTEE* jobs. When you configure job as a *PREEMPTOR*, you should also increase its priority (for details, see **PREEMPTPRIOJOBSELECTWEIGHT** and **PREEMPTRTIMEWEIGHT**).

PREEMPTOR	
Example	`QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=<user> PRIORITY=10000`

Additional preemptor and preemptee information

Preemptor priority plays a big role in preemption. Generally, you should assign the preemptor job a higher priority value than any other queued jobs so that it will move to (or near to) the top of the eligible queue.

You can set the **RESERVATIONPOLICY** parameter to *NEVER*. With this configuration, preemptee jobs can start whenever idle resources become available. These jobs will be allowed to run until a preemptor job arrives, at which point the preemptee jobs are preempted, freeing the resource. This configuration allows near immediate resource access for the preemptor jobs. Using this approach, a cluster can maintain near 100% system utilization while still delivering excellent turnaround time to the most important jobs.

In environments where job checkpointing or job suspension incur significant overhead, you might want to constrain the rate at which job preemption is allowed. You can use the **JOBPREEMPTMINACTIVETIME** parameter to throttle job preemption. In essence, this parameter prevents a newly started or newly resumed job from being eligible for preemption until it has executed for a specified amount of time. Conversely, you can exclude jobs from preemption after they have run for a certain amount of time by using the **JOBPREEMPTMAXACTIVETIME** parameter.

Related topics

- **About preemption** on page 751
- **Using QoS preemption** on page 767
- **Manual preemption commands** on page 768
- **PREEMPTPOLICY types** on page 770
- **Testing and troubleshooting preemption** on page 774

23.3.3 PREEMPTPOLICY types

You can use the **PREEMPTPOLICY** parameter to control how the scheduler preempts a job. This parameter enforces preemption using one of the following methods:

PREEMPTPOLICY type	Description
SUSPEND	Causes active jobs to stop executing, but to remain in memory on the allocated compute nodes.

PREEMPTPOLICY type	Description
CHECKPOINT	Saves the current job state and either terminates or continues running the job. A checkpointed job may restart at any time and resume execution from its most recent checkpoint.
REQUEUE	Terminates active jobs and returns them to the job queue in an idle state.
CANCEL	Cancels active jobs.

Each of these methods varies in the level of disruption to the job, *SUSPEND* being the least disruptive and *CANCEL* being the most disruptive.

Moab uses preemption escalation to free up resources. So for example, if the PREEMPTPOLICY is set to *SUSPEND*, Moab uses this method if it is available; however, Moab will escalate it to something potentially more disruptive if necessary to preempt and free up resources.

Related topics

- **Suspending jobs with preemption** on page 760
- **Checkpointing jobs with preemption** on page 755
- **Requeueing jobs with preemption** on page 757
- **Canceling jobs with preemption** on page 752
- **About preemption** on page 751
- **Preemption flags** on page 769

23.3.4 Simple example of preemption

This section illustrates the process of setting up preemption on your system from beginning to end and contains examples of what actions to take and what you should see as you go.

Example scenario

For this basic setup example, we will have a user who can submit to either a "test1" or "test2" QoS. This example will use a *REQUEUE* preemption type.

We will go through three parts to set up this preemption:

- Configuring the `moab.cfg` file
- Submitting a job to the *PREEMPTEE* QoS
- Submitting a job to the *PREEMPTOR* QoS

Okay, let's get started!

Configuring moab.cfg

First, you will need to make some configurations to the `moab.cfg` file.

1. Set **GUARANTEEDPREEMPTION** to *TRUE*. (This causes Moab to lock *PREEMPTOR* jobs until JOBRETRYTIME expires.)
2. Make sure that **JOBNODEMATCHPOLICY** is *not* set to *EXACTNODE*, which is not currently supported for preemption (for more information, see **Testing and troubleshooting preemption** on page 774).
3. Set the **PREEMPTPOLICY** type. In this example, **PREEMPTPOLICY** is set to *REQUEUE*. For more information, see **PREEMPTPOLICY types** on page 770.
4. Set up QFLAGS to mark jobs as *PREEMPTEE* (a lower-priority job that can be preempted by a higher-priority job), or as *PREEMPTOR* (a higher-priority job that can preempt a lower-priority job). For more information, see **Preemption flags** on page 769.

   For this example, we also set **JOBFLAGS**=*RESTARTABLE* (because this example uses *REQUEUE*). For more information, see **Requeueing jobs with preemption** on page 757.

5. Make sure that the *PREEMPTEE* job has a lower priority than the *PREEMPTOR* job.

Here is an example of how that would all look in a `moab.cfg` file (text marked red for emphasis).

```
GUARANTEEDPREEMPTION TRUE
#should not be JOBNODEMATCHPOLICY EXACTNODE as it causes problems when starting jobs

PREEMPTPOLICY REQUEUE

QOSCFG[test1] QFLAGS=PREEMPTEE JOBFLAGS=RESTARTABLE MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=1000
```

Now you can submit a job to the preemptee QoS (`test1`).

Submitting a job to the preemptee QoS

Let's submit a job to the preemptee QoS (`test1`), requesting all processor cores in the cluster:

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test1 -l procs=128
```

Take a look at the showq and checkjob output:

```
Moab.1
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.1    john        Running    128       00:09:59      Wed Nov 9 15:56:33

1 active job     128 of 128 processors in use by local jobs (100.00%)
                 2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs
```

```
blocked jobs------------------------
JOBID     USERNAME    STATE     PROCS       WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

```
[john@g06]# checkjob Moab.1
job Moab.1

State: Running
Creds: user:john group:john qos:test1
WallTime: 00:00:00 of 00:10:00
SubmitTime: Wed Nov 9 15:56:33
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Wed Nov 9 15:56:33
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses

Allocated Nodes:
node[01-02]*64

IWD: /opt/native/
SubmitDir: /opt/native/
Executable: /opt/native/spool/moab.job.zOyf1N

StartCount: 1
Flags: RESTARTABLE,PREEMPTEE,GLOBALQUEUE,PROCSPECIFIED
Attr: PREEMPTEE
StartPriority: 100
Reservation 'Moab.1' (-00:00:03 -> 00:09:57 Duration: 00:10:00
```

Submitting a job to the preemptor QoS

Now we will submit a preemptor QoS job (`test2`) to preempt the first job (`test1`):

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test2 -l procs=128
```

Examine the following output for showq and checkjob:

```
Moab.2
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS      REMAINING     STARTTIME
Moab.2    john        Running    128        00:09:59      Wed Nov 9 15:56:47

1 active job 128 of 128 processors in use by local jobs (100.00%)
             2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME
Moab.1    john        Idle       128        00:10:00      Wed Nov 9 15:56:33

1 eligible job

blocked jobs-----------------------
```

```
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total jobs: 2
```

Note that the preemptor job (Moab.2) moved to Running, while the preemptee job (Moab.1) was requeued.

```
[john@g06]# checkjob Moab.2
job Moab.2

State: Running
Creds: user:john group:john qos:test2
WallTime: 00:02:04 of 00:10:00
SubmitTime: Wed Nov 9 15:56:46
(Time Queued Total: 00:00:01 Eligible: 00:00:00)

StartTime: Wed Nov 9 15:56:47
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses
NodeCount: 2

Allocated Nodes:
node[01-02]*64


IWD: /opt/native/
SubmitDir: /opt/native/
Executable: /opt/native/spool/moab.job.ELoX5Q

StartCount: 1
Flags: HASPREEMPTED,PREEMPTOR,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 10000
Reservation 'Moab.2' (-00:02:21 -> 00:07:39 Duration: 00:10:00)
```

Note the flag, HASPREEMPTED. HASPREEMPTED is set when the PREEMPTOR job has preempted the PREEMPTEE job. Also note that the preemptor job priority plays a very big role in preemption. Generally, you should assign the preemptor a higher priority than any other queued jobs so that it will move to (or near to) the top of the eligible queue.

Related topics

- **About preemption** on page 751
- **Preemption flags** on page 769
- **PREEMPTPOLICY types** on page 770
- **Manual preemption commands** on page 768
- **Testing and troubleshooting preemption** on page 774

23.3.5 Testing and troubleshooting preemption

There are multiple steps associated with setting up a working preemption policy. With preemption, issues arise because it appears that Moab is not allowing preemptor jobs to preempt preemptee jobs in the right way. To diagnose this, use the following checklist:

	Verify that preemptor jobs are marked with the *PREEMPTOR* flag. (Verify with checkjob <JOBID> | grep Flags.)
	Verify that preemptee jobs are marked with the *PREEMPTEE* flag. (Verify with checkjob <JOBID> | grep Flags.)
	Verify that the start priority of the preemptor job is higher than the priority of the preemptee job. (Verify with checkjob <JOBID> | grep Priority.)
	Verify that the resources allocated to the preemptee job match those requested by the preemptor job.
	Verify that the preemptor job is within the 32-preemptee limit.
	Verify that there are no policies preventing preemption from occurring. (Verify with checkjob -v -n <NODEID> <JOBID>.)
	Verify that the **PREEMPTPOLICY** parameter is properly set. (See **PREEMPTPOLICY types** on page 770.)
	Verify that the preemptee job is properly marked as restartable, suspendable, or checkpointable. (Verify with checkjob <JOBID> | grep Flags.)
	Verify that **GUARANTEEDPREEMPTION** is set to *TRUE*.
	Verify that **JOBNODEMATCHPOLICY** is *not* set; either comment out or remove the line from your moab.cfg file when enabling preemption. Moab does not currently consider JOBNODEMATCHPOLICY when it handles preemption, resulting in unexpected behavior when it is set in an environment with preemption.
	Verify that **NODEACCESSPOLICY** is *not* set to *SINGLEUSER*. (*SHARED* is recommended.)
	Verify that **BACKFILLPOLICY** is set to *FIRSTFIT*.
	Verify that the resource manager is properly responding to preemption requests. (Use **mdiag -R**.)
	If there is a resource manager level race condition, verify that Moab is properly holding target resources. (Verify with **mdiag -S** and set **RESERVATIONRETRYTIME** if needed.)

Related topics

- **About preemption** on page 751
- **Quality of Service (QoS) Facilities** on page 423
- **Managing QoS Access** on page 430
- **JOBMAXPREEMPTPERITERATION** on page 854
- **Trigger components** on page 674
- **Checkpoint/Restart Facilities** on page 452

- **ENABLEFSVIOLATIONPREEMPTION** on page 823
- **PREEMPTPRIOJOBSELECTWEIGHT** on page 890
- **PREEMPTSEARCHDEPTH** on page 891
- **USAGEEXECUTIONTIMEWEIGHT** on page 935 (control priority of suspended jobs)
- **IGNOREPREEMPTEEPRIORITY** on page 846 (relative job priority is ignored in preemption decisions)
- **DISABLESAMECREDPREEMPTION** on page 819 (jobs cannot preempt other jobs with the same credential)
- **PREEMPTRTIMEWEIGHT** on page 890 (add remaining time of jobs to preemption calculation)

24.0 Job templates

24.1 About job templates

A Moab job template is a set of pre-configured settings, attributes, and resources that Moab applies to jobs that match certain criteria or to which you manually apply it. They perform three primary functions:

1. They generically match and categorize jobs.
2. They set arbitrary default or forced attributes for certain jobs.
3. They generate workflows that create and maintain user-requested services in a cloud environment. For more information about creating cloud services, see **About workload-driven cloud services** on page 733.

You can use job templates in many aspects of scheduling, including cloud environments. Job templates are defined using the **JOBCFG** on page 849 configuration parameter.

Two methods exist for applying job templates to jobs. You can use the **JOBMATCHCFG** on page 852 parameter to mark a template that contains the criteria a job must meet for eligibility and another template as the one to be applied to the job if it is eligible. This allows you to automate the use of templates. For example, to force all interactive jobs to run on a certain set of nodes, you can set one template (the criteria template) to have the *interactive* flag, then give the other template the desired host list. You can also apply a template directly to a job at submission if that ability is enabled for that template.

Job template how-to's

- **Creating job templates** on page 778
- **Viewing job templates** on page 779
- **Applying templates based on job attributes** on page 779
- **Requesting job templates directly** on page 780
- **Creating workflows with job templates** on page 781

Job template references

- **Job template extension attributes** on page 782
- **Job template matching attributes** on page 793
- **Job template examples** on page 794
- **Job template workflow examples** on page 795

24.2 Job template how-to's

24.2.1 Creating job templates

Context

Job templates are created in the Moab configure file using the **JOBCFG** on page 849 parameter.

To create a job template

1. Open `moab.cfg`. Add the JOBCFG parameter and give the new job template a unique name.

```
JOBCFG[newtemplate]
```

2. Configure any desired attributes (see **Job template extension attributes** on page 782.). Some of the important attributes include:

   - **FLAGS** on page 784 - Lets you specify any job flags that should be applied.

   ```
   JOBCFG[newtemplate] FLAGS=SUSPENDABLE
   ```

   When Moab applies `newtemplate` to a job, the job is marked as suspendable.

   - **SELECT** on page 790 - Lets you apply the template directly at job submission.

   ```
   JOBCFG[newtemplate] FLAGS=SUSPENDABLE SELECT=TRUE
   ```

   When you submit a job via msub, you can specify that your job has `newtemplate` applied to it. When Moab applies the template to a job, that job is marked as suspendable.

   - **TEMPLATEDEPEND** on page 791 - Lets you create dependencies when you create a job template workflow (see **Creating workflows with job templates** on page 781.).

   ```
   JOBCFG[newtemplate] FLAGS=SUSPENDABLE SELECT=TRUE TEMPLATEDEPEND=AFTER:job1.pre
   ```

   When Moab applies `newtemplate` to a job, the job cannot run until job `job1.pre` has finished running; the job is also marked as suspendable. You can specify that Moab apply this template to a job as you submit it.

3. If you want to automate job template application, see **Applying templates based on job attributes** on page 779 for instructions. If you want to apply the template manually on job submission, see **Requesting job templates directly** on page 780 for instructions.

Related topics

- **Job template extension attributes** on page 782
- **Job template examples** on page 794

24.2.2 Viewing job templates

Context

You can view a job template by running the mdiag -j command.

To view a job template

- Run the mdiag -j command with the *policy* flag. Moab returns a list of job templates configured in `moab.cfg`.

```
> mdiag -j --flags=policy --blocking
```

24.2.3 Applying templates based on job attributes

Context

The **JOBMATCHCFG** on page 852 parameter allows you to establish relationships between a number of job templates. JMAX and JMIN function as filters to determine whether a job is eligible for a subsequent template to be applied to the job. If a job is eligible, JDEF and JSET templates apply attributes to the job. See **Job template extension attributes** on page 782 for more information about the JOBMATCHCFG attributes. The table on that page indicates which job template types are compatible with which job template extension attributes.

JSETs and **JDEF**s have only been tested using msub as the job submission command.

To apply a job template based on job attributes

1. In the Moab configuration file, create a job template with a set of criteria that a job must meet in order for Moab to apply the template. In the following example, Moab will apply a template to all interactive jobs, so the first template sets the *interactive* flag.

```
JOBCFG[inter.min] FLAGS=interactive
```

2. Create the job template that Moab should apply to the job if it meets the requirements set in the first template. In this example, Moab ignores all configured policies, so the second template sets the *ignpolicies* flag.

```
JOBCFG[inter.set] FLAGS=ignpolicies
```

3. Use the JOBMATCHCFG parameter and its **JMAX** or **JMIN** (specify the template specifying maximum or minimum requirements) and **JDEF** or **JSET** (specify the template to be applied) attributes to demonstrate the relationship between the two templates (See **Job template matching attributes** on page 793 for more information.). In this case, all interactive jobs ignore policies; in other words, if a submitted job has at least the *inter.min* template settings, Moab applies the *inter.set* template settings to the job.

```
JOBMATCHCFG[interactive] JMIN=inter.min JSET=inter.set
```

Moab applies the inter.set template to all jobs with the interactive flag set, causing them to ignore Moab's configured policies.

4. To control which job template is applied to a job that matches multiple templates, use **FLAGS**=*BREAK*. Job templates are processed in the order they are listed in the configuration file and using the *BREAK* flag causes Moab to stop evaluating **JOBMATCHCFG** entries that occur after the current match.

```
JOBMATCHCFG[small] JMIN=small.min JMAX=small.max JSET.set=small.set FLAGS=BREAK
JOBMATCHCFG[large] JMIN=large.min JMAX=large.max JSET=large.set
```

*In this case, the large template would not be applied when a job matches both the small and large templates. The small template matches first, and because of **FLAGS**=BREAK, Moab stops evaluating further JOBMATCHFG entries for the job.*

Related topics

- **Requesting job templates directly** on page 780
- **Job template examples** on page 794

24.2.4 Requesting job templates directly

Context

When a job template has its **SELECT** on page 790 attribute set to *TRUE*, you can request that template directly on job submission.

To directly request job templates

1. Set the **SELECT** attribute on the template in `moab.cfg`.

```
JOBCFG[medium.set] NODESET=ONEOF:FEATURE:fast,slow SELECT=true
```

2. Submit a job with a resource list (msub -l), requesting the template using the format `template=<templateName>`.

```
> msub -l template=medium.set
```

Moab creates a job with the *medium.set* job template created in step 1.

Attributes set in the template are evaluated as if they were part of the job submission. They are still subject to all of the same ACLs and policies.

Related topics

- **Applying templates based on job attributes** on page 779

24.2.5 Creating workflows with job templates

Context

Moab can create workflows from individual jobs using job templates.

To build a workflow with job templates

1. Create the jobs in the workflow using the **JOBCFG** on page 849 parameter (See **Creating job templates** on page 778 for more information.). Specify the order in which they should run with the **TEMPLATEDEPEND** on page 791 attribute. Please see the Job dependency syntax table for a list of valid dependency options.

```
JOBCFG[setup.pre]    TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/setup.pre.sh STAGEIN=TRUE
JOBCFG[setup.pre2]   TEMPLATEDEPEND=AFTER:setup.pre SELECT=TRUE
EXEC=/nfs/tools/setup.pre2.sh
JOBCFG[engineering] TEMPLATEDEPEND=AFTER:setup.pre2
```

*When Moab applies the engineering template to a qualifying job, the job will not run until template job setup.pre and then setup.pre2 are created from the specified **EXEC** strings and finish running.*

The Moab naming convention for jobs created with job templates is *<moabId>.<templateName>*. By default, when Moab submits jobs to only one resource manager, the job IDs are synchronized with the resource manager's job IDs. You can use the parameter **USEMOABJOBID** on page 938 so that a template-created job is easily associated with its parent job (such as `moab.1`, `moab.1.setup.pre`).

Since the `setup.pre.sh` script is not on an NFS mount, it must be staged in. **STAGEIN** has a value of *TRUE* or *FALSE*. If set to *TRUE*, Moab ensures the executable is staged out to the compute host where the dependent job runs. If set to *FALSE*, Moab assumes the executable is already available on the compute host.

2. Create the job template that will act as the criteria a job must meet for Moab to apply the *engineering* template. In this situation, the job must be submitted with the account name *engineering*.

```
JOBCFG[engineering.match] ACCOUNT=engineering
```

3. Create the **JOBMATCHCFG** on page 852 configuration to tell Moab that when a job matches the *engineering.match* template, it should apply the *engineering* template.

```
JOBMATCHCFG[engineering.job] JMIN=engineering.match JSET=engineering
```

Related topics

- **Job template extension attributes** on page 782
- **Job template workflow examples** on page 795
- **Creating job templates** on page 778

24.3 Job template references

24.3.1 Job template extension attributes

When creating a job template, you can use any attribute acceptable within the WIKI workload query data format. In addition, job templates can use any of the extension attributes in the following table. Note that the Template type (**JMIN**, **JMAX**, **JDEF**, **JSET**) row indicates compatibility with the associated attribute (See **Applying templates based on job attributes** on page 779 for more information.).

Attributes set in a template are evaluated as if they were part of the original job submission. Their jobs are still subject to all the same ACLs and policies.

ACCOUNT	
Format	<ACCOUNT>[,<ACCOUNT>]...
Template type	**JMIN** **JDEF** **JSET**
Description	Account credentials associated with job. This is used for job template matching.

Example	`JOBCFG[public] FLAGS=preemptee`
	`JOBCFG[public.min] ACCOUNT=public_acct`
	`JOBMATCHCFG[public] JMIN=public.min JSET=public`

CLASS	
Format	<CLASS>[,<CLASS>]...
Template type	**JMIN** **JDEF** **JSET**
Description	Class credentials associated with job. This is used for job template matching.

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

CPULIMIT	
Format	[[[DD:]HH:]MM:]SS
Template type	**JMIN** **JMAX** **JDEF** **JSET**
Description	Maximum amount of CPU time used by all processes in the job.
Example	`JOBCFG[job.min] CPULIMIT=1:00:00:00` `JOBCFG[job.max] CPULIMIT=2:00:00:00`

DESCRIPTION	
Format	<STRING>
Template type	**JMAX** **JDEF**
Description	Description of the job. When you run the checkjob command, the description appears as Reason.
Example	`JOBCFG[webdb] DESCRIPTION="Template job"`

DPROCS	
Format	<INTEGER>
Template type	**JMIN** **JMAX** **JSET**
Description	Number of processors dedicated per task. The default is 1.
Example	`JOBCFG[job.min] DPROCS=2` `JOBCFG[job.max] DPROCS=4`

EXEC	
Format	<STRING>
Template type	**JSET**
Description	Specifies what the job runs, regardless of what the user set.
Example	`JOBCFG[setup.pre] EXEC=nfs/tools/setup.pre.sh`

FLAGS	
Format	<JOBFLAG>[,<JOBFLAG>]...
Template type	**JMIN** **JDEF** **JSET**
Description	One or more legal job flag values.
Example	`JOBCFG[webdb]` `FLAGS=NORMSTART`

GNAME	
Format	<STRING>
Template type	**JDEF** **JSET**
Description	Group credential associated with job.
Example	`JOBCFG[webserv] GNAME=service` For matching the group, see the GROUP attribute.

GRES	
Format	<genericResource>[:<COUNT>][,<genericResource>[:<COUNT>]]...
Template type	**JMAX** **JDEF**
Description	Consumable generic attributes associated with individual nodes or the special pseudo-node global, which provides shared cluster (floating) consumable resources. Use the NODECFG parameter to configure such resources.
Example	`JOBCFG[gres.set] GRES=abaqus:2` *In this example, the gres.set template applies two Abaqus licenses per task to a matched job.*

GROUP	
Format	<GROUP>[,<GROUP>]...
Template type	**JMIN**
Description	Group credentials associated with job. This is used for job template matching.
Example	`JOBCFG[webserv] GROUP=service` For information about setting the group, see the GNAME attribute.

MEM	
Format	<INTEGER>
Template type	**JMIN** **JMAX** **JDEF** **JSET**
Description	Maximum amount of physical memory per task used by the job in megabytes. You can optionally specify other units with your integer (`300kb` or `2gb`, for example). See Requesting Resources for more information.

MEM	
Example	`JOBCFG[smalljobs] MEM=25`

NODEACCESSPOLICY	
Format	One of the following: SHARED, SHAREDONLY, SINGLEJOB, SINGLETASK , SINGLEUSER, or UNIQUEUSER
Template type	**JDEF** **JSET**
Description	Specifies how node resources will be shared by a job. See the **Node Access Policies** on page 309 for more information.
Example	`JOBCFG[serverapp] NODEACCESSPOLICY=SINGLEJOB`

NODERANGE	
Format	<MIN>[,<MAX>]
Template type	**JMAX** **JDEF**
Description	Minimum and maximum nodes allowed to be allocated to job.
Example	`JOBCFG[vizserver] NODERANGE=1,16`

NODES	
Format	<INTEGER>
Template type	**JMIN** **JMAX** **JSET**
Description	Number of nodes required by the job. The default is 1. See Node Definition for more information.

NODES	
Example	`JOBCFG[job.min] NODES=2` `JOBCFG[job.max] NODES=4`

NODESET	
Format	<STRING>
Template type	**JSET**
Description	See **Node Set Overview** on page 439 for more information.
Example	`JOBCFG[medium.set]` `NODESET=ONEOF:FEATURE:fast,slow`

PARTITION	
Format	<PARTITION>[:<PARTITION>]...
Template type	**JMIN** **JDEF** **JSET**
Description	Specifies the partition (or partitions) in which a job must run.
Example	`JOBCFG[meis] PARTITION=math:geology`

PREF	
Format	<FEATURE>[,<FEATURE>]...
Template type	**JDEF** **JSET**
Description	Specifies which node features are preferred by the job and should be allocated if available. See PREF for more information.
Example	`JOBCFG[meis] PREF=bigmem`

PRIORITY	
Format	<INTEGER>
Template type	**JMAX** **JDEF**
Description	Relative job priority. **PRIORITY** works only as a default setting and not as an override (**JSET**) setting.
Example	`JOBCFG[meis] PRIORITY=25000`

PROCRANGE	
Format	<MIN>[,<MAX>]
Template type	JDEF JSET
Description	Minimum and maximum processors allowed to be allocated to job.
Example	`JOBCFG[meis] PROCRANGE=2,64`

QOS	
Format	<QOS>[,<QOS>]...
Template type	**JMIN** **JDEF** **JSET**
Description	QoS credentials associated with job. This is used for job template matching.

Example	`JOBCFG[admin] RFEATURES=bigmem`
	`JOBCFG[admin.min] QOS=admin_qos`
	`JOBMATCHCFG[admin] JMIN=admin.min JSET=admin`

RARCH	
Format	<STRING>
Template type	**JSET**
Description	Architecture required by job.
Example	`JOBCFG[servapp]` `RARCH=i386`

RFEATURES	
Format	<FEATURE>[,<FEATURE>]...
Template type	**JMIN** **JDEF** **JSET**
Description	List of features required by job.
Example	`JOBCFG[servapp]` `RFEATURES=fast,bigmem`

RM	
Format	<STRING>
Template type	**JDEF** **JSET**
Description	Destination resource manager to be associated with job.
Example	`JOBCFG[webdb] RM=slurm`

ROPSYS	
Format	<STRING>

ROPSYS	
Template type	JDEF JSET
Description	Operating system required by job.
Example	`JOBCFG[test.set]` `ROPSYS=windows`

SELECT	
Format	<BOOLEAN> : TRUE | FALSE
Description	Job template can be directly requested by job at submission.
Example	`JOBCFG[servapp] SELECT=TRUE`

SOFTWARE	
Format	<RESTYPE>[{+|:}<COUNT>][@<TIMEFRAME>]
Template type	JDEF JSET
Description	Indicates generic resources required by the job. See SOFTWARE for more information.
Example	`JOBCFG[servapp] SOFTWARE=matlab:2`

SYSTEMJOBTYPE	
Template type	JMIN
Description	System job type (ex. vmcreate).

Example	`JOBCFG[vmcreate.min] SYSTEMJOBTYPE=vmcreate`
	`JOBCFG[vmcreate.set]`
	`TRIGGER=atype=reserve,action="00:05:00",etype=end`
	`JOBMATCHCFG[vmcreate] JMIN=vmcreate.min JSET=vmcreate.set`

TASKS	
Format	<INTEGER>
Template type	**JMIN** **JMAX** **JSET**
Description	Number of tasks required by job. The default is 1. See Task Definition for more information.
Example	`JOBCFG[job.min] TASKS=4` `JOBCFG[job.max] TASKS=8`

TASKPERNODE	
Format	<INTEGER>
Template type	**JMIN** **JMAX** **JDEF**
Description	Exact number of tasks required per node. The default is 0. **TASKPERNODE** works only as a default setting and not as an override (**JSET**) setting.
Example	`JOBCFG[job.min] TASKPERNODE=2` `JOBCFG[job.max] TASKPERNODE=4`

TEMPLATEDEPEND	
Format	<TYPE>:<TEMPLATE_NAME>
Description	Create another job from the *<TEMPLATE_NAME>* job template, on which any jobs using this template will depend. This is used for dynamically creating workflows. See Job Dependencies for more information.
Example	`JOBCFG[engineering] TEMPLATEDEPEND=AFTER:setup.pre` `JOBCFG[setup.pre] SELECT=TRUE EXEC=/tools/setup.pre.sh`

UNAME	
Format	<STRING>
Default	**JDEF** **JSET**
Description	User credential associated with job.
Example	`JOBCFG[webserv] UNAME=service` For matching the user, see the USER attribute.

USER	
Format	<USER>[,<USER>]...
Template type	**JMIN** **JMAX**
Description	User credentials associated with job.
Example	`JOBCFG[webserv] USER=service` For setting the user, see the UNAME attribute.

VARIABLE	
Format	<NAME>[:<VAL>]
Template type	**JMIN** **JSET**
Description	Variables attached to the job template.
Example	`JOBCFG[this] VARIABLE=var1:1 VARIABLE=var2:1` Variables are set upon successful completion of the job.

WCLIMIT	
Format	[[HH:]MM:]SS
Template type	**JMIN** **JMAX** **JDEF** **JSET**
Description	Walltime required by job. The default is 8640000 (100 days).
Example	`JOBCFG[job.min] WCLIMIT=2:00:00` `JOBCFG[job.max] WCLIMIT=12:00:00`

Related topics

- **Job template examples** on page 794
- **Creating job templates** on page 778

24.3.2 Job template matching attributes

The **JOBMATCHCFG** on page 852 parameter allows you to establish relationships between a number of job templates. The table in **Job template extension attributes** on page 782 indicates which job template types are compatible with which job template extension attributes. The following types of templates can be specified with the JOBMATCHCFG parameter:

Attribute	Description
JMAX	A potential job is rejected if it has matching attributes set or has resource requests that exceed those specified in this template. For **JMAX**, a job template can specify only positive non-zero numbers as maximum limits for generic resources. If a job requests a generic resource that is not limited by the template, then the template can still be used.
JMIN	A potential job is rejected if it does not have matching attributes set or has resource requests that do not meet or exceed those specified in this template.
JDEF	A matching job has the specified attributes set as defaults but all values can be overridden by the user if the matching attribute is explicitly set at job submission time.
JSET	A matching job has the specified attributes forced to these values and these values override any values specified by the submitter at job submission time.

Attribute	Description
JSTAT	A matching job has its usage statistics reported into this template.

Related topics

- **Job template extension attributes** on page 782
- **Job template examples** on page 794
- **Applying templates based on job attributes** on page 779

24.3.3 Job template examples

Job templates can be used for a wide range of purposes including enabling automated learning, setting up custom application environments, imposing special account constraints, and applying group default settings. The following examples highlight some of these uses:

Example 24-1: Setting up application-specific environments

```
JOBCFG[xxx] EXEC=*app* JOBPROLOG=/usr/local/appprolog.x
```

Example 24-2: Applying job preferences and defaults

```
JOBCFG[xxx] CLASS=appq EXEC=*app* PREF=clearspeed
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT] PRIORITYF=5.0*PREF
```

Example 24-3: Applying resource constraints to fuzzy collections

In the following example, a job template match is set up. Using the **JOBMATCHCFG** on page 852 parameter, Moab is configured to apply all attributes of the *inter.set* job template to all jobs that match the constraints of the *inter.min* job template. In this example, all interactive jobs are assigned the *ignpolicies* flag that allows them to ignore active, idle, system, and partition level policies. Interactive jobs are also locked into the test standing reservation and thus only allowed to run on the associated nodes.

```
# limit all users to a total of two non-interactive jobs
USERCFG[DEFAULT] MAXJOB=2
SRCFG[test] DESCRIPTION="compute pool for interactive and short duration jobs"
SRCFG[test] JOBATTRLIST=INTERACTIVE
SRCFG[test] MAXTIME=1:00:00
SRCFG[test] HOSTLIST=R:atl[16-63]
JOBCFG[inter.min] FLAGS=interactive
JOBCFG[inter.set] FLAGS=ignpolicies
JOBMATCHCFG[interactive] JMIN=inter.min JSET=inter.set
```

Example 24-4: 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.

```
JOBCFG[no_inter] FLAGS=interactive
JOBCFG[preempt_job] FLAGS=preemptee
RMCFG[gridA.in] MAX.JOB=no_inter SET.JOB=preempt_job
```

Related topics

- **Job template extension attributes** on page 782
- **Job template workflow examples** on page 795
- **Creating job templates** on page 778

24.3.4 Job template workflow examples

Example 24-5: A workflow with multiple dependencies

In this example the job will depend on the completion of two other jobs Moab creates. Both jobs execute at the same time.

```
# Engineering2
JOBCFG[engineering2] TEMPLATEDEPEND=AFTER:engineering2.pre2
TEMPLATEDEPEND=AFTER:engineering2.pre
JOBCFG[engineering2.pre2] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/engineering2.pre2.sh
JOBCFG[engineering2.pre] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/engineering2.pre.sh
JOBCFG[engineering2.match] ACCOUNT=engineering2
JOBMATCHCFG[engineering2.job] JMIN=engineering2.match JSET=engineering2
```

Example 24-6: Jobs that run after the submission job

Three additional jobs are created that depend on the submitted job.

```
# Workflow 2
JOBCFG[workflow2] TEMPLATEDEPEND=BEFORE:workflow2.post1
TEMPLATEDEPEND=BEFORE:workflow2.post2 TEMPLATEDEPEND=BEFORE:workflow2.post3
JOBCFG[workflow2.post1] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow2.post1.sh
JOBCFG[workflow2.post2] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow2.post2.sh
JOBCFG[workflow2.post3] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow2.post3.sh
JOBCFG[workflow2.match] ACCOUNT=workflow2
JOBMATCHCFG[workflow2.job] JMIN=workflow2.match JSET=workflow2
```

Example 24-7: A complex workflow

A complex workflow that handles failures.

```
# Workflow 4
JOBCFG[workflow4.step1] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step1.sh TEMPLATEDEPEND=BEFOREFAIL:workflow4.fail1
JOBCFG[workflow4.fail1] TASKS=1 WCLIMIT=00:00:30 SELECT=TRUE
EXEC=/usr/tools/workflow.fail.1.sh TEMPLATEDEPEND=BEFOREANY:workflow4.fail2
JOBCFG[workflow4.fail2] TASKS=1 WCLIMIT=00:00:30 SELECT=TRUE
EXEC=/usr/tools/workflow.fail.2.sh
# Submission job
JOBCFG[workflow4.step2] TEMPLATEDEPEND=AFTEROK:workflow4.step1
TEMPLATEDEPEND=BEFOREOK:workflow4.step3.1 TEMPLATEDEPEND=BEFOREOK:workflow4.step3.2
JOBCFG[workflow4.step3.1] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step3.1.sh
JOBCFG[workflow4.step3.2] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step3.2.sh TEMPLATEDEPEND=BEFOREOK:workflow4.step4
JOBCFG[workflow4.step4] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step4.sh
JOBCFG[workflow4.step4] TEMPLATEDEPEND=BEFOREOK:workflow4.step5.1
TEMPLATEDEPEND=BEFOREOK:workflow4.step5.2 TEMPLATEDEPEND=BEFORENOTOK:workflow4.step5.3
JOBCFG[workflow4.step5.1] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step5.1.sh
JOBCFG[workflow4.step5.2] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step5.2.sh
JOBCFG[workflow4.step5.3] TASKS=1 WCLIMIT=00:00:30 SELECT=TRUE
EXEC=/usr/tools/workflow.step5.3.sh
JOBCFG[workflow4.match] ACCOUNT=workflow4
```

Related topics

- **Creating workflows with job templates** on page 781
- **Applying templates based on job attributes** on page 779
- **Job template examples** on page 794
- **Job template extension attributes** on page 782

25.0 Appendices

Appendix A: Moab Parameters

See the Parameters Overview in the Moab Admin Manual for further information about specifying parameters.

Index: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

ACCOUNTCFG[<ACCOUNTID>]

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: General Credential Flags, **CHARGERATE**, PRIORITY, ENABLEPROFILING, MEMBERULIST, **PLIST**, **QDEF**, **QLIST**, usage limit, or a fairness usage limit specification (FSCAP, FSTARGET, and FSWEIGHT).
Default	---
Description	Specifies account specific attributes. See the account overview for general information and the job flag overview for a description of legal flag values.
Example	`ACCOUNTCFG[projectX] MAXJOB=50 QDEF=highprio` *Up to 50 jobs submitted under the account ID projectX will be allowed to execute simultaneously and will be assigned the QoS highprio by default.*

ACCOUNTINGINTERFACEURL

Format	*<URL>* where protocol can be one of **exec** or **file**
Default	---
Description	Specifies the interface to use for real-time export of Moab accounting/auditing information. See Exporting Events in Real-Time for more information.
Example	`ACCOUNTINGINTERFACEURL exec:///$TOOLSDIR/dumpacc.pl`

ACCOUNTWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the priority weight to be applied to the specified account priority. See Credential (CRED) Factor.
Example	`ACCOUNTWEIGHT 100`

ADMIN1, ADMIN2, ADMIN3

Format	Space-delimited list of user names
Default	*root*
Description	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.
Example	`ADMIN1 moabuser steve scott jenny` *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**, **GROUPS**, SERVICES, or **NAME**
Default	---
Description	Allows a site to configure which services and users belong to a particular level of administration. **Note:** The first user listed in the ADMINCFG[1] users list is considered to be the primary admin. The option **USERS**=*ALL* is allowed. The groups list adds the groups' users as if they were listed individually as **USERS**. To prevent Moab from assigning a primary user from the first group listed, you must specify a primary user first using the **USERS** attribute, then list the desired groups.
Example	

```
ADMINCFG[1] USERS=root,john
ADMINCFG[1] GROUPS=admin
ADMINCFG[1] SERVICES=ALL
ADMINCFG[1] NAME=batchadmin
ADMINCFG[3] USERS=bob,carol,smoore
ADMINCFG[3] GROUPS=science,math
ADMINCFG[3] SERVICES=mjobctl,mcredctl,runjob
ADMINCFG[3] NAME=helpdesk
```

Members of the batchadmin admin role and members of the admin group are allowed to run all commands. Members of the helpdesk role and science and math groups are allowed to run mjobctl. They are also able to view and modify credential objects (i.e. users, groups, accounts, etc.) See the security overview for more details.

```
ADMINCFG[4] USERS=ALL SERVICES=checknode
```

All users can execute mdiag -n or checknode to get information on any node.

AGGREGATENODEACTIONS

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Consolidates queued node actions into as few actions as possible to reduce communication burden with resource manager. Node actions are queued until the AGGREGATENODEACTIONSTIME setting. This may delay some node actions. When reprovisioning, the system job may expire before the provision action occurs; while the action will still occur, the job will not show it.
Example	`AGGREGATENODEACTIONS TRUE` *Queues node actions together when possible.*

AGGREGATENODEACTIONSTIME

Format	*<SECONDS>*
Default	*60*
Description	The delay time for the AGGREGATENODEACTIONS parameter to aggregate requests before sending job batches.
Example	`AGGREGATENODEACTIONSTIME 120` *Sets the AGGREGATENODEACTIONS delay to two minutes.*

ALLOWMULTIREQNODEUSE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	By default Moab does not allow different requirements on the same job to occupy the same node. For example, if a job is submitted with `nodes=2:ppn=8+4:fast:ppn=16`, it's possible that some of the tasks requested could overlap onto the same node. This parameter instructs Moab to allow overlapping the same node, or not. This parameter also applies to the various `-w` clauses of an **mshow -a** on page 192 command.
Example	`ALLOWMULTIREQNODEUSE TRUE`

ALLOWROOTJOBS	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether batch jobs from the root user (UID=0) are allowed to be executed. **Note**: The resource manager must also support root jobs.
Example	`ALLOWROOTJOBS TRUE` *Jobs from the root user can execute.*

ALLOWVMMIGRATION	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Enables Moab to migrate VMs.
Example	`ALLOWVMMIGRATION TRUE`

ALWAYSEVALUATEALLJOBS

Format	*<BOOLEAN>*
Default	*FALSE*
Description	When scheduling priority jobs, Moab stops scheduling when it encounters the first job that cannot run and cannot get a reservation. ALWAYSEVALUATEALLJOBS directs Moab to continue scheduling until all priority jobs (jobs that do not violate any limits) are evaluated.
Example	`ALWAYSEVALUATEALLJOBS TRUE`

AMCFG

Format	One or more key-value pairs as described in the Allocation Manager Configuration Overview.
Default	---
Description	Specifies the interface and policy configuration for the scheduler-allocation manager interface. Described in detail in the Allocation Manager Configuration Overview.
Example	`AMCFG[mam] SERVER=mam://master.ufl.edu STARTFAILUREACTION=HOLD TIMEOUT=15`

APPLICATIONLIST

Format	Space-delimited list of generic resources.
Default	---
Description	Specifies which generic resources represent actual applications on the cluster. 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.

ARRAYJOBPARLOCK	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, all sub jobs of an array are locked to a single partition. The default behavior when scheduling array sub jobs is to span the jobs across partitions when possible. The ARRAYJOBPARLOCK job flag can be used to specify partition locking at submit time. The ARRAYJOBPARSPAN job flag overrides this parameter.
Example	`ARRAYJOBPARLOCK TRUE`

ASSIGNVLANFEATURES	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	When set to *TRUE*, this forces all VMs to be contained in VLANs.
Example	`ASSIGNVLANFEATURES TRUE`

ATTRATTRWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight to be applied to jobs with the specified job attribute. See Attribute (ATTR) Factor.
Example	`ATTRATTRWEIGHT 100`

ATTRGRESWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight to be applied to jobs requesting the specified generic resource. See Attribute (ATTR) Factor.
Example	`ATTRGRESWEIGHT 200`

ATTRSTATEWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight to be applied to jobs with the specified job state. See Attribute (ATTR) Factor.
Example	`ATTRSTATEWEIGHT 200`

ATTRWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the priority component weight to be applied to the ATTR subcomponents. See Attribute (ATTR) Factor.
Example	`ATTRWEIGHT 2` `ATTRSTATEWEIGHT 200`

BACKFILLDEPTH	
Format	*<INTEGER>*
Default	*0* (no limit)
Description	Specifies the number of idle jobs to evaluate for backfill. The backfill algorithm will evaluate the top *<X>* priority jobs for scheduling. By default, all jobs are evaluated.
Example	`BACKFILLDEPTH 128` *Evaluate only the top 128 highest priority idle jobs for consideration for backfill.*

BACKFILLPOLICY	
Format	One of *FIRSTFIT* or *NONE*
Default	*FIRSTFIT*
Description	Specifies which backfill algorithm will be used. See Configuring Backfill for more information.
Example	`BACKFILLPOLICY  NONE`

BFCHUNKDURATION	
Format	*[[[DD:]HH:]MM:]SS*
Default	*0* (chunking disabled)
Description	Specifies the duration during which freed resources will be aggregated for use by larger jobs. Used in conjunction with **BFCHUNKSIZE** on page 806. See Configuring Backfill for more information.

Example	`BFCHUNKDURATION 00:05:00`
	`BFCHUNKSIZE     4`
	Aggregate backfillable resources for up to 5 minutes, making resources available only to jobs of size 4 or larger.

BFCHUNKSIZE

Format	*<INTEGER>*
Default	*0* (chunking disabled)
Description	Specifies the minimum job size which can utilize chunked resources. Used in conjunction with **BFCHUNKDURATION** on page 805. See Configuring Backfill for more information.

Example	`BFCHUNKDURATION 00:05:00`
	`BFCHUNKSIZE 4`
	Aggregate backfillable resources for up to 5 minutes, making resources available only to jobs of size 4 or larger.

BFMINVIRTUALWALLTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	---
Description	Specifies the minimum job wallclock time for virtual scaling (optimistic-like backfilling.) Any job with a wallclock time less than this setting will *not* be virtually scaled. The value specified relates to a job's original walltime and not its virtually-scaled walltime.
Example	`BFMINVIRTUALWALLTIME 00:01:30`

BFPRIORITYPOLICY

Format	One of *RANDOM*, *DURATION*, or *HWDURATION*
Default	---
Description	Specifies policy to use when prioritizing backfill jobs for preemption
Example	`BFPRIORITYPOLICY DURATION` *Use length of job in determining which backfill job to preempt.*

BFVIRTUALWALLTIMECONFLICTPOLICY	
Format	One of the following: *PREEMPT*
Default	---
Description	Specifies how to handle scheduling conflicts when a virtually scaled job "expands" to its original wallclock time. This occurs when the job is within one scheduling iteration - **RMPOLLINTERVAL** on page 907 - of its virtually scaled wallclock time expiring.
Example	`BFVIRTUALWALLTIMECONFLICTPOLICY  PREEMPT`

BFVIRTUALWALLTIMESCALINGFACTOR	
Format	*<DOUBLE>*
Default	*0* (virtual scaling disabled)
Description	Specifies the factor by which eligible jobs' wallclock time is virtually scaled (optimistic-like backfilling).
Example	`BFVIRTUALWALLTIMESCALINGFACTOR .4`

BYPASSCAP	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the max weighted value allowed from the bypass count subfactor when determining a job's priority (see Priority Factors for more information).
Example	`BYPASSWEIGHT 5000` `BYPASSCAP    30000`

BYPASSWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight to be applied to a job's backfill bypass count when determining a job's priority (see Priority Factors for more information).
Example	`BYPASSWEIGHT 5000`

CHECKPOINTDIR

Format	*<STRING>*
Default	---
Description	Specifies the directory for temporary job checkpoint files (usually of the form `jobid.cp`). This is *not* the directory for Moab's checkpoint file (`.moab.ck`).
Example	`CHECKPOINTDIR  /tmp/moabcheckpoint`

CHECKPOINTEXPIRATIONTIME

Format	*[[[DD:]HH:]MM:]SS* or *UNLIMITED*
Default	*3,000,000 seconds*
Description	Specifies how 'stale' checkpoint data can be before it is ignored and purged.
Example	`CHECKPOINTEXPIRATIONTIME 1:00:00:00` *Expire checkpoint data which has been stale for over 1 day.*

CHECKPOINTFILE	
Format	*<STRING>*
Default	*moab.ck*
Description	Name (absolute or relative) of the Moab checkpoint file.
Example	`CHECKPOINTFILE /var/adm/moab/moab.ck` *Maintain the Moab checkpoint file in the file specified.*

CHECKPOINTINTERVAL	
Format	*[[[DD:]HH:]MM:]SS*
Default	*00:05:00*
Description	Time between automatic Moab checkpoints. If RMPOLLINTERVAL does not specify both a minimum and maximum poll time, Moab will ignore CHECKPOINTINTERVAL and checkpoint every iteration.
Example	`CHECKPOINTINTERVAL 00:15:00` *Moab should checkpoint state information every 15 minutes.*

CHECKPOINTWITHDATABASE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, Moab stores checkpoint information to a database rather than to the `.moab.ck` flat text file.
Example	`CHECKPOINTWITHDATABASE TRUE`

CHECKSUSPENDEDJOBPRIORITY

Format	*<BOOLEAN>*
Default	*TRUE*
Description	Prevents Moab from starting a job on any node containing a suspended job of higher priority.
Example	`CHECKSUSPENDEDJOBPRIORITY FALSE`

CHILDSTDERRCHECK

Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, child processes Moab executes are considered failed if their standard error stream contains the text "ERROR".
Example	`CHILDSTDERRCHECK TRUE`

CLASSCFG[<CLASSID>]

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: General Credential Flags, DEFAULT.ATTR, DEFAULT.DISK, DEFAULT.FEATURES, DEFAULT.GRES, DEFAULT.MEM, DEFAULT.NODE, DEFAULT.NODESET, DEFAULT.PROC, ENABLEPROFILING, EXCL.FEATURES, EXCLUDEUSERLIST, HOSTLIST, JOBEPILOG, JOBPROLOG, MAXPROCPERNODE, MAX.NODE, MAX.PROC, MAX.WCLIMIT, MIN.NODE, MIN.PROC, MIN.TPN, MIN.WCLIMIT, PARTITION, PRIORITY, PRIORITYF, QDEF, QLIST, REQ.FEATURES, REQUIREDACCOUNTLIST, REQUIREDUSERLIST, RESFAILPOLICY, SYSPRIO, WCOVERRUN, usage limit, or fairshare usage limit specification.
Default	---
Description	Specifies class specific attributes (see Credential Overview for details).
Example	`CLASSCFG[batch] MAXJOB=50 QDEF=highprio` *Up to 50 jobs submitted to the class batch will be allowed to execute simultaneously and will be assigned the QoS highprio by default.*

CLASSWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the weight to be applied to the class priority of each job (see Credential (CRED) Factor and credential priority).
Example	`CLASSWEIGHT 10`

CLIENTCFG[<X>]

Format	One or more of *<ATTR>-<VALUE>* pairs where *<X>* indicates the specified peer and *<ATTR>* is one of the following: **AUTH**, **AUTHCMD**, **AUTHTYPE**, **HOST**, **KEY**, or **DEFAULTSUBMITPARTITION**.
Default	---
Description	Specifies the shared secret key and authentication method which Moab will use to communicate with the named peer daemon. See Security Overview for more information. **Note**: The **AUTHTYPE** and **KEY** attributes of this parameter may only be specified in the `moab-private.cfg` config file.
Example	`CLIENTCFG[silverB] KEY=apple7 AUTH=admin1` *Moab will use the session key apple7 for peer authentication and for encrypting and decrypting messages sent from silverB. Also, client connections from this interface will be authorized at an admin 1 level.*

CLIENTMAXCONNECTIONS

Format	*<INTEGER>*
Default	*128*
Description	Changes the maximum number of connections that can simultaneously connect to Moab. The value can be increased during runtime, but it cannot be decreased. The value cannot be reduced below the default value of *128*.
Example	`CLIENTMAXCONNECTIONS 256` *Doubles the maximum number of connections.*

CLIENTMAXPRIMARYRETRY	
Format	*<INTEGER>* or *INFINITY*
Default	*1*
Description	Specifies how many times the client command will attempt to retry its connection to the primary server if Moab is not available.

Example	`CLIENTMAXPRIMARYRETRY 5`
	`CLIENTMAXPRIMARYRETRYTIMEOUT 1000`
	The client command will attempt to retry its connection to the primary server 5 times with 1 second intervals before giving up. ***Note****: If INFINITY is specified, Moab will attempt 2,140,000,000 times.*

CLIENTMAXPRIMARYRETRYTIMEOUT	
Format	*<INTEGER>* (milliseconds)
Default	*2000*
Description	Specifies how much time to wait until the client command will attempt to retry its connection to the primary server if Moab is not available.

Example	`CLIENTMAXPRIMARYRETRY 3`
	`CLIENTMAXPRIMARYRETRYTIMEOUT 500`
	The client command will attempt to retry its connection to the primary server 3 times with .5 second intervals before giving up.

CLIENTTIMEOUT

Format	*[[[DD:]HH:]MM:]SS*
Default	*00:00:30*
Description	Time which Moab client commands will wait for a response from the Moab server. See Client Configuration for more information. **Note**: May also be specified as an environment variable.
Example	`CLIENTTIMEOUT 00:15:00` *Moab clients will wait up to 15 minutes for a response from the server before timing out.*

CREDDISCOVERY

Format	*TRUE*
Default	*FALSE*
Description	Specifies that Moab should create otherwise unknown credentials when it discovers them in the statistics files.
Example	`CREDDISCOVERY TRUE`

CREDWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the credential component weight associated with the credential priority. See Credential (CRED) Factor for more information.
Example	`CREDWEIGHT 2`

DATASTAGEHOLDTYPE	
Format	Any Job Hold type
Default	*DEFER*
Description	Specifies what to do if a job's data staging operations fail.
Example	`DATASTAGEHOLDTYPE  BATCH`

DEADLINEPOLICY	
Format	One of *CANCEL*, *HOLD*, *IGNORE*, or *RETRY*
Default	*NONE*
Description	Specifies what to do when a requested deadline cannot be reached (see Job Deadlines).
Example	`DEADLINEPOLICY  IGNORE`

DEFAULTCLASSLIST	
Format	Space-delimited list of one or more <*STRING*>s.
Default	---
Description	Specifies the default classes supported on each node for RM systems which do not provide this information.
Example	`DEFAULTCLASSLIST serial parallel`

DEFAULTSUBMITPARTITION	
Format	See parameter CLIENTCFG[].
Default	---
Description	If a user submits a job using msub which does not specify host, feature, or partition constraints, then the msub client will insert the specified default submit partition into the newly submitted job as a hard requirement.
Example	`CLIENTCFG[DEFAULT] DEFAULTSUBMITPARTITION=partition1`

DEFERCOUNT	
Format	*<INTEGER>*
Default	*24*
Description	Specifies the number of times a job can be deferred before it will be placed in batch hold.
Example	`DEFERCOUNT 12`

DEFERSTARTCOUNT	
Format	*<INTEGER>*
Default	*1*
Description	Specifies the number of times a job will be allowed to fail in its start attempts before being deferred. JOBRETRYTIME overrides DEFERSTARTCOUNT; DEFERSTARTCOUNT only begins when the JOBRETRYTIME window elapses. **Note**: A job's startcount will increase each time a start request is made to the resource manager regardless of whether or not this request succeeded. This means start count increases if job starts fail or if jobs are started and then rejected by the resource manager. (For related information, see Reservation Policies, DEFERTIME, RESERVATIONRETRYTIME, NODEFAILURERESERVETIME, JOBRETRYTIME, and GUARANTEEDPREEMPTION.)
Example	`DEFERSTARTCOUNT 3`

DEFERTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*1:00:00*
Description	Specifies the amount of time a job will be held in the deferred state before being released back to the Idle job queue. **Note**: A job's defer time will be restarted if Moab is restarted. (For related information, see Reservation Policies, DEFERSTARTCOUNT, RESERVATIONRETRYTIME, NODEFAILURERESERVETIME, JOBRETRYTIME, and GUARANTEEDPREEMPTION.)
Example	`DEFERTIME 0:05:00`

DELETESTAGEOUTFILES

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether the scheduler should delete explicitly specified stageout files after they are successfully staged. By default, such files are not deleted but are left on the nodes where the job ran.

Example	`DELETESTAGEOUTFILES TRUE`
	`Example of an explicit stageout request`
	`msub x=MSTAGEOUT:ssh://source_node/tmp/file,file:///results_folder`
	`job.cmd`
	With this parameter set to TRUE, `/tmp/file` on source_node is deleted after it is copied to the specified destination (`file:///results_folder`). If the parameter is not set, or if it is set to FALSE, `/tmp/file` remains on source_node after the job terminates.

DEPENDFAILUREPOLICY	
Format	*HOLD* or *CANCEL*
Default	*HOLD*
Description	Specifies what happens to a job if its dependencies cannot be fulfilled; that is, what happens when a job depends on another job to complete successfully but the other job fails.
Example	`DEPENDFAILUREPOLICY CANCEL` *If job A is submitted with depend=afterok:B and job B fails, job A is canceled.*

DIRECTORYSERVER	
Format	*<HOST>[:<PORT>]*
Default	---
Description	Specifies the interface for the directory server.
Example	`DIRECTORYSERVER calli3.icluster.org:4702`

DISABLEEXCHLIST	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If the resource manager rejects a job and the value is set to *TRUE*, then the node is not added to the job's exclude host list.
Example	`DISABLEEXCHLIST TRUE`

DISABLEINTERACTIVEJOBS

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Disallows interactive jobs submitted with msub -I. **Note**: It is possible for users to submit interactive jobs directly to a resource manager, which can bypass the DISABLEINTERACTIVEJOBS parameter. However, some resource managers (such as TORQUE) will check with Moab before allowing an interactive job.
Example	`DISABLEINTERACTIVEJOBS TRUE`

DISABLEREGEXCACHING

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Turns off regular expression caching. Turning off regular expression caching preserves memory with host list reservations and speeds up start time.
Example	`DISABLEREGEXCACHING TRUE`

DISABLEREQUIREDGRESNONE

Format	*<BOOLEAN>*
Default	*FALSE*
Description	When set to *TRUE*, this causes Moab to reject msub requests that have a gres of "none". ENFORCEGRESSACCESSS must also be set to TRUE for this feature to work.

Example	`########## moab.cfg ##########`
	`ENFORCEGRESACCESS TRUE`
	`DISABLEREQUIREDGRESNONE TRUE`
	`################################`
	
	`> msub -A ee -l nodes=1,ttc=5,walltime=600,partition=g02 -l gres=none`
	`ERROR: cannot submit job - cannot locate required resource 'none'`

DISABLESAMECREDPREEMPTION	
Format	Comma-delimited list of one or more credentials: *ACCT*, *CLASS*, *GROUP*, *QOS*, or *USER*
Default	---
Description	This parameter prevents specified credentials from preempting its own jobs.
Example	`DISABLESAMECREDPREEMPTION  QOS,USER`

DISABLESCHEDULING	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether or not the scheduler will schedule jobs. If set to *TRUE*, Moab will continue to update node and job state but will not start, preempt, or otherwise modify jobs. The command mschedctl -r will clear this parameter and resume normal scheduling.
Example	`DISABLESCHEDULING FALSE`

DISABLETHRESHOLDTRIGGERS	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	This makes Moab not fire threshold-based triggers, but will log the intended action to the event logs. Similar to DISABLEVMDECISIONS.
Example	`DISABLETHRESHOLDTRIGGERS TRUE`

DISABLEVMDECISIONS

Format	*<BOOLEAN>*
Default	*FALSE*
Description	This makes Moab not take any automatic decisions with respect to VM's, namely powering on/off nodes and migrating VMs. Intended actions will instead be logged in the event logs. Similar to DISABLETHRESHOLDTRIGGERS.
Example	`DISABLEVMDECISIONS TRUE`

DISKWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight to be applied to the amount of dedicated disk space required per task by a job (in MB).

Example	`RESWEIGHT  10`
	`DISKWEIGHT 100`
	*If a job requires 12 tasks and 512 MB per task of dedicated local disk space, Moab will increase the job's priority by 10 * 100 * 12 * 512*

DISPLAYFLAGS	
Format	One or more of the following values (space delimited): *ACCOUNTCENTRIC*, *HIDEBLOCKED*, *HIDEDRAINED*, *NODECENTRIC*, or *USEBLOCKING*
Default	---
Description	Specifies flags that control how Moab client commands display varied information. *ACCOUNTCENTRIC* will display account information in showq, rather than group information. *HIDEBLOCKED* will prevent showq from listing information about blocked jobs which are not owned by the user if the user is not an admin. *HIDEDRAINED* prevents mdiag -n from displaying nodes and mvmctl -q from displaying VMs in the DRAINED state. An override option of mdiag -n -w nodestate=drained lists only those nodes with a DRAINED state. Similarly, an override option of mvmctl -q -w state=drained lists only those VMs with a DRAINED state. *NODECENTRIC* will display node allocation information instead of processor allocation information in showq. *USEBLOCKING* disables threading for commands that support it; those commands include showq, mdiag -n, and mdiag -j.
Example	`DISPLAYFLAGS NODECENTRIC`

DISPLAYPROXYUSERASUSER	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, Moab shows the proxy users instead of the real user on some queries of system jobs that have proxy users. Commands affected include mjobctl -q diag and checkjob.
Example	`DISPLAYPROXYUSERASUSER TRUE`

DONTCANCELINTERACTIVEHJOBS

Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, Moab does not cancel interactive jobs that are held.
Example	`DONTCANCELINTERACTIVEHJOBS TRUE`

DONTENFORCEPEERJOBLIMITS

Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, only the scheduler that is running the job can cancel the job or enforce other limits.
Example	`DONTENFORCEPEERJOBLIMITS TRUE`

EMULATIONMODE

Format	*SLURM*
Default	---
Description	Specifies whether or not the scheduler will perform the automatic setup of a particular resource manager environment.
Example	`EMULATIONMODE SLURM` *Moab will perform the automated setup steps as if it were interfacing with a slurm resource manager (automatic QoS creation).*

ENABLEFAILUREFORPURGEDJOB

Format	*<BOOLEAN>*

ENABLEFAILUREFORPURGEDJOB	
Default	*FALSE*
Description	By default, when a job is purged or removed by the TORQUE resource manager for a walltime violation, the job takes on a state of Completed and a completion code of 0. If *TRUE*, the job state is set to Removed and has a completion code of 98. ENABLEFAILUREFORPURGEDJOB is for the TORQUE resource manager only. For ENABLEFAILUREFORPURGEDJOB to return Removed job states, you must reset the TORQUE server attribute `keep_completed` to `0` in `qmgr`. See "Queue attributes" in the TORQUE Administrator Guide for more information.
Example	`ENABLEFAILUREFORPURGEDJOB TRUE` *Jobs that are purged or removed by TORQUE are given a state of Removed and a completion code of 98.*

ENABLEFSVIOLATIONPREEMPTION	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, Moab will allow jobs within the same class/queue to preempt when the preemptee is violating a fairshare target and the preemptor is not.
Example	`ENABLEFSVIOLATIONPREEMPTION TRUE`

ENABLEHIGHTHROUGHPUT	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Configures Moab so that it will accept msub submissions, start jobs, process triggers, etc., in a manner which minimizes their processing time. The downside is that Moab will return minimal information about these jobs at submit time. If ENABLEHIGHTHROUGHPUT is *TRUE*, you must set **NODEALLOCATIONPOLICY** on page 872 to *FIRSTAVAILABLE*.
Example	`ENABLEHIGHTHROUGHPUT TRUE` *Moab can now accept hundreds of jobs per second using msub instead of 20-30.*

ENABLEJOBARRAYS	
Format	*<BOOLEAN>*
Default	*TRUE*
Description	If set to *TRUE*, job arrays will be enabled .
Example	`ENABLEJOBARRAYS TRUE`

ENABLENEGJOBPRIORITY	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, the scheduler allows job priority value to range from -INFINITY to MMAX_PRIO; otherwise, job priority values are given a lower bound of '1'. For more information, see REJECTNEGPRIOJOBS.
Example	`ENABLENEGJOBPRIORITY TRUE` *Job priority may range from -INFINITY to MMAX_PRIO.*

ENABLENODEADDRLOOKUP	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, the scheduler will use the default host name service lookup mechanism (i.e., `/etc/hosts`, DNS, NIS, etc.) to determine the IP address of the nodes reported by the resource manager. This information is used to correlate information reported by multi-homed hosts.
Example	`ENABLENODEADDRLOOKUP TRUE`

ENABLEPOSUSERPRIORITY	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, the scheduler will allow users to specify positive job priority values which will be honored. In other words, users can specify a priority that falls in the range of -1024 to +1023, inclusive. If set to *FALSE* (the default), user priority values are given an upper bound of '0' when users request a positive priority. See USERPRIOWEIGHT.
Example	`ENABLEPOSUSERPRIORITY TRUE` *Users may now specify positive job priorities and have them take effect (e.g. `msub -p 100 job.cmd`).*

ENABLESPVIOLATIONPREEMPTION	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, Moab will allow jobs within the same class/queue to preempt when the preemptee is violating a *soft*usage policy and the preemptor is not.
Example	`ENABLESPVIOLATIONPREEMPTION TRUE`

ENABLEVMDESTROY

Format	*<BOOLEAN>*
Default	*FALSE*
Description	If set to *TRUE*, enables the automatic destruction of a VM when the VM wall time is expired or when the VM is stale and configured to be destroyed (for more information, see VMSTALEACTION).
Example	`ENABLEVMDESTROY TRUE`

ENFORCEACCOUNTACCESS

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether or not Moab will enforce account access constraints without an allocation manager.
Example	`ENFORCEACCOUNTACCESS TRUE`

ENFORCEGRESACCESS

Format	*<BOOLEAN>*
Default	*FALSE*
Description	If a user submits a job with a non-existent gres (e.g. in the case of a typo) and ENFORCEGREACCESS is not set in `moab.cfg`, or is set to *FALSE*, then the requested gres will be created (but will not exist on any nodes) and the job will be deferred (similar to ENFORCEACCOUNTACCESS).
Example	`ENFORCEGRESACCESS TRUE`

EVENTLOGWSPASSWORD	
Format	*<STRING>*
Default	---
Description	Specifies the user password required for logging in to your web services. You should set this parameter in the `moab-private.cfg` file. In conjunction with this parameter, you will also need to configure the following parameters to set up event logging to Moab Web Services: • PUSHEVENTSTOWEBSERVICE • EVENTLOGWSURL • EVENTLOGWSUSER
Example	`EVENTLOGWSPASSWORD adminpw`

EVENTLOGWSURL	
Format	*<STRING>*
Default	---
Description	Specifies your web services event log URL. For example, if you are using Moab Web Services: *< HOST>:< PORT><URL>* In conjunction with this parameter, you will also need to configure the following parameters to set up event logging to Moab Web Services: • PUSHEVENTSTOWEBSERVICE • EVENTLOGWSUSER • EVENTLOGWSPASSWORD
Example	`EVENTLOGWSURL wsServer:8080/mws/rest/events`

EVENTLOGWSUSER

Format	*<STRING>*
Default	---
Description	Specifies the username required for logging in to your web services. You should set this parameter in the `moab-private.cfg` file. In conjunction with this parameter, you will also need to configure the following parameters to set up event logging to Moab Web Services: • PUSHEVENTSTOWEBSERVICE • EVENTLOGWSURL • EVENTLOGWSPASSWORD
Example	`EVENTLOGWSUSER admin`

EVENTSERVER

Format	*<HOST>[:<PORT>]*
Default	---
Description	Specifies the interface for the event server.
Example	`EVENTSERVER calli3.icluster.org:4702`

FEATURENODETYPEHEADER	
Format	*<STRING>*
Default	---
Description	Specifies the header used to specify node type via node features (i.e. LL features or PBS node attributes).
Example	`FEATURENODETYPEHEADER xnt` *Moab will interpret all node features with the leading string xnt as a nodetype specification - as used by the allocation manager and other allocation managers, and assign the associated value to the node. i.e., xntFast.*

FEATUREPARTITIONHEADER	
Format	*<STRING>*
Default	---
Description	Specifies the header used to specify node partition via node features (i.e. LL features or PBS node attributes).
Example	`FEATUREPARTITIONHEADER xpt` *Moab will interpret all node features with the leading string xpt as a partition specification and assign the associated value to the node. i.e., xptGold.*

FEATUREPROCSPEEDHEADER

Format	*<STRING>*
Default	---
Description	Specifies the header used to extract node processor speed via node features (i.e., LL features or PBS node attributes). **Note**: Adding a trailing '$' character will specifies that only features with a trailing number be interpreted. For example, the header 'sp$' will match 'sp450' but not 'sport'.
Example	`FEATUREPROCSPEEDHEADER xps` *Moab will interpret all node features with the leading string xps as a processor speed specification and assign the associated value to the node. i.e., xps950.*

FEATURERACKHEADER

Format	*<STRING>*
Default	---
Description	Specifies the header used to extract node rack index via node features (i.e., LL features or PBS node attributes). **Note**: Adding a trailing '$' character will specifies that only features with a trailing number be interpreted. For example, the header 'rack$' will match 'rack4' but not 'racket'.
Example	`FEATURERACKHEADER rack` *Moab will interpret all node features with the leading string rack as a rack index specification and assign the associated value to the node. i.e., rack16.*

FEATURESLOTHEADER	
Format	*<STRING>*
Default	---
Description	Specifies the header used to extract node slot index via node features (i.e., LL features or PBS node attributes). **Note**: Adding a trailing '$' character will specifies that only features with a trailing number be interpreted. For example, the header 'slot$' will match 'slot12' but not 'slotted'.
Example	`FEATURESLOTHEADER slot` *Moab will interpret all node features with the leading string slot as a slot index specification and assign the associated value to the node. i.e., slot16.*

FEEDBACKPROGRAM	
Format	*<STRING>*
Default	---
Description	Specifies the name of the program to be run at the completion of each job. If not fully qualified, Moab will attempt to locate this program in the 'tools' subdirectory.
Example	`FEEDBACKPROGRAM /var/moab/fb.pl` *Moab will run the specified program at the completion of each job.*

FILEREQUESTISJOBCENTRIC	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether a job's file request is a total request for the job or a per task request.
Example	`FILEREQUESTISJOBCENTRIC TRUE` *Moab will treat file requests as a total request per job.*

FILTERCMDFILE

Format	*<BOOLEAN>*
Default	*TRUE*
Description	Running the msub command performs the following operations on the submission script: • Replace all comments with spaces (excludes Resource Manager directives) • Strip empty lines • Replace `\r` with `\n` • Lock job to a PBS resource manager if `$PBS` is found in the script Include the FILTERCMDFILE parameter in the `moab.cfg` file that resides on the clients. FILTERCMDFILE must be *FALSE* for **REJECTDOSSCRIPTS** on page 898 to work correctly.
Example	`FILTERCMDFILE FALSE` *Running the msub command does not perform the actions detailed earlier.*

FORCENODEREPROVISION

Format	*<BOOLEAN>*
Default	*FALSE*
Description	When set to *TRUE*, this config option causes Moab to reprovision a node, even if it is to the same operating system (in essence rewriting the OS).
Example	`FORCENODEREPROVISION TRUE`

FORCERSVSUBTYPE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies that admin reservations must have a subtype associated with them.
Example	`FORCERSVSUBTYPE TRUE` *Moab will require all admin reservations to include a subtype.*

FREETIMELOOKAHEADDURATION	
Format	*[[[DD:]HH:]MM:]SS*
Default	*2 Months*
Description	Specifies how far ahead Moab will look when calculating free time on a node.
Example	`FREETIMELOOKAHEADDURATION 7:00:00:00` *Moab will look 1 week ahead when it calculates free time on a node.*

FSACCOUNTWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to the account subcomponent of the fairshare component of priority.
Example	`FSACCOUNTWEIGHT 10`

FSCAP

Format	*<DOUBLE>*
Default	*0* (NO CAP)
Description	Specifies the maximum allowed absolute value for a job's total pre-weighted fairshare component.
Example	`FSCAP 10.0` *Moab will bound a job's pre-weighted fairshare component by the range +/- 10.0.*

FSCLASSWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to the class subcomponent of the fairshare component of priority.
Example	`FSCLASSWEIGHT 10`

FSDECAY

Format	*<DOUBLE>*
Default	*1.0*
Description	Specifies decay rate applied to past fairshare interval when computing effective fairshare usage. Values may be in the range of 0.01 to 1.0. A smaller value causes more rapid decay causing *aged* usage to contribute less to the overall effective fairshare usage. A value of 1.0 indicates that no decay will occur and all fairshare intervals will be weighted equally when determining effective fairshare usage. See Fairshare Overview.

Example	`FSPOLICY  DEDICATEDPS`
	`FSDECAY  0.8`
	`FSDEPTH  8`
	Moab will apply a decay rate of 0.8 to all fairshare windows.

FSDEPTH	
Format	*<INTEGER>*
Default	*8*
Description	**Note**: The number of available fairshare windows is bounded by the MAX_FSDEPTH value (32 in Moab). See Fairshare Overview.
Example	`FSDEPTH 12`

FSENABLECAPPRIORITY	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Fairshare priority will increase to target and stop.
Example	`FSENABLECAPPRIORITY TRUE`

FSGROUPWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	
Example	`FSGROUPWEIGHT 4`

FSINTERVAL	
Format	*[[[DD:]HH:]MM:]SS*
Default	*12:00:00*
Description	Specifies the length of each fairshare window.
Example	`FSINTERVAL 12:00:00` *Track fairshare usage in 12 hour blocks.*

FSJPUWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the fairshare weight assigned to jobs per user.
Example	`FSJPUWEIGHT 10`

FSMOSTSPECIFICLIMIT	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	When checking policy usage limits in a fairshare tree, if the most specific policy limit is passed then do not check the same policy again at higher levels in the tree. For example, if a user has a MaxProc policy limit then do not check the MaxProc policy limit on the account for this same user.
Example	`FSMOSTSPECIFICLIMIT  TRUE`

FSPOLICY	
Format	*<POLICY>[*]* where *<POLICY>* is one of the following: *DEDICATEDPS*, *DEDICATEDPES*, or *UTILIZEDPS*.
Default	---
Description	Specifies the unit of tracking fairshare usage. *DEDICATEDPS* tracks dedicated processor seconds. *DEDICATEDPES* tracks dedicated processor-equivalent seconds. *UTILIZEDPS* tracks the number of utilized processor seconds. If the optional '%' (percentage) character is specified, percentage based fairshare will be used. See Fairshare Overview and Fairshare Consumption Metrics or more information.
Example	`FSPOLICY DEDICATEDPES` *Moab will track fairshare usage by dedicated process-equivalent seconds.*

FSPPUWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the fairshare weight assigned to processors per user.
Example	`FSPPUWEIGHT 10`

FSPSPUWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the fairshare weight assigned to processor-seconds per user.
Example	`FSPSPUWEIGHT 10`

FSQOSWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight assigned to the QoS fairshare subcomponent.
Example	`FSQOSWEIGHT 16`

FSTARGETISABSOLUTE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether Moab should base fairshare targets off of delivered cycles or up/available cycles.
Example	`FSTARGETISABSOLUTE TRUE`

FSTREE	
Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: SHARES or MEMBERLIST
Default	---
Description	Specifies the share tree distribution for job fairshare prioritization (see Hierarchical Share Tree Overview).
Example	`FSTREE[geo]  SHARES=16  MEMBERLIST=geo103,geo313,geo422`

FSTREEACLPOLICY	
Format	*OFF*, *PARENT*, or *FULL*
Default	*FULL*
Description	Specifies how Moab should interpret credential membership when building the FSTREE (see Hierarchical Share Tree Overview).
Example	`FSTREEACLPOLICY PARENT` *Credentials will be given access to their parent node when applicable.*

FSTREEISREQUIRED	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether a job must have an applicable node in the partition's FSTREE in order to execute within that partition (see Hierarchical Share Tree Overview).
Example	`FSTREEISREQUIRED TRUE` *Jobs must have an applicable node in the FSTREE in order to execute.*

FSTREEUSERISREQUIRED	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether the user must be given explicit access to a branch in the FSTREE (see Hierarchical Share Tree Overview).
Example	`FSTREEUSERISREQUIRED TRUE` *Users must be given explicit access to FSTREE nodes in order to gain access to the FSTREE.*

FSUSERWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight assigned to the user fairshare subfactor.
Example	`FSUSERWEIGHT 8`

FSWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the priority weight assigned to the summation of the fairshare subfactors (see Priority Factor and Fairshare overviews).
Example	`FSWEIGHT 500`

GEVENTCFG[<GEVENT>]

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is **ACTION**, **ECOUNT**, **REARM**, or **SEVERITY**. See Responding to Generic Events for details on values you can assign to each attribute.
Default	---
Description	Specifies how the scheduler should behave when various cluster events are detected. See the Generic Events Overview for more information.

Example	`GEVENTCFG[hitemp] ACTION=avoid,record,notify  REARM=00:10:00`
	`GEVENT[nodeerror] SEVERITY=3`
	If a hitemp event is detected, Moab adjusts the node allocation policy to minimize the allocation of the node. Moab also sends emails to cluster administrators and reports the event in the Moab event log.

GRESCFG[<GRES>]

Format	List of zero or more space delimited *<ATTR >=<VALUE>* pairs where *<ATTR>* is one of the following: **TYPE**, **PRIVATE**, **INHERITREQFEATURES**, **INVERTTASKCOUNT**, FEATUREGRES, or STARTDELAY
Default	---
Description	Specifies associations of generic resources into resource groups. When **PRIVATE** is set to *TRUE*, Moab puts the requested generic resource on a separate job request. By default a private request is a request with 1 task with *X* number of generic resources per task. When **INVERTTASKCOUNT***and***PRIVATE** are set to *TRUE*, Moab makes the generic resource's private request a request with *X* number of tasks with 1 generic resource per task. If **INHERITREQFEATURES** is also *TRUE*, then the private request will inherit the features of the primary request, causing Moab to place the private requests on the same nodes as the primary request. See 12.6 Managing Consumable Generic Resources for more information.

Example	`GRESCFG[scsi1] TYPE=fastio`
	`GRESCFG[scsi2] TYPE=fastio`
	`GRESCFG[scsi3] TYPE=fastio`
	The generic resources scsi1, scsi2, and scsi3 are all associated with the generic resource type fastio.

GRESTOJOBATTR

Format	Comma delimited list of generic resources
Default	---
Description	The list of generic resources will also be interpreted as JOB features. See Managing Reservations.
Example	`GRESTOJOBATTR  matlab,ccs` *Jobs which request the generic resources matlab or ccs will have a corresponding job attribute assigned to them.*

GROUPCFG[<GROUPID>]

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: General Credential Flags, **PRIORITY**, ENABLEPROFILING, **QLIST**, **QDEF**, **PLIST**, **FLAGS**, usage limits, or a fairshare usage limit specification.
Default	---
Description	Specifies group specific attributes. See the flag overview for a description of legal flag values.
Example	`GROUPCFG[staff] MAXJOB=50 QDEF=highprio` *Up to 50 jobs submitted by members of the group staff will be allowed to execute simultaneously and will be assigned the QoS highprio by default.*

GROUPWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the priority weight assigned to the specified group priority (See Credential (CRED) Factor).
Example	`GROUPWEIGHT 20`

GUARANTEEDPREEMPTION	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Causes Moab to lock PREEMPTOR jobs until JOBRETRYTIME expires (essentially, waiting for the PREEMPTEE jobs to finish). It may take some time for the PREEMPTEE jobs to clear out. During that time, the PREEMPTOR job might want to look elsewhere to run, which would be disruptive as it might preempt another set of jobs. If you wish it prevent this, it is recommended that you set GUARANTEEDPREEMPTION to **TRUE**. For related information, see About preemption, Reservation Policies, DEFERSTARTCOUNT, DEFERTIME, RESERVATIONRETRYTIME, NODEFAILURERESERVETIME, and JOBRETRYTIME.
Example	`GUARANTEEDPREEMPTION TRUE`

HALOCKCHECKTIME	
Format	[[[DD:]HH:]MM:]SS
Default	9
Description	Specifies how frequently the secondary server checks the timestamp on the lock file. See High Availability Overview for more info.
Example	`HALOCKCHECKTIME 00:00:15` *The Moab fallback server will check the health of the Moab primary server every 15 seconds.*

HALOCKUPDATETIME	
Format	[[[DD:]HH:]MM:]SS
Default	3
Description	Specifies how frequently the primary server checks the timestamp on the lock file. See High Availability Overview for more info.

HALOCKUPDATETIME

Example	`HALOCKUPDATETIME 00:00:03` *The Moab primary server will check the timestamp of the lock file every 3 seconds.*

HIDEVIRTUALNODES

Format	*<BOOLEAN>*
Default	---
Description	Enables VM management; also used to reveal hypervisors.
Example	`HIDEVIRTUALNODES TRANSPARENT`

IDCFG[X]

Format	One or more of the following attribute/value pairs: *BLOCKEDCREDLIST*, CREATECRED, CREATECREDURL, *REFRESHPERIOD*, *RESETCREDLIST* or *SERVER*.
Default	---
Description	This parameter enables the identity manager interface allowing credential, policy, and usage information to be shared with an external information service. Only one identity manager can be configured at a time.
Example	`IDCFG[info] SERVER=exec://dbquery.pl REFRESHPERIOD=hour` *Moab will refresh credential info every hour using the specified script.*

IGNOREMDATASTAGING

Format	*<BOOLEAN>*
Default	*FALSE*
Description	When set to *TRUE*, Moab will ignore any resource manager specific data staging on a job and assume the resource manager is processing the request. Currently, this only applies to PBS.
Example	`IGNORERMDATASTAGING TRUE`

IGNORECLASSES

Format	*[!]<CLASS>[,<CLASS>]...*
Default	---
Description	By default, if using the TORQUE resource manager, jobs from all listed classes are ignored and not scheduled, tracked, or otherwise processed by Moab. If the **not**(i.e., '!') character is specified, only jobs from listed classes are processed. See the Moab Side-by-Side Analysis for more information.
Example	`IGNORECLASSES dque,batch` *Moab will ignore jobs from classes dque and batch.*

IGNOREJOBS

Format	*[!]<JOBID>[,<JOBID>]...*
Default	---
Description	By default, listed jobs are ignored and not scheduled, tracked, or otherwise processed by Moab. If the **not**(i.e., '!') character is specified, only listed jobs are processed. See the Moab Side-by-Side Analysis for more information.
Example	`IGNOREJOBS !14221,14223` *Moab will ignore jobs all classes except 14221 and 14223.*

IGNORENODES

Format	*[!]<NODE>[,<NODE>]...*
Default	---
Description	By default, all listed nodes are ignored and not scheduled, tracked, or otherwise processed by Moab. If the **not**(i.e., '!') character is specified, only listed nodes are processed. See the Moab Side-by-Side Analysis for more information.
Example	`IGNORENODES !host3,host4` *Moab will only process nodes host3 and host4.*

IGNOREPREEMPTEEPRIORITY

Format	*<BOOLEAN>*
Default	*FALSE*
Description	By default, preemptor jobs can only preempt preemptee jobs if the preemptor has a higher job priority than the preemptee. When this parameter is set to true, the priority constraint is removed allowing any preemptor to preempt any preemptees once it reaches the top of the eligible job queue.
Example	`IGNOREPREEMPTEEPRIORITY TRUE` *A preemptor job can preempt any preemptee jobs when it is at the top of the eligible job queue.*

IGNOREUSERS	
Format	*[!]<USERNAME>[,<USERNAME>]...*
Default	---
Description	By default, if using the TORQUE resource manager, jobs from all listed users are ignored and not scheduled, tracked, or otherwise processed by Moab. If the **not**(i.e., '!') character is specified, only jobs from listed users are processed. (See the Moab Side-by-Side Analysis for more information.)
Example	`IGNOREUSERS testuser1,annapolis` *Moab will ignore jobs from users testuser1 and annapolis.*

#INCLUDE	
Format	*<STRING>*
Default	---
Description	Specifies another file which contains more configuration parameters. If *<STRING>* is not an absolute path, Moab will search its home directory for the file.
Example	`#INCLUDE moab.acct` *Moab will process the parameters in `moab.acct` as well as `moab.cfg`*

INSTANTSTAGE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	**Deprecated**. Use JOBMIGRATEPOLICY. Specifies whether Moab should instantly stage jobs to the underlying resource manager when a job is submitted through msub.
Example	`INSTANTSTAGE TRUE`

INVALIDFSTREEMSG

Format	"*<STRING>*"
Default	*"no valid fstree node found"*
Description	Specifies the error message that should be attached to jobs that cannot run because of a fairshare tree configuration violation.
Example	`INVALIDFSTREEMSG "account is invalid for requested partition"`

JOBACTIONONNODEFAILURE

Format	**CANCEL** on page 316, **FAIL** on page 316, **HOLD** on page 316, **IGNORE** on page 316, **NOTIFY** on page 316, or **REQUEUE** on page 316
Default	---
Description	Specifies the action to take if Moab detects that a node allocated to an active job has failed (state is down). By default, Moab only reports this information via diagnostic commands. If this parameter is set, Moab will cancel or requeue the active job. See Reallocating Resources When Failures Occur for more information. **Note**: The *HOLD* value is only applicable when using checkpointing.
Example	`JOBACTIONONNODEFAILURE REQUEUE` *Moab will requeue active jobs which have allocated nodes which have failed during the execution of the job.*

JOBAGGREGATIONTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*0*
Description	Specifies the minimum amount of time the scheduler should wait after receiving a job event until it should process that event. This parameter allows sites with *bursty* job submissions to process job events in groups decreasing total job scheduling cycles and allowing the scheduler to make more intelligent choices by aggregating job submissions and choosing between the jobs. See Considerations for Large Clusters.

Example	`JOBAGGREGATIONTIME 00:00:04`
	`RMPOLLINTERVAL 30,30`
	Moab will wait 4 seconds between scheduling cycles when job events have been received and will wait 30 seconds between scheduling cycles otherwise.

JOBCFG

Format	*<ATTR>=<VAL>* where *<ATTR>* is one of FLAGS, GRES, NODERANGE, PRIORITYF, PROCRANGE, QOS, RARCH, RFEATURES, ROPSYS, SELECT, or TARGETLOAD
Default	---
Description	Specifies attributes for jobs which satisfy the specified profile. The **SELECT** attribute allows users to specify the job template by using msub -l template=. The JOBCFG parameter supports the following attributes: **NONE**, **ACCOUNT**, **ACTION**, **AUTOSIZE**, **CLASS**, **CPULIMIT**, **DESCRIPTION**, **DGRES**, **FAILUREPOLICY**, **GROUP**, **IFLAGS**, **JOBSCRIPT MEM** (for **MEM**=*<value>*), **MEMORY** (for **MEMORY**=*$LEARN*), **NODEACCESSPOLICY**, **NODEMOD**, **PARTITION**, **PREF**, **QOS**, **RESTARTABLE**, **RM**, **RMSERVICEJOB**, **SELECT**, **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 job reported by one or more workload queries. See Wiki Attributes and Job Template Extensions.
Example	`JOBCFG[sql] RFEATURES=sqlnode QOS=service` *When the sql job is detected, it will have the specified default QoS and node feature attributes set.*

JOBCPURGETIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*00:05:00*
Description	Specifies the amount of time Moab will preserve detailed information about a completed job (see showq -c and checkjob).
Example	`JOBCPURGETIME 02:00:00` *Moab will maintain detailed job information for 2 hours after a job has completed.*

JOBCTRUNCATENLCP	
Format	*<BOOLEAN>*
Default	*TRUE*
Description	Specifies whether Moab will store only the first node of the node list for a completed job in the checkpoint file.
Example	`JOBCTRUNCATENLCP TRUE` *JOBCTRUNCATENLCP reduces the amount of memory Moab uses to store completed job information.*

JOBEXTENDSTARTWALLTIME

Format	*<BOOLEAN>*
Default	---
Description	Extends the job walltime when Moab starts the job up to the lesser of the maximum or the next reservation (rounded down to the nearest minute). JOBEXTENDSTARTWALLTIME TRUE and JOBEXTENDDURATION cannot be configured together. If they are in the same moab.cfg or are both active, then the JOBEXTENDDURATION will not be honored.

Example	`JOBEXTENDSTARTWALLTIME TRUE`
	Submit job with a minimum wallclock limit and a walltime; for example:
	`echo sleep 500 | msub -A ee -l nodes=5,minwclimit=5:00,walltime=30:00,partition=g02`
	At job start, Moab recognizes the nodes assigned to the specified job and extends the walltime for the job (one time at job start) up to the lesser of the maximum walltime requested or the least amount of time available for any of the nodes until the next reservation on that node.

JOBFAILRETRYCOUNT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the number of times a job is requeued and restarted by Moab if the job fails (if the job itself returns a non-zero exit code). Some types of jobs may succeed if automatically retried several times in short succession. This parameter was created with these types of jobs in mind. Note that the job in question must also be restartable (the job needs to have the "RESTARTABLE" flag set on it) and the RM managing the job must support requeuing and starting completed jobs. If a job fails too many times, and reaches the number of retries given by JobFailRetryCount, then a UserHold is placed on the job and a message is attached to it signifying that the job has a "restart count violation."
Example	`JOBFAILRETRYCOUNT 7` *Any job with a RESTARTABLE flag is requeued, if it fails, up to 7 times before a UserHold is placed on it.*

JOBIDWEIGHT

Format	*<INTEGER>*
Default	---
Description	Specifies the weight to be applied to the job's id. See Attribute (ATTR) Factor.
Example	`JOBIDWEIGHT -1` *Later jobs' priority will be negatively affected.*

JOBMATCHCFG

Format	*<ATTR>=<VAL>* where *<ATTR>* is one of JMIN, JMAX, JDEF, JSET, or JSTAT, or the *<ATTR>=<VAL>* pair can be **FLAGS**=*BREAK*.
Default	---
Description	Specifies the job templates which must be matched and which will be applied in the case of a match. To force Moab to break from matching, use **FLAGS**=*BREAK*. If a job matches a job template that has **FLAGS**=*BREAK* enabled, Moab stops evaluating further **JOBMATCHCFG** entries for that job.

Example	`JOBMATCHCFG[sql] JMIN=interactive JSTAT=istat`
	`JOBMATCHCFG[small] JMIN=small.min JMAX=small.max JSET.set=small.set FLAGS=BREAK`
	`JOBMATCHCFG[large] JMIN=large.min JMAX=large.max JSET=large.set`
	*In this case, the large template would not be applied when a job matches both the small and large templates. The small template matches first, and because of **FLAGS**=*BREAK*, Moab stops evaluating further **JOBMATCHFG** entries for the job.*

JOBMAXHOLDTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	---
Description	Specifies the amount of time a job can be held before it is canceled automatically.
Example	`JOBMAXHOLDTIME 02:00:00` *Moab will keep jobs in any HOLD state for 2 hours before canceling them.*

JOBMAXNODECOUNT	
Format	*<INTEGER>*
Default	*1024*
Description	Specifies the maximum number of nodes which can be allocated to a job. After changing this parameter, Moab must be restarted. **Note**: This value cannot exceed either **MMAX_NODE** or **MMAX_TASK_PER_JOB**. If larger values are required, these values must also be increased. Moab must be restarted before changes to this command will take effect. The command mdiag -S will indicate if any job node count overflows have occurred. See Consideration for Large Clusters.
Example	`JOBMAXNODECOUNT 4000`

JOBMAXOVERRUN

Format	[*[[DD:]HH:]MM:]SS*,][*[[DD:]HH:]MM:]SS*
Default	(no soft limit), *10 minutes* (hard limit)
Description	Soft and hard limit of the amount of time Moab will allow a job to exceed its wallclock limit before it first sends a mail to the primary admin (soft limit) and then terminates the job (hard limit). See WCVIOLATIONACTION or Usage-based Limits. If you run Moab with the TORQUE resource manager, you must set the $ignwalltime parameter to *true* in the `/var/spool/torque/mom_priv/config` file, otherwise the pbs_mom will kill any job that exceeds its walltime. See the TORQUE documentation for more information.
Example	`JOBMAXOVERRUN 15:00,1:00:00` *Jobs may exceed their wallclock limit by up to 1 hour, but Moab will send an email to the primary administrator when a job exceeds its walltime by 15 minutes.*

JOBMAXTASKCOUNT

Format	*<INTEGER>*
Default	*4096*
Description	Specifies the total number of tasks allowed per job.
Example	`JOBMAXTASKCOUNT 226000`

JOBMAXPREEMPTPERITERATION

Format	*<INTEGER>*
Default	*0* (No Limit)
Description	Maximum number of jobs allowed to be preempted per iteration.
Example	`JOBMAXPREEMPTPERITERATION 10`

JOBMAXSTARTPERITERATION	
Format	*<INTEGER>*
Default	0 (No Limit)
Description	Maximum number of jobs allowed to start per iteration.
Example	`JOBMAXSTARTPERITERATION 10`

JOBMAXSTARTTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*-1* (NO LIMIT)
Description	Length of time a job is allowed to remain in a 'starting' state. If a 'started' job does not transition to a running state within this amount of time, Moab will cancel the job, believing a system failure has occurred.
Example	`JOBMAXSTARTTIME 2:00:00` *Jobs may attempt to start for up to 2 hours before being canceled by the scheduler*

JOBMIGRATEPOLICY	
Format	One of the following: *IMMEDIATE*, *JUSTINTIME*, or *AUTO*
Default	*AUTO*
Description	Upon using the msub command to submit a job, you can allocate the job to immediately (*IMMEDIATE*) migrate to the resource manager, or you can instruct Moab to only migrate the job to the resource manager when it is ready to run (*JUSTINTIME*). Specifying *AUTO* allows MOAB to determine on a per-job basis whether to use *IMMEDIATE* or *JUSTINTIME*.
Example	`JOBMIGRATEPOLICY JUSTINTIME`

JOBNAMEWEIGHT

Format	*<INTEGER>*
Default	---
Description	Specifies the weight to be applied to the job's name if the Name contains an integer. See Attribute (ATTR) Factor.
Example	`JOBNAMEWEIGHT 1`

JOBNODEMATCHPOLICY

Format	*EXACTNODE* or *EXACTPROC*
Default	---
Description	Specifies additional constraints on how compute nodes are to be selected. *EXACTNODE* indicates that Moab should select as many nodes as requested even if it could pack multiple tasks onto the same node. *EXACTPROC* indicates that Moab should select only nodes with exactly the number of processors configured as are requested per node even if nodes with excess processors are available.
Example	`JOBNODEMATCHPOLICY EXACTNODE` *In a PBS/Native job with resource specification `nodes=<x>:ppn=<y>`, Moab will allocate exactly <y> task on each of <x> distinct nodes.*

JOBPREEMPTMAXACTIVETIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*0*
Description	The amount of time in which a job may be eligible for preemption. See Job Preemption.
Example	`JOBPREEMPTMAXACTIVETIME 00:05:00` *A job is preemptible for the first 5 minutes of its run time.*

JOBPREEMPTMINACTIVETIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*0*
Description	The minimum amount of time a job must be active before being considered eligible for preemption. See Job Preemption.
Example	`JOBPREEMPTMINACTIVETIME 00:05:00` *A job must execute for 5 minutes before Moab will consider it eligible for preemption.*

JOBPRIOACCRUALPOLICY

Format	*ACCRUE* or *RESET*
Default	*ACCRUE*
Description	Specifies how Moab should track the dynamic aspects of a job's priority. *ACCRUE* indicates that the job will accrue queuetime based priority from the time it is submitted unless it violates any of the policies not specified in JOBPRIOEXCEPTIONS. *RESET* indicates that it will accrue priority from the time it is submitted unless it violates any of the JOBPRIOEXCEPTIONS. However, with *RESET*, if the job does violate JOBPRIOEXCEPTIONS then its queuetime based priority will be reset to 0. JOBPRIOACCRUALPOLICY is a global parameter, but can be configured to work only in QOSCFG: `QOSCFG[arrays]  JOBPRIOACCRUALPOLICY=ACCRUE` The following old JOBPRIOACCRUALPOLICY values have been deprecated and should be adjusted to the following values: • *QUEUEPOLICY*= *ACCRUE* and JOBPRIOEXCEPTIONS*SOFTPOLICY,HARDPOLICY* • *QUEUEPOLICYRESET*= *RESET* and JOBPRIOEXCEPTIONS*SOFTPOLICY,HARDPOLICY* • *ALWAYS*= *ACCRUE* and JOBPRIOEXCEPTIONS*ALL* • *FULLPOLICY*= *ACCRUE* and JOBPRIOEXCEPTIONS*NONE* • *FULLPOLICYRESET*= *RESET* and JOBPRIOEXCEPTIONS*NONE*
Example	`JOBPRIOACCRUALPOLICY  RESET` *Moab will adjust the job's dynamic priority subcomponents, i.e., QUEUETIME, XFACTOR, and TARGETQUEUETIME, etc. each iteration that the job does not violate any JOBPRIOEXCEPTIONS, if it is found in violation, its queuetime will be reset to 0.*

JOBPRIOEXCEPTIONS

Format	Comma delimited list of any of the following: *DEFER*, *DEPENDS*, *SOFTPOLICY*, *HARDPOLICY*, *IDLEPOLICY*, *USERHOLD*, *BATCHHOLD*, and *SYSTEMHOLD* (*ALL* or *NONE* can also be specified on their own)
Default	*NONE*
Description	Specifies exceptions for calculating a job's dynamic priority (QUEUETIME, XFACTOR, TARGETQUEUETIME). Normally, when a job violates a policy, is placed on hold, or has an unsatisfied dependency, it will not accrue priority. Exceptions can be configured to allow a job to accrue priority in spite of any of these violations. With DEPENDS a job will increase in priority even if there exists an unsatisfied dependency. With *SOFTPOLICY*, *HARDPOLICY*, or *IDLEPOLICY* a job can accrue priority despite violating a specific limit. With *DEFER*, *USERHOLD*, *BATCHHOLD*, or *SYSTEMHOLD* a job can accrue priority despite being on hold. **JOBPRIOEXCEPTIONS** is a global parameter, but can be configured to work only in QOSCFG: `QOSCFG[arrays]  JOBPRIOEXCEPTIONS=IDLEPOLICY`
Example	`JOBPRIOEXCEPTIONS BATCHHOLD,SYSTEMHOLD,DEPENDS` *Jobs will accrue priority in spite of batchholds, systemholds, or unsatisfied dependencies.*

JOBPRIOF

Format	*<ATTRIBUTE>[<VALUE>]=<PRIORITY>* where *<ATTRIBUTE>* is one of **ATTR**, **GRES** or **STATE**
Default	---
Description	Specifies attribute priority weights for jobs with specific attributes, generic resource requests, or states. State values must be one of the standard Moab job states. See Attribute-Based Job Prioritization.

Example	`JOBPRIOF  STATE[Running]=100  STATE[Suspended]=1000  ATTR[PREEMPTEE]=200`
	`GRES[biocalc]=5`
	`ATTRATTRWEIGHT  1`
	`ATTRSTATEWEIGHT  1`
	Moab will adjust the job's dynamic priority subcomponents.

JOBPURGETIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*0* (purge immediately if the resource manager does not report the job)
Description	The amount of time Moab will keep a job record which is no longer reported by the resource manager. Useful when using a resource manager which *drops* information about a job due to internal failures. See JOBCPURGETIME.
Example	`JOBPURGETIME 00:05:00` *Moab will maintain a job record for 5 minutes after the last update regarding that object received from the resource manager.*

JOBREJECTPOLICY

Format	One or more of *CANCEL*, *HOLD*, *IGNORE* (beta), *MAIL*, or *RETRY*
Default	*HOLD*
Description	Specifies the action to take when the scheduler determines that a job can never run. *CANCEL* issues a call to the resource manager to cancel the job. *HOLD* places a *batch* hold on the job preventing the job from being further evaluated until released by an administrator. (**Note**: Administrators can dynamically alter job attributes and possibly *fix* the job with mjobctl -m.) With *IGNORE*(currently in beta), the scheduler will allow the job to exist within the resource manager queue but will neither process it nor report it. MAIL will send email to both the admin and the user when rejected jobs are detected. If *RETRY* is set, then Moab will allow the job to remain idle and will only attempt to start the job when the policy violation is resolved. Any combination of attributes may be specified. See QOSREJECTPOLICY.
Example	`JOBREJECTPOLICY  MAIL,CANCEL`

JOBREMOVEENVVARLIST

Format	Comma-delimited list of strings
Default	---
Description	Moab will remove the specified environment variables from the job's environment before migrating the job to its destination resource manager. This is useful when jobs submit themselves from one cluster to another with the full environment. This parameter is currently only supported with TORQUE resource managers.
Example	`JOBREMOVEENVVARLIST PBS_SERVER,TZ` *Moab will remove the environment variables PBS_SERVER and TZ before submitting jobs.*

JOBRETRYTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*00:00:60*
Description	Period of time Moab will continue to attempt to start a job which has failed to start due to transient failures or which has successfully started and was then rejected by the resource manager due to transient failures. (For related information, see Reservation Policies, DEFERSTARTCOUNT, DEFERTIME, RESERVATIONRETRYTIME, NODEFAILURERESERVETIME, and GUARANTEEDPREEMPTION.)
Example	`JOBRETRYTIME   00:05:00` *Moab will try for up to 5 minutes to restart jobs if the job start has failed due to transient errors.*

LIMITEDJOBCP	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether there should be limited job checkpointing (see Consideration for Large Clusters). With LIMITEDJOBCP enabled, Moab will only checkpoint a job if it is modified with **mjobctl** on page 132 or if it has been submitted with **msub** on page 203 but has not migrated. In all other cases, Moab does not checkpoint the job and all Moab-specific information (such as messages attached to the job) is lost. No TORQUE-specific information will be lost.
Example	`LIMITEDJOBCP TRUE` *Moab will only maintain scheduler checkpoint information for jobs with explicitly modified job attributes. Some minor job performance and usage statistics may be lost.*

LIMITEDNODECP	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether there should be limited node checkpointing (see Consideration for Large Clusters).
Example	`LIMITEDNODECP TRUE` *Moab will only maintain scheduler checkpoint information for nodes with explicitly modified job attributes. (some minor node performance and usage statistics may be lost)*

LOADALLJOBCP	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether Moab should load, during startup, all non-completed jobs in the checkpoint files regardless of whether or not their corresponding resource managers are active. For example, this allows source peers to continue showing remote jobs in the queue based on checkpointed info, even though the destination peer is offline.
Example	`LOADALLJOBCP TRUE` *Moab will load, at startup, all non-completed jobs from all checkpoint files.*

LOCKFILE	
Format	*<STRING>*
Default	---
Description	Specifies the path for the lock (pid) file used by Moab.
Example	`LOCKFILE /var/spool/moab/lock`

LOGDIR	
Format	*<STRING>*
Default	log
Description	Specifies the directory in which log files will be maintained. If specified as a relative path, LOGDIR will be relative to *$(MOABHOMEDIR)* See Logging Overview for more information.
Example	`LOGDIR /var/spool/moab` *Moab will record its log files directly into the `/var/spool/moab` directory*

LOGFACILITY	
Format	Colon delimited list of one or more of the following: *CORE*, *SCHED*, *SOCK*, *UI*, *LL*, *CONFIG*, *STAT*, *SIM*, *STRUCT*, *FS*, *CKPT*, *BANK*, *RM*, *PBS*, *WIKI*, *ALL*
Default	*ALL*
Description	Specifies which types of events to log (see Logging Overview).
Example	`LOGFACILITY RM:PBS` *Moab will log only events involving general resource manager or PBS interface activities.*

LOGFILE	
Format	*<STRING>*
Default	*moab.log*
Description	Name of the Moab log file. This file is maintained in the directory pointed to by *<LOGDIR>* unless *<LOGFILE>* is an absolute path (see Logging Overview)
Example	`LOGFILE moab.test.log` *Log information will be written to the file moab.test.log located in the directory pointed to by the LOGDIR parameter.*

LOGFILEMAXSIZE	
Format	*<INTEGER>*
Default	*10000000*
Description	Maximum allowed size (in bytes) of the log file before it will be rolled (see Logging Overview).
Example	`LOGFILEMAXSIZE 50000000` *Log files will be rolled when they reach 50 MB in size*

LOGFILEROLLDEPTH	
Format	*<INTEGER>*
Default	*3*
Description	Number of old log files to maintain (i.e., when full, `moab.log` will be renamed `moab.log.1`, `moab.log.1` will be renamed `moab.log.2`, ...). See Logging Overview.
Example	`LOGFILEROLLDEPTH 5` *Moab will maintain and roll the last 5 log files.*

LOGLEVEL	
Format	*<INTEGER>* (0-9)
Default	*0*
Description	Specifies the verbosity of Moab logging where 9 is the most verbose (**Note**: each logging level is approximately an order of magnitude more verbose than the previous level). See Logging Overview.
Example	`LOGLEVEL 4` *Moab will write all Moab log messages with a threshold of 4 or lower to the `moab.log` file.*

LOGLEVELOVERRIDE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	When this parameter is on, if someone runs a command with `--loglevel=<x>`, that loglevel, if higher than the current loglevel, is used on the scheduler side for the duration of the command. All logs produced during that time are put into a separate log file (this creates a "gap" in the normal logs). This can be very useful for debugging, but it is recommend that this be used only when diagnosing a specific problem so that users can't affect performance by submitting multiple `--loglevel` commands. This parameter does not work with threaded commands (such as showq, mdiag -n, and mdiag -j).
Example	`LOGLEVELOVERRIDE TRUE`

LOGPERMISSIONS	
Format	*<INTEGER>*
Default	*644*
Description	Specifies the octal number that represents read, write, and execute permissions.
Example	`LOGPERMISSIONS 600` *Allows the file owner to read and write permissions, but denies rights to the group and others.*

LOGROLLACTION

Format	*<STRING>*
Default	---
Description	Specifies a script to run when the logs roll. The script is run as a trigger and can be viewed using mdiag -T. For example, a script can be specified that always moves the first rolled log file, `moab.-log.1`, to an archive directory for longer term storage.
Example	`LOGROLLACTION /usr/local/tools/logroll.pl`

MAILPROGRAM

Format	[*<Full_Path_To_Mail_Command>* | *DEFAULT* | *NONE*][@*<DEFAULTMAILDOMAIN>*]
Default	*NONE*
Description	If set to *NONE*, no mail is sent. If set to *DEFAULT*, Moab sends mail via the system's default mail program (usually `/usr/bin/sendmail`). If set to the local path of a mail program, Moab uses the specified mail program to send mail. By default, Moab mail notification is disabled. To enable, you must set MAILPROGRAM to *DEFAULT* or specify some other locally available mail program. If the *default mail domain* is set, emails will be routed to this domain unless a per-user domain is specified using the **EMAILADDRESS** attribute of the USERCFG parameter. If neither of these values is set, Moab uses "@localhost" as the mail domain. See Notify Admins. For jobs, the email address used on the msub -M option overrides all other user email addresses. Additionally, administrators are notified in the case of job violations.

Example	`MAILPROGRAM DEFAULT`
	Moab sends mail via the system's default mail program, `/usr/bin/sendmail`.
	`MAILPROGRAM /usr/local/bin/sendmail@mydomain.com`
	Moab sends mail via the mail program located at /usr/local/bin/sendmail with default mail domain @mydomain.com

MAXGRES	
Format	*<INTEGER>*
Default	*512*
Description	Specifies how many generic resources Moab should manage.
Example	`MAXGRES 1024`

MAXGMETRIC	
Format	*<INTEGER>*
Default	*10*
Description	Specifies how many generic metrics Moab should manage.
Example	`MAXGMETRIC 20`

MAXJOB	
Format	*<INTEGER>*
Default	*4096*
Description	Specifies the maximum quantity of jobs for which Moab should allocate memory used for tracking jobs. If Moab is tracking the maximum quantity of jobs specified by this parameter, it rejects subsequent jobs submitted by any user since it has no memory left with which to track newly submitted jobs. If a user submitted a job with the msub command, this rejection behavior requires the user to resubmit the job at a later time after other jobs have completed, which frees memory in which Moab can place later-submitted jobs. If a user submitted a job with the TORQUE qsub command, TORQUE will automatically resubmit the job to Moab until Moab accepts it. The mdiag -S command indicates if any job overflows have occurred. If this parameter's value is changed, it does not go into effect until Moab restarts. Moab reads the parameter only on initial startup and uses its value to allocate the memory it uses to track jobs.
Example	`MAXJOB 45000`

MAXNODE	
Format	*<INTEGER>*
Default	*5120*
Description	Specifies the maximum number of compute nodes supported.
Example	`MAXNODE 10000`

MAXRSVPERNODE	
Format	*<INTEGER>*
Default	*48*
Description	Specifies the maximum number of reservations on a node. For large SMP systems (>512 processors/node), Adaptive Computing advises adjusting the value to approximately twice the average sum of admin, standing, and job reservations present. A second number, led by a comma, can also be specified to set a maximum number of reservations for nodes that are part of the SHARED partition. The maximum possible value of MAXRSVPERNODE is *8192* for a global node and *4096* for any other node. Moab must be restarted for any changes to this parameter to take effect. The command mdiag -S indicates whether any node reservation overflows have occurred. See Considerations for Large Clusters. ⚠ Do not lower the MAXRSVPERNODE value while there are active jobs in the queue. This can lead to queue instability and certain jobs could become stuck or disconnected from the system.

Example	`MAXRSVPERNODE 64`
	64 is the maximum number of reservations on a single node.
	
	`MAXRSVPERNODE 100,7000`
	100 is the maximum number of reservations on a single node, and 7000 is the maximum number of reservations for global nodes.

MEMREFRESHINTERVAL

Format	*[[[DD:]HH:]MM:]:SS* | job:*<COUNT>*
Default	---
Description	Specifies the time interval or total job query count at which Moab will perform garbage collection to free memory associated with resource manager API's which possess memory leaks (i.e., Loadleveler, etc.).

Example	`# free memory associated with leaky RM API`
	`MEMREFRESHINTERVAL 24:00:00`
	Moab will perform garbage collection once every 24 hours.

MEMWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the coefficient to be multiplied by a job's MEM (dedicated memory in MB) factor. See Resource Priority Overview.

Example	`RESWEIGHT 10`
	`MEMWEIGHT 1000`
	*Each job's priority will be increased by 10 * 1000 * <request memory>.*

MINADMINSTIME

Format	*<INTEGER>*
Default	*60* seconds
Description	Specifies the minimum time a job will be suspended if suspended by an administrator or by a scheduler policy.
Example	`MINADMINSTIME 00:10:00` *Each job suspended by administrators or policies will stay in the suspended state for at least 10 minutes.*

MISSINGDEPENDENCYACTION

Format	*CANCEL*, *HOLD*, or *RUN*
Default	*HOLD*
Description	Controls what Moab does with a dependent job when its dependency job cannot be found when Moab evaluates the dependent job for scheduling. This only affects jobs whose dependent job cannot be found.
Example	`MISSINGDEPENDENCYACTION CANCEL` *Any job that has a dependent job that cannot be found is canceled.*

MSUBQUERYINTERVAL

Format	*<INTEGER>*
Default	*5* seconds
Description	Specifies the length of the interval (in seconds) between job queries when using msub -K. Jobs submitted with the -K option query the scheduler every MSUBQUERYINTERVAL seconds until the job is completed. MSUBQUERYINTERVAL can exist as an environment variable. Any value in `moab.cfg` overrides the environment variable. **Note**: If MSUBQUERYINTERVAL is set to 0, the -K option will be disabled. Jobs will still submit correctly, but the client will not continue to check on the job.
Example	`MSUBQUERYINTERVAL 60` *If a user uses the msub -K command, the client remains open and queries the server every 60 seconds until the job completes.*

NODEACCESSPOLICY

Format	One of the following: SHARED, SHAREDONLY, SINGLEJOB, SINGLETASK, SINGLEUSER, or UNIQUEUSER
Default	**SHARED**
Description	Specifies how node resources will be shared by various tasks (See the Node Access Overview for more information).
Example	`NODEACCESSPOLICY SINGLEUSER` Moab will allow resources on a node to be used by more than one job provided that the jobs are all owned by the same user.

NODEALLOCATIONPOLICY

Format	One of the following: FIRSTAVAILABLE, LASTAVAILABLE, MINRESOURCE, CPULOAD, LOCAL, CONTIGUOUS, MAXBALANCE, PRIORITY, or PLUGIN.
Default	*LASTAVAILABLE*
Description	Specifies how Moab should allocate available resources to jobs. See Node Allocation Overview for more information. If **ENABLEHIGHTHROUGHPUT** on page 824 is *TRUE*, you must set NODEALLOCATIONPOLICY to *FIRSTAVAILABLE*.
Example	`NODEALLOCATIONPOLICY MINRESOURCE` *Moab will apply the node allocation policy MINRESOURCE to all jobs by default.*

NODEALLOCRESFAILUREPOLICY

Format	One of the following: *CANCEL*, *HOLD*, *IGNORE*, *MIGRATE*, *NOTIFY*, or *REQUEUE*
Default	*NONE*
Description	Specifies how Moab should handle active jobs which experience node failures during execution. See the RESFAILPOLICY resource manager extension or the Node Availability Overview.
Example	`NODEALLOCRESFAILUREPOLICY REQUEUE` *Moab will requeue jobs which have allocated nodes fail during execution.*

NODEAVAILABILITYPOLICY

Format	*<POLICY>[:<RESOURCETYPE>]* ... where *<POLICY>* is one of **COMBINED**, **DEDICATED**, or **UTILIZED** and *<RESOURCETYPE>* is one of *PROC*, *MEM*, *SWAP*, or *DISK*
Default	**COMBINED**
Description	Specifies how available node resources are reported. Moab uses the following calculations to determine the amount of available resources: **Dedicated**(use what Moab has scheduled to be used): Available = Configured - Dedicated **Utilized**(use what the resource manager is reporting is being used): Available = Configured - Utilized **Combined**(use the larger of dedicated and utilized): Available = Configured - (MAX(Dedicated,Utilized)) Moab marks a node as busy when it has no available processors, so NODEAVAILABILTYPOLICY, by affecting how many processors are reported as available, also affects node state. See Node Availability Policies for more information.
Example	`NODEAVAILABILITYPOLICY DEDICATED:PROCS COMBINED:MEM` *Moab will ignore resource utilization information in locating available processors for jobs but will use both dedicated and utilized memory information in determining memory availability.*

NODEBUSYSTATEDELAYTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*0:01:00* (one minute)
Description	Length of time Moab will assume busy nodes will remain unavailable for scheduling if a system reservation is not explicitly created for the node.
Example	`NODEBUSYSTATEDELAYTIME 0:30:00` *Moab will assume busy nodes are not available for scheduling for at least 30 minutes from the current time. Thus, these nodes will never be allocated to starting jobs. Also, these nodes will only be available for reservations starting more than 30 minutes in the future.*

NODECATCREDLIST

Format	*<LABEL>=<NODECAT>[,<NODECAT>]...[<LABEL>=<NODECAT>[,<NODECAT>]...]...* where *<LABEL>* is any string and *<NODECAT>* is one of the defined node categories.
Default	---
Description	If specified, Moab will generate node category groupings and each iteration will assign usage of matching resources to pseudo-credentials with a name matching the specified label. See the Node Categorization section of the Admin manual for more information.
Example	`NODECATCREDLIST down=BatchFailure,HardwareFailure,NetworkFailure idle=Idle` *Moab will create a **down** user, group, account, class, and QoS and will associate BatchFailure, HardwareFailure, and NetworkFailure resources with these credentials. Additionally, Moab will assign all Idle resources to matching **idle** credentials.*

NODECFG[X]

Format	List of space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: ACCESS, CHARGERATE, FEATURES, FLAGS, GRES, LOGLEVEL, MAXJOB, MAXJOBPERUSER, MAXLOAD, MAXPE, NODEINDEX, NODETYPE, OSLIST, PARTITION, **POWERPOLICY** on page 477, PRIORITY, PRIORITYF, PROCSPEED, RACK, RADISK, SLOT, SPEED, or TRIGGER
Default	---
Description	Specifies node-specific attributes for the node indicated in the array field. See the General Node Administration Overview for more information.
Example	`NODECFG[nodeA] MAXJOB=2 SPEED=1.2` *Moab will only allow 2 simultaneous jobs to run on node nodeA and will assign a relative machine speed of 1.2 to this node.*

NODEDOWNSTATEDELAYTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*-1* (never)
Description	Length of time Moab will assume down, drained(offline), or corrupt nodes will remain unavailable for scheduling if a system reservation is not explicitly created for the node. The default specification of "-1" causes Moab to never create job reservations on down nodes. See Node Availability for more information.
Example	`NODEDOWNSTATEDELAYTIME 0:30:00` *Moab will assume down, drained, and corrupt nodes are not available for scheduling for at least 30 minutes from the current time. Thus, these nodes will never be allocated to starting jobs. Also, these nodes will only be available for reservations starting more than 30 minutes in the future.*

NODEDOWNTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	---
Description	The maximum time a previously reported node remains unreported by a resource manager before the node is considered to be in the down state. This can happen if communication with a resource manager or a peer server is lost for more than the specified length of time, or if there is communication with the resource manager but it fails to report the node status.
Example	`NODEDOWNTIME 10:00` *If Moab loses communication with the resource manager for more than 10 minutes, it sets the state of all nodes belonging to that resource manager to DOWN.*

NODEDRAINSTATEDELAYTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*3:00:00* (three hours)
Description	Length of time Moab will assume drained nodes will remain unavailable for scheduling if a system reservation is not explicitly created for the node. Specifying "-1" will cause Moab to never create job reservations on drained nodes. See Node Availability for more information.
Example	`NODEDRAINSTATEDELAYTIME 0:30:00` *Moab will assume down, drained, and corrupt nodes are not available for scheduling for at least 30 minutes from the current time. Thus, these nodes will never be allocated to starting jobs. Also, these nodes will only be available for reservations starting more than 30 minutes in the future.*

NODEFAILURERESERVETIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*0:05:00*
Description	Duration of reservation Moab will place on any node in which it detects a failure from the resource manager (0 indicates no reservation will be placed on the node). See Node Availability for more information. See also RMCFG[] NODEFAILURERSVPROFILE. (For related information, see Reservation Policies, DEFERSTARTCOUNT, DEFERTIME, RESERVATIONRETRYTIME, JOBRETRYTIME, and GUARANTEEDPREEMPTION.)
Example	`NODEFAILURERESERVETIME 10:00` *Moab will reserve failed nodes for 10 minutes.*

NODEIDFORMAT

Format	*<STRING>*
Default	**$N**
Description	Specifies how a node id can be processed to extract possible node, rack, slot, and cluster index information. The value of the parameter may include the markers *$C* (cluster index), *$N* (node index), *$R* (rack index), or *$S* (slot index) separated by *(asterisk - representing any number of non-numeric characters) or other characters to indicate this encoding. See Node Selection for more information on use of node, rack, and slot indices.
Example	`NODEIDFORMAT *$R*$S` *Moab will extract rack and slot information from the cluster node ids (i.e. `tg-13s08`).*

NODEMAXLOAD

Format	*<DOUBLE>*
Default	*0.0*
Description	Specifies that maximum cpu load on an idle or running node. If the node's load reaches or exceeds this value, Moab will mark the node busy.
Example	`NODEMAXLOAD 0.75` *Moab will adjust the state of all idle and running nodes with a load >= .75 to the state busy.*

NODEMEMOVERCOMMITFACTOR

Format	*<DOUBLE>*
Default	---
Description	The parameter overcommits available and configured memory and swap on a node by the specified factor (for example: mem/swap * factor). Used to show that the node has more mem and swap than it really does. Only works for PBS RM types.
Example	`NODEMEMOVERCOMMITFACTOR .5` *Moab will overcommit the memory and swap of the node by a factor of 0.5.*

NODEPOLLFREQUENCY

Format	*<INTEGER>*
Default	*0* (Poll Always)
Description	Specifies the number of scheduling iterations between scheduler initiated node manager queries. If set to ' -2, Moab will never query the node manager daemons. If set to ' -1', Moab will only query on the first iteration. **Note**: this parameter is most often used with OpenPBS and PBSPro. It is not required when using TORQUE, LoadLeveler, LSF, or SGE as the resource managers.
Example	`NODEPOLLFREQUENCY 5` *Moab will update node manager based information every 5 scheduling iterations.*

NODESETATTRIBUTE

Format	*FEATURE*
Default	---
Description	Specifies the type of node attribute by which node set boundaries will be established. See Node Set Overview.

Example	`NODESETPOLICY ONEOF`
	`NODESETATTRIBUTE FEATURE`
	Moab will create node sets containing nodes with common features.

NODESETDELAY

Format	*[[[DD:]HH:]MM:]SS*
Default	*0:00:00*
Description	Causes Moab to attempt to span a job evenly across node sets unless doing so delays the job beyond the requested NODESETDELAY. Must use with **NODESETPLUS** on page 881 set to *SPANEVENLY*; if you do not want to use *SPANEVENLY*, use **NODESETISOPTIONAL** on page 880 instead of NODESETDELAY.

Example	`NODESETPLUS SPANEVENLY`
	`NODESETDELAY 5:00`
	Moab tries to span the job evenly across node sets unless doing so delays the job by 5 minutes.

NODESETISOPTIONAL

Format	*<BOOLEAN>*
Default	*TRUE*
Description	Specifies whether or not Moab will start a job if a requested node set cannot be satisfied. See Node Set Overview.
Example	`NODESETISOPTIONAL TRUE` *Moab will not block a job from running if its node set cannot be satisfied.*

NODESETLIST

Format	*<ATTR>[{ :,|}<ATTR>]...*
Default	---
Description	Specifies the list of node attribute values which will be considered for establishing node sets. See Node Set Overview.

Example	`NODESETPOLICY ONEOF`
	`NODESETATTRIBUTE FEATURE`
	`NODESETLIST switchA,switchB`
	Moab will allocate nodes to jobs either using only nodes with the switchA feature or using only nodes with the switchB feature.

NODESETPLUS

Format	*DELAY* or *SPANEVENLY*
Default	---
Description	Specifies how Moab distributes jobs among node sets. See Node Set Overview. This parameter will not work with multi-req jobs or preemption.
Example	`NODESETPLUS SPANEVENLY` *Moab attempts to fit all jobs on a single node set or to span them evenly across a number of node sets, unless doing so would delay a job beyond the requested NODESETDELAY.*

NODESETPOLICY

Format	*ANYOF*, *FIRSTOF*, or *ONEOF*
Default	---
Description	Specifies how nodes will be allocated to the job from the various node set generated. See Node Set Overview.

Example	`NODESETPOLICY ONEOF`
	`NODESETATTRIBUTE NETWORK`
	Moab will create node sets containing nodes with common network interfaces.

NODESETPRIORITYTYPE

Format	one of *AFFINITY*, *BESTFIT*, *WORSTFIT*, or *MINLOSS*
Default	*MINLOSS*
Description	Specifies how resource sets will be selected when more than one feasible resource can be found. See Node Set Overview.

Example	`NODESETPRIORITYTYPE BESTFIT`
	`NODESETATTRIBUTE PROCSPEED`
	Moab will select the resource set that most closely matches the resources requested.

NODESYNCTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*00:10:00*
Description	Specifies the length of time after which Moab will sync up a node's expected state with an unexpected reported state. **IMPORTANT Note**: Moab will not start new jobs on a node with an expected state which does not match the state reported by the resource manager.
Example	`NODESYNCTIME 1:00:00`

NODETOJOBATTRMAP

Format	Comma delimited list of node features
Default	---
Description	Job requesting the listed node features will be assigned a corresponding job attribute. These job attributes can be used to enable reservation access, adjust job priority or enable other capabilities.
Example	`NODETOJOBATTRMAP  fast,big` *Jobs requesting node feature fast or big will be assigned a corresponding job attribute.*

NODEUNTRACKEDRESDELAYTIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*0:00:00*
Description	Length of time Moab will assume untracked generic resources will remain unavailable for scheduling if a system reservation is not explicitly created for the node. If NODEUNTRACKEDRESDELAYTIME is enabled and there is an untracked resource preventing a job from running, then the job remains in the idle queue instead of being deferred.
Example	`NODEUNTRACKEDRESDELAYTIME 0:30:00` *Moab will assume untracked generic resources are not available for scheduling for at least 30 minutes from the current time. Thus, these nodes will never be allocated to starting jobs. Also, these nodes will only be available for reservations starting more than 30 minutes in the future.*

NODEWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight which will be applied to a job's requested node count before this value is added to the job's cumulative priority. **Note**: this weight currently only applies when a nodecount is specified by the user job. If the job only specifies tasks or processors, no node factor will be applied to the job's total priority. This will be rectified in future versions.
Example	`NODEWEIGHT 1000`

NOLOCALUSERENV

Format	*<BOOLEAN>*
Default	*FALSE*
Description	If *TRUE*, specifies that a user's UserID, GroupID, and HomeDirectory are available on the Moab server host.
Example	`NOLOCALUSERENV TRUE`

NOJOBHOLDNORESOURCES

Format	*<BOOLEAN>*
Default	*FALSE*
Description	If *TRUE*, Moab does not place a hold on jobs that don't have feasible resources. For example, suppose there are 20 processors available for ClassA and 50 processors for the entire system. If a job requests 21 or more processors from ClassA, or 51 or more processors from the entire system, Moab idles the job (instead of putting a hold on it) until the resources become available.
Example	`NOJOBHOLDNORESOURCES TRUE`

NOTIFICATIONPROGRAM	
Format	*<STRING>*
Default	---
Description	Specifies the name of the program to handle all notification call-outs.
Example	`NOTIFICATIONPROGRAM  tools/notifyme.pl`

NOWAITPREEMPTION	
Format	*<BOOLEAN>*
Default	---
Description	Generally when a job is trying to preempt another, it just waits for the original jobs that it chose to preempt to end. If this parameter is on, the preemptor will continue trying to preempt jobs until it can get in.
Example	`NOWAITPREEMPTION  TRUE`

OSCREDLOOKUP

Format	NEVER
Default	---
Description	Disables all Moab OS credential lookups, including UID, GID, user to group mappings, and any other OS specific information. Setting OSCREDLOOKUP by itself does not allow job submission; additional configuration is required. When submitting jobs from user accounts that do not exist on the head node (where Moab Workload Manager and TORQUE are running), you must also set the PROXYJOBSUBMISSION flag in addition to specifying configuration settings in the resource manager configuration file. See the example that follows for information on required resource manager settings.

Example	`OSCREDLOOKUP NEVER`
	`RMCFG[] FLAGS=PROXYJOBSUBMISSION`
	To allow job submission, in the TORQUE configuration file (torque.cfg):
	`VALIDATEPATH FALSE`
	Run the following qmgr directive:
	`set server disable_server_id_check = True`
	Restart both Moab Workload Manager and pbs_server.

PARALLOCATIONPOLICY

Format	One of *BestFit*, *BestFitP*, *FirstStart*, *LoadBalance*, *LoadBalanceP*, *Random*, or *RoundRobin*
Default	*FirstStart*
Description	Specifies the approach to use to allocate resources when more than one eligible partition can be found.
Example	`PARALLOCATIONPOLICY  LOADBALANCE` *New jobs will be started on the most lightly allocated partition.*

PARCFG	
Format	NODEPOWEROFFDURATION, NODEPOWERONDURATION, NODEALLOCATIONPOLICY or one or more key-value pairs as described in the Partition Overview
Default	---
Description	Specifies the attributes, policies, and constraints for the given partition.
Example	`PARCFG[oldcluster] MAX.WCLIMIT=12:00:00` *Moab will not allow jobs to run on the oldcluster partition which has a wallclock limit in excess of 12 hours.*

PBSACCOUNTINGDIR	
Format	*<PATH>*
Default	---
Description	When specified, Moab will write out job events in standard PBS/ TORQUEtracejob format to the specified directory using the standard PBS/TORQUE log file naming convention.
Example	`PBSACCOUNTINGDIR /var/spool/torque/sched_logs/` *Job events will be written to the specified directory (can be consumed by PBS's tracejob command).*

PERPARTITIONSCHEDULING	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	By default Moab's scheduling routine schedules each job on each partition using the following algorithm: *prioritize* *foreach (job)* *find the partition on which that job should run* *schedule job* In this model, a job's priority is the same on each partition as it uses a single global priority. Because a job's priority is the same on every partition, Moab prioritizes the queue once and then schedules the prioritized queue across all partitions. When `PERPARTITIONSCHEDULING TRUE` is set, the following algorithm is used: *foreach (partition)* *prioritize* *foreach (job)* *schedule job* In this case, each partition may have a unique priority configuration and Moab will re-prioritize the jobs for each partition on the system. Each job is prioritized and scheduled on each partition. See **PARCFG** on page 887 for more information. Also, note that Moab will order the partitions as they are discovered in the moab.cfg file. Partitions should be explicitly ordered via PARCFG in the moab.cfg file.

Example	`PERPARTITIONSCHEDULING TRUE`
	`PARCFG[p1] CONFIGFILE=/opt/moab/etc/p1.cfg`
	`PARCFG[p2] CONFIGFILE=/opt/moab/etc/p2.cfg`
	Rather than prioritizing the job queue once, Moab prioritizes the job queue for each partition, p1 and p2 respectively, and schedules each partition in turn using the policies located in their respective configuration files. (See **Per-Partition Settings** on page 422 *for more information).*

PEWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the coefficient to be multiplied by a job's PE (processor equivalent) priority factor.

Example	`RESWEIGHT 10`
	`PEWEIGHT  100`
	*Each job's priority will be increased by 10 * 100 * its PE factor.*

PREEMPTPOLICY	
Format	one of the following: CANCEL, REQUEUE, SUSPEND, or CHECKPOINT
Default	*REQUEUE*
Description	Specifies how preemptible jobs will be preempted. **Note:** If this policy is set to REQUEUE, preemptible jobs should be marked as *RESTARTABLE*. If this policy is set to SUSPEND, preemptible jobs should be marked as *SUSPENDABLE*. **Note:** Moab uses preemption escalation to preempt resources if the specified preemption facility is not applicable. This means if the policy is set to *SUSPEND* and the job is not *SUSPENDABLE*, Moab may attempt to requeue or even cancel the job.
Example	`PREEMPTPOLICY CHECKPOINT` *Jobs that are to be preempted will be checkpointed and restarted at a later time.*

PREEMPTPRIOJOBSELECTWEIGHT	
Format	*<DOUBLE>*
Default	*256.0*
Description	Determines which jobs to preempt based on size or priority. The higher the value, the more emphasis is placed on the priority of the job, causing the lower priority jobs to be preempted first. The lower the value, the more emphasis is placed on the size of the job, causing the smaller jobs to be preempted first. If set to *0*, job priority will be ignored, job size will take precedence and the smallest jobs will be preempted. The special setting of *-1* places the emphasis solely on resource utilization. This means that jobs will be preempted in a manner that keeps the resource utilization at the highest level, regardless of job priority or size.
Example	`PREEMPTPRIOJOBSELECTWEIGHT 220.5`

PREEMPTRTIMEWEIGHT	
Format	*<DOUBLE>*
Default	*0*
Description	If set to anything other than *0*, a job's remaining time is added into the calculation of which jobs will be preempted. If a positive weight is specified, jobs with a longer remaining time are favored. If a negative weight is specified, jobs with a shorter remaining time are favored.
Example	`PREEMPTRTWEIGHT 1.5`

PREEMPTSEARCHDEPTH

Format	*<INTEGER>*
Default	*unlimited*
Description	Specifies how many preemptible jobs will be evaluated as potential targets for serial job preemptors. See Preemption Overview for more information.
Example	`PREEMPTSEARCHDEPTH 8` *Serial job preemptors will only consider the first 8 feasible preemptee jobs when determining the best action to take.*

PRIORITYTARGETDURATION

Format	*[[[DD:]HH:]MM:]SS*
Default	---
Description	Specifies the *ideal* job duration which will maximize the value of the WALLTIMEWEIGHT priority factor. If specified, this factor will be calculated as the distance from the ideal. Consequently, in most cases, the associated subcomponent weight should be set to a negative value.
Example	`WALLTIMEWEIGHT         -2500` `PRIORITYTARGETDURATION  1:00:00`

PRIORITYTARGETPROCCOUNT

Format	*<INTEGER>*{+|-|%}
Default	---
Description	Specifies the ideal job requested proc count which will maximize the value of the PROCWEIGHT priority factor. If specified, this factor will be calculated as the distance from the ideal (proc count - ideal = coefficient of PROCWEIGHT). Consequently, in most cases, the associated subcomponent weight should be set to a negative value.
Example	`PROCWEIGHT              -1000` `PRIORITYTARGETPROCCOUNT  64`

PROCWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the coefficient to be multiplied by a job's requested processor count priority factor.
Example	`PROCWEIGHT 2500`

PROFILECOUNT	
Format	*<INTEGER>*
Default	*600*
Description	Specifies the number of statistical profiles to maintain. PROFILECOUNT must be set high enough that at least one day of statistics is maintained. The statistics time window can be determined by measuring PROFILEDURATION * PROFILECOUNT. If PROFILEDURATION is one hour then PROFILECOUNT must be at least 24 so 24 hours worth of statistics are maintained. If PROFILEDURATION is 30:00 then PROFILECOUNT must be set to at least 48. If PROFILECOUNT is not high enough for at least one day of statistics, Moab adjusts it automatically.
Example	`PROFILECOUNT 300`

PROFILEDURATION	
Format	*[[[DD:]HH:]MM:]SS*
Default	*00:30:00*
Description	Specifies the duration of each statistical profile. The duration cannot be more than 24 hours, and any specified duration must be a factor of 24. For example, factors of 1/4, 1/2, 1, 2, 3, 4, 6, 8, 12, and 24 are acceptable durations.
Example	`PROFILEDURATION 24:00:00`

PURGETIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*0*
Description	The amount of time Moab will keep a job or node record for an object no longer reported by the resource manager. Useful when using a resource manager which 'drops' information about a node or job due to internal failures. Note: This parameter is superseded by **JOBPURGETIME** on page 860.
Example	`PURGETIME 00:05:00` *Moab will maintain a job or node record for 5 minutes after the last update regarding that object received from the resource manager.*

PUSHCACHETOWEBSERVICE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether or not you want to send cache objects (nodes, jobs, services, etc.) to Moab Web Services.
Example	`PUSHCACHETOWEBSERVICE TRUE`

PUSHEVENTSTOWEBSERVICE

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether or not you want to send event logs to web services for storage. For more information, see Event logging with web services. In conjunction with this parameter, you will also need to configure the following parameters to set up event logging to Moab Web Services: • EVENTLOGWSURL • EVENTLOGWSUSER • EVENTLOGWSPASSWORD
Example	`PUSHEVENTSTOWEBSERVICE TRUE`

QOSCFG[<QOSID>]

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: General Credential Flags, **PRIORITY**, ENABLEPROFILING, **FSTARGET**, JOBPRIOACCRUALPOLICY, JOBPRIOEXCEPTIONS, MEMBERULIST, **QTWEIGHT**, **QTTARGET**, **XFWEIGHT**, **XFTARGET**, PREEMPTMINTIME, PREEMPTMAXTIME, **PREEMPTQTTHRESHOLD**, **PREEMPTXFTHRESHOLD**, PREEMPTEES, **RSVQTTHRESHOLD**, **RSVXFTHRESHOLD**, **ACLBLTHRESHOLD**, **ACLQTTHRESHOLD**, **ACLXFTHRESHOLD**, **PLIST**, QFLAGS, or a usage limit.
Default	---
Description	Specifies QoS specific attributes. See the flag overview for a description of legal flag values. See the QoS Overview section for further details.
Example	`QOSCFG[commercial] PRIORITY=1000 MAXJOB=4 MAXPROC=80` *Moab will increase the priority of jobs using QoS commercial, and will allow up to 4 simultaneous QoS commercial jobs with up to 80 total allocated processors.*

QOSDEFAULTORDER

Format	Comma-delimited list of QoS names.
Default	---
Description	Sets a global QoS default order for all QoSes which overrides any specific default QoS. If the order is defined as `b,a,c` and a user has access to `c,a` and submits a job without requesting a specific QoS, the job is assigned `a` as the default QoS.
Example	`QOSDEFAULTORDER  b,a,c` *If the job does not have a QoS specified, it is assigned a QoS from the QOSDEFAULTORDER list (if the user has access to one of them).*

QOSISOPTIONAL

Format	*<BOOLEAN>*
Default	*FALSE*
Description	An entity's default QoS will be the first QoS specified in the QLIST parameter. When this parameter is set to *TRUE* the default QoS for the associated credential (user, account, class, etc.) will not be automatically set to the first QoS specified in the QLIST.

Example	`QOSISOPTIONAL TRUE`
	`USERCFG[bob]  QLIST=high,low`
	Moab will set the QoSList for user bob to high and low but will not set the QDEF. Should bob decide to submit to a particular QoS he will have to do so manually.

QOSREJECTPOLICY

Format	One or more of *CANCEL*, *HOLD*, *IGNORE*, or *MAIL*
Default	*HOLD* (*IGNORE* for SLURM users)
Description	Specifies the action to take when Moab determines that a job cannot access a requested QoS. *CANCEL* issues a call to the resource manager to cancel the job. *HOLD* places a *batch* hold on the job preventing the job from being further evaluated until released by an administrator. (**Note**: Administrators can dynamically alter job attributes and possibly *fix* the job with mjobctl -m.) With *IGNORE*, Moab will ignore the QoS request and schedule the job using the default QoS for that job. MAIL will send email to both the admin and the user when QoS request violations are detected. Most combinations of attributes may be specified; however, if both *MAIL* and *IGNORE* are specified, Moab will not implement *MAIL*. Similarly, while *CANCEL* and *HOLD* are mutually exclusive, *CANCEL* will supersede *HOLD* if both are specified. (see JOBREJECTPOLICY).
Example	`QOSREJECTPOLICY  MAIL,CANCEL`

QOSWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the weight to be applied to the QoS priority of each job (see Credential (CRED) Factor).
Example	`QOSWEIGHT 10`

QUEUETIMECAP

Format	*<DOUBLE>*
Default	*0* (NO CAP)
Description	Specifies the maximum allowed absolute pre-weighted queuetime priority factor.

Example	`QUEUETIMECAP    10000`
	`QUEUETIMEWEIGHT 10`
	*A job that has been queued for 40 minutes will have its queuetime priority factor calculated as 'Priority = QUEUETIMEWEIGHT * MIN(10000,40)'.*

QUEUETIMEWEIGHT	
Format	*<INTEGER>*
Default	*1*
Description	Specifies multiplier applied to a job's queue time (in minutes) to determine the job's queuetime priority factor.
Example	`QUEUETIMEWEIGHT 20` *A job that has been queued for 4:20:00 will have a queuetime priority factor of 20 * 260.*

REALTIMEDBOBJECTS	
Format	Comma-delimited list of one or more of the following: *JOB*, *NODE*, *RSV* (reservation), *TRIG* (trigger), *VC* (virtual container). You can also specify *ALL* or *NONE*.
Default	*ALL*
Description	Specifies which objects Moab will store in the unixodbc database.
Example	`REALTIMEDBOBJECTS JOB,RSV,TRIG` *Moab stores jobs, reservations, and triggers in the uxodbc database. It will no longer record real time information about nodes and VCs.*

RECORDEVENTLIST

Format	One or more comma (',') or plus ('+') separated events of GEVENT, *ALLSCHEDCOMMAND*, *JOBCANCEL*, *JOBCHECKPOINT*, *JOBEND*, *JOBFAILURE*, *JOBMIGRATE*, *JOBMODIFY*, *JOBPREEMPT*, *JOBREJECT*, *JOBRESUME*, *JOBSTART*, *JOBSUBMIT*, *NODEDOWN*, *NODEFAILURE*, *NODEUP*, *QOSVIOLATION*, *RMDOWN*, *RMPOLLEND*, *RMPOLLSTART*, *RMUP*, *RSVCANCEL*, *RSVCREATE*, *RSVEND*, *RSVMODIFY*, *RSVSTART*, *SCHEDCOMMAND*, *SCHEDCYCLEEND*, *SCHEDCYCLESTART*, *SCHEDPAUSE*, *SCHEDSTART*, *SCHEDSTOP*, *VMCREATE*, *VMDESTROY*, *VMMIGRATE*, *VMPOWEROFF*, *VMPOWERON*, or *ALL*
Default	*JOBSTART*, *JOBCANCEL*, *JOBEND*, *JOBFAILURE*, *SCHEDPAUSE*, *SCHEDSTART*, *SCHEDSTOP*, *TRIGEND*, *TRIGFAILURE*, *TRIGSTART*
Description	Specifies which events should be recorded in the appropriate event file found in Moab's `stats/` directory. These events are recorded for both local and remotely staged jobs. (See Event Log Overview) **Note**: If a plus character is included in the list, the specified events will be added to the default list; otherwise, the specified list will replace the default list.
Example	`RECORDEVENTLIST JOBSTART,JOBCANCEL,JOBEND` *When a local and/or remote job starts, is canceled, or ends, the respective event will be recorded.*

REJECTDOSSCRIPTS

Format	*<BOOLEAN>*
Default	*TRUE*
Description	Moab rejects DOS-formatted scripts submitted with the msub command. This is useful if you use SLURM as your resource manager, since it does not handle DOS scripts well. For REJECTDOSSCRIPTS to work correctly, **FILTERCMDFILE** on page 832 must be *FALSE*. Otherwise, Moab modifies the script instead of rejecting it, leading to job errors.
Example	`REJECTDOSSCRIPTS FALSE` *Moab does not reject DOS-formatted scripts submitted with msub.*

REJECTINFEASIBLEJOBS

Format	*<BOOLEAN>*
Default	*FALSE*
Description	If zero feasible nodes are found for a job among the currently available nodes on the cluster, the scheduler rejects the job. See JOBREJECTPOLICY for more information.

Example	`REJECTINFEASIBLEJOBS TRUE`
	`JOBREJECTPOLICY MAIL,CANCEL`
	Any job with zero feasible nodes for execution will be rejected.

REJECTNEGPRIOJOBS

Format	*<BOOLEAN>*
Default	*TRUE*
Description	If enabled, the scheduler will refuse to start any job with a negative priority. See Job Priority Overview and ENABLENEGJOBPRIORITY for more information.

Example	`ENABLENEGJOBPRIORITY TRUE`
	`REJECTNEGPRIOJOBS TRUE`
	Any job with a priority less than zero will be rejected.

REMAPCLASS

Format	<ClassID>
Default	---
Description	Specifies which class/queue will be remapped based on the processors, nodes, and node features requested and the resource limits of each class. See Remap Class Overview for more information. In order to use REMAPCLASS, you must specify a **DEFAULTCLASS**.

Example	`RMCFG[internal]  DEFAULTCLASS=batch`
	`REMAPCLASS       batch`
	`CLASSCFG[small]  MAX.PROC=2`
	`CLASSCFG[medium] MAX.PROC=16`
	`CLASSCFG[large]  MAX.PROC=1024`
	Class batch will be remapped based on the number of processors requested.

REMAPCLASSLIST

Format	Comma delimited list of class names
Default	---
Description	Specifies the order in which classes will be searched when attempting to remap a class. Only classes included in the list will be searched and Moab will select the first class with matches. **Note**: If no REMAPCLASSLIST is specified, Moab will search all classes and will search them in the order they are discovered. See Remap Class Overview for more information.

Example	`RMCFG[internal] DEFAULTCLASS=batch`
	`REMAPCLASS      batch`
	`REMAPCLASSLIST  short,medium,long`
	Class batch will be re-mapped to one of the listed classes.

REMOTEFAILTRANSIENT

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Only applicable to Moab configurations with multiple resource managers able to run jobs. When Moab attempts to migrate a job to one of these resource managers, a remote failure may occur. REMOTEFAILTRANSIENT controls how Moab reacts to remote errors. By default, Moab considers such an error permanent and does not try to migrate the same job to that resource manager again. If REMOTEFAILTRANSIENT is set to *TRUE*, then Moab considers such an error as transient and will not exclude the erring resource manager in future migration attempts.
Example	`REMOTEFAILTRANSIENT    TRUE`

REMOVETRIGOUTPUTFILES

Format	*<BOOLEAN>*
Default	*FALSE*
Description	When Moab launches external trigger actions, the standard output and error of those trigger actions are redirected to files located in Moab's spool directory. By default, these files are cleaned every 24 hours. (Files older than 24 hours are removed.) If, however, you wish to have Moab immediately remove the spool files after they are no longer needed, set RemoveTrigOutputFiles to *TRUE*.
Example	`REMOVETRIGOUTPUTFILES  TRUE`

RESCAP

Format	*<DOUBLE>*
Default	*0* (NO CAP)
Description	Specifies the maximum allowed absolute pre-weighted job resource priority factor.
Example	`RESCAP 1000` *The total resource priority factor component of a job will be bound by +/- 1000*

RESERVATIONDEPTH[X]

Format	*<INTEGER>*
Default	*1*
Description	Specifies the number of priority reservations which are allowed in the associated reservation bucket. **Note**: The array index, *X*, is the bucket label and can be any string up to 64 characters. This label should be synchronized with the RESERVATIONQOSLIST parameter. See Reservation Policies.

Example	`RESERVATIONDEPTH[bigmem]  4`
	`RESERVATIONQOSLIST[bigmem] special,fast,joshua`
	Jobs with QoSes of special, fast, or joshua can have a cumulative total of up to 4 priority reservations.

RESERVATIONPOLICY

Format	One of the following: *CURRENTHIGHEST*, *HIGHEST*, *NEVER*
Default	*CURRENTHIGHEST*
Description	Specifies how Moab reservations will be handled. (See also RESERVATIONDEPTH) See Reservation Policies.

Example	`RESERVATIONPOLICY  CURRENTHIGHEST`
	`RESERVATIONDEPTH[DEFAULT]  2`
	Moab will maintain reservations for only the 2 currently highest priority jobs.

RESERVATIONQOSLIST[X]	
Format	One or more QoS values or *[ALL]*
Default	*[ALL]*
Description	Specifies which QoS credentials have access to the associated reservation bucket. **Note**: The array index, *X*, is the bucket label and can be any string up to 64 characters. This label should be synchronized with the RESERVATIONDEPTH parameter. See Reservation Policies.

Example	`RESERVATIONDEPTH[big]  4`
	`RESERVATIONQOSLIST[big] hi,low,med`
	Jobs with QoSes of hi, low, or med can have a cumulative total of up to 4 priority reservations.

RESERVATIONRETRYTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*60 seconds*
Description	Period of time Moab will continue to attempt to allocate resources to start a job after the time resources should be made available. This parameter takes into account resource manager node state race conditions, nodes with residual high load, network glitches, etc. (For related information, see Reservation Policies, DEFERSTARTCOUNT, DEFERTIME, NODEFAILURERESERVETIME, JOBRETRYTIME, and GUARANTEEDPREEMPTION.)
Example	`RESERVATIONRETRYTIME  00:05:00` *Moab will try for up to 5 minutes to maintain immediate reservations if the reservations are blocked due to node state, network, or batch system based race conditions.*

RESOURCELIMITMULTIPLIER[<PARID>]

Format	*<RESOURCE>:<MULTIPLIER>[,...]* Where *<RESOURCE>* is one of the following: **NODE**, **PROC**, **JOBPROC**, **MEM**, **JOBMEM**, **SWAP**, **DISK**, or **WALLTIME**
Default	*1.0*
Description	If set to less than one, then the hard limit will be the specified limit and the soft limit will be the specified limit multiplied by the multiplier. If set to a value greater than one, then the specified limit will be the soft limit and the hard limit will be the specified limit multiplied by the multiplier. See Usage-based Limits.
Example	`RESOURCELIMITMULTIPLER PROC:1.1,MEM:2.0` *Sets hard limit for **PROC** at 1.1 times the **PROC** soft limit, and the hard limit of **MEM** to 2.0 times the MEM soft limit.*

RESOURCELIMITPOLICY

Format	*<RESOURCE>:[<SPOLICY>,]<HPOLICY> :[<SACTION>,]<HACTION> [:[<SVIOLATIONTIME>,]<HVIOLATIONTIME>]...* Where **RESOURCE** is one of *CPUTIME*, *DISK*, *JOBMEM*, *JOBPROC*, *MEM*, *MINJOBPROC*, *NETWORK*, *PROC*, *SWAP*, or *WALLTIME* where ***POLICY** is one of *ALWAYS*, *EXTENDEDVIOLATION*, or *BLOCKEDWORKLOADONLY* and where ***ACTION** is one of *CANCEL*, *CHECKPOINT*, *NOTIFY*, *REQUEUE*, *SIGNAL*, or *SUSPEND*.
Default	No limit enforcement.
Description	Specifies how the scheduler should handle jobs which utilize more resources than they request. See Usage-based Limits.
Example	`RESOURCELIMITPOLICY MEM:ALWAYS,BLOCKEDWORKLOADONLY:REQUEUE,CANCEL` *Moab will cancel all jobs which exceed their requested memory limits.*

RESTARTINTERVAL

Format	*[[[DD:]HH:]MM:]SS*
Default	---
Description	Causes Moab daemon to recycle/restart when the given interval of time has transpired.
Example	`RESTARTINTERVAL 20:00:00` *Moab daemon will automatically restart every 20 hours.*

RESOURCEQUERYDEPTH

Format	*<INTEGER>*
Default	*3*
Description	Maximum number of options which will be returned in response to an mshow -a resource query.
Example	`RESOURCEQUERYDEPTH 1` *The mshow -a command will return at most 1 valid collection of resources.*

RESWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	All resource priority components are multiplied by this value before being added to the total job priority. See Job Prioritization.

Example	`RESWEIGHT 5`
	`MEMWEIGHT 10`
	`PROCWEIGHT 100`
	`SWAPWEIGHT 0`
	`RESCAP 2000`
	*The job priority resource factor will be calculated as MIN(2000,5 * (10 * JobMemory + 100 * JobProc)).*

RMCFG	
Format	One or more key-value pairs as described in the Resource Manager Configuration Overview
Default	---
Description	Specifies the interface and policy configuration for the scheduler-resource manager interface. Described in detail in the Resource Manager Configuration Overview.
Example	`RMCFG[TORQUE3] TYPE=PBS` *The PBS server will be used for resource management.*

RMMSGIGNORE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether or not Moab should adjust node state based on generic resource manager failure messages. See RM Health Check for more info. For green or ONDEMAND computing, RMMSGIGNORE must be set to *TRUE* to prevent Moab from powering off a down node.
Example	`RMMSGIGNORE TRUE` *Moab will load and report resource manager failure messages but will not adjust node state as a result of them.*

RMPOLLINTERVAL

Format	*[<MINPOLLTIME>,]<MAXPOLLTIME>* where poll time is specified as *[[[DD:]HH:]MM:]SS*
Default	*0,30*
Description	Specifies interval between RM polls. The poll interval will be no less than MINPOLLTIME and no more than MAXPOLLTIME. If you specify a single value, Moab interprets the value as the MAXPOLLTIME with a MINPOLLTIME of 0. If you use TORQUE as your resource manager, prevent communication errors by giving tcp_timeout at least twice the value of the Moab RMPOLLINTERVAL.
Example	`RMPOLLINTERVAL 30,45` *Moab will refresh its resource manager information between a minimum of 30 seconds and a maximum of 45 seconds. **Note**: This parameter specifies the default global poll interval for all resource managers.*

RMRETRYTIMECAP

Format	*[[[DD:]HH:]MM:]SS*
Default	*1:00:00*
Description	Moab attempts to contact RMs that are in state 'corrupt' (not down). If the attempt is unsuccessful, Moab tries again later. If the second attempt is unsuccessful, Moab increases the gap (the gap grows exponentially) between communication attempts. RMRETRYTIMECAP puts a cap on the length between connection attempts.
Example	`RMRETRYTIMECAP 24:00:00` *Moab stops increasing the gap between connection attempts once the retry gap reaches 24 hours.*

RSVLIMITPOLICY	
Format	*HARD* or *SOFT*
Default	---
Description	Specifies what limits should be enforced when creating reservations.
Example	`RSVLIMITPOLICY  HARD` *Moab will limit reservation creation based on the HARD limits configured.*

RSVNODEALLOCATIONPOLICY	
Format	One of the following: FIRSTAVAILABLE, LASTAVAILABLE, MINRESOURCE, CPULOAD, LOCAL, CONTIGUOUS, MAXBALANCE, or PRIORITY
Default	*LASTAVAILABLE*
Description	Specifies how Moab should allocate available resources to reservations.
Example	`RSVNODEALLOCATIONPOLICY MINRESOURCE` *Moab will apply the node allocation policy MINRESOURCE to all reservations by default.*

RSVNODEALLOCATIONPRIORITYF	
Format	User specified algorithm
Default	---
Description	When RSVNODEALLOCATIONPOLICY is set to *PRIORITY*, this parameter allows you to specify your own priority algorithm. The priority functions available are the same as the node priority functions.
Example	`RSVNODEALLOCATIONPOLICY PRIORITY` `RSVNODEALLOCATIONPRIORITYF 'SPEED + .01 * AMEM - 10 * JOBCOUNT'`

RSVPROFILE[X]	
Format	One or more of the following: *Allowed*: TRIGGERACL (ACCOUNTLIST, CLASSLIST, GROUPLIST, MAXTIME, QOSLIST, USERLIST) HostExp (HOSTLIST) Features (NODEFEATURES) FLAGS TASKCOUNT RSVACCESSLIST **Note**: Lists of more than one ACL value cannot be whitespace delimited. Such lists must be delimited with the comma, pipe, or colon characters. *Not allowed*: ACCESS CHARGEACCOUNT DAYS DEPTH ENDTIME OWNER PARTITION PERIOD **PRIORITY** RESOURCES STARTTIME TPN
Default	---
Description	Specifies attributes of a reservation profile using syntax similar to that for specifying a standing reservation. See Using Reservation Profiles for details.

Example	`RSVPROFILE[fast] USERLIST=john,steve`
	`RSVPROFILE[fast] QOSLIST=high,low`
	`RSVPROFILE[fast]`
	`TRIGGER=ETYPE=start,OFFSET=5:00,ATYPE=exec,ACTION="/opt/moab/rp.pl"`
	Moab will create a reservation profile including trigger and ACL information.

RSVSEARCHALGO

Format	*LONG* or *WIDE*
Default	*NONE*
Description	When Moab is determining when and where a job can run, it either searches for the most resources (WIDE) or the longest range of resources (LONG). In almost all cases, searching for the longest range is ideal and returns the soonest starttime. In some rare cases, however, a particular job may need to search for the most resources. In those cases sites can configure this parameter to prevent the starvation of large jobs that fail to hold onto their reservation starttimes. See the WIDERSVSEARCHALGO job flag. If this parameter is not set, it will be displayed in mschedctl -l as *NONE* but the algorithm that is used will be *LONG*.
Example	`RSVSEARCHALGO WIDE`

SCHEDCFG

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: FBSERVER, FLAGS, **MAXRECORDEDCJOBID**, **MINJOBID**, HTTPSERVERPORT, MODE, RECOVERYACTION, SERVER, or TRIGGER
Default	---
Description	Specifies scheduler policy and interface configuration. The **SERVER** attribute can also be set using the environment variable **$MOABSERVER**. Using this variable allows you to quickly change to Moab server that client commands will connect to. `> export MOABSERVER=cluster2:12221`
Example	`SCHEDCFG[zylem3] SERVER=geronimo.scc.com:3422 MODE=NORMAL` *Moab will execute in NORMAL mode on the host geronimo.scc.com.*

SERVERHOST

Format	*<HOSTNAME>*
Default	---
Description	**Deprecated**. Host name 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*
Description	**Deprecated**. Specifies how Moab interacts with the outside world. See SCHEDCFG for replacement parameter.
Example	`SERVERMODE SIMULATION`

SERVERNAME

Format	*<STRING>*
Default	*<SERVERHOST>*
Description	Specifies the name the scheduler will use to refer to itself in communication with peer daemons. See SCHEDCFG for replacement parameter.
Example	`SERVERNAME moabA`

SERVERPORT

Format	*<INTEGER>* (range: 1-64000)
Default	*40559*
Description	Port on which moab will open its user interface socket. See SCHEDCFG for replacement parameter.
Example	`SERVERPORT 30003` *Moab will listen for client socket connections on port 30003.*

SERVERSUBMITFILTER

Format	*<PORT>*
Default	---
Description	Specifies the location of a global job submit filter script. When you configure a global job submit filter, Moab executes it on the head node and uses it to filter every job submission it receives. See **Global job submit filter** on page 219 for more information about job submit filters.
Example	`SERVERSUBMITFILTER /opt/moab/scripts/globalfilter.pl` *Moab uses /opt/moab/scripts/globalfilter.pl to filter every job submitted to Moab.*

SERVICEWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the service component weight associated with the service factors. See Service (SERV) Factor for more information.
Example	`SERVICEWEIGHT 2`

SHOWMIGRATEDJOBSASIDLE	
Format	*<BOOLEAN>*
Default	*FALSE*
Description	By default, migrated jobs will show as blocked. This is to prevent jobs from counting against the idle policies of multiple clusters rather than just the cluster to which the job was migrated.
Example	`SHOWMIGRATEDJOBSASIDLE TRUE` *When set to TRUE, migrated jobs will show as idle and will count against the idle policies of the cluster showing the job as migrated.*

SIMAUTOSHUTDOWN	
Format	*<BOOLEAN>*
Default	*TRUE*
Description	If *TRUE*, the scheduler will end simulations when the active queue and idle queue become empty.
Example	`SIMAUTOSHUTDOWN TRUE` *The simulation will end as soon as there are no jobs running and no idle jobs which could run.*

SIMINITIALQUEUEDEPTH

Format	*<INTEGER>*
Default	*16*
Description	Specifies how many jobs the simulator will initially place in the idle job queue (see **Simulations** on page 618).

Example	`SCHEDCFG[sim1] MODE=SIMULATION`
	`SIMINITIALQUEUEDEPTH 64`
	`SIMJOBSUBMISSIONPOLICY CONSTANTJOBDEPTH`
	Moab will initially place 64 idle jobs in the queue and, because of the specified queue policy, will attempt to maintain this many jobs in the idle queue throughout the duration of the simulation.

SIMJOBSUBMISSIONPOLICY

Format	One of the following: *NORMAL*, *CONSTANTJOBDEPTH*, *CONSTANTPSDEPTH*, or REPLAY
Default	*CONSTANTJOBDEPTH*
Description	Specifies how the simulator will submit new jobs into the idle queue. *NORMAL* mode causes jobs to be submitted at the time recorded in the workload trace file, *CONSTANTJOBDEPTH* and *CONSTANTPSDEPTH* attempt to maintain an idle queue of SIMINITIALQUEUEDEPTH jobs and procseconds 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 **Simulations** on page 618).
Example	`SIMJOBSUBMISSIONPOLICY NORMAL` *Moab will submit jobs with the relative time distribution specified in the workload trace file.*

SIMPURGEBLOCKEDJOBS

Format	*<BOOLEAN>*
Default	*TRUE*
Description	Specifies whether Moab should remove jobs which can never execute (see Simulation Overview).
Example	`SIMPURGEBLOCKEDJOBS  FALSE`

SIMRMRANDOMDELAY

Format	*<INTEGER>*
Default	*0*
Description	Specifies the random delay added to the RM command base delay accumulated when making any resource manager call in simulation mode.
Example	`SIMRMRANDOMDELAY  5` *Moab will add a random delay of between 0 and 5 seconds to the simulated time delay of all RM calls.*

SIMSTARTTIME

Format	*[HH[:MM[:SS]]][_MO[/DD[/YY]]]*
Default	---
Description	Specifies the time when the simulation starts.
Example	`SIMSTARTTIME 00:00:00_01/01/00` *Moab will set its clock to January 1, 2000 at 12:00:00 in the morning before starting the simulation*

SIMSTOPTIME

Format	*[HH[:MM[:SS]]][_MO[/DD[/YY]]]*
Default	---
Description	Specifies the time when the simulation should pause.
Example	`SIMSTOPTIME 00:00:00_01/01/04` *Moab will stop scheduling when its internal simulation time reaches January 1, 2004.*

SIMWORKLOADTRACEFILE

Format	*<STRING>*
Default	*Traces/workload.trace*
Description	Specifies the file from which moab will obtain job information when running in simulation mode. Moab will attempt to locate the file relative to *<MOABHOMEDIR>* unless specified as an absolute path. See Simulation Overview and Workload Accounting Records.
Example	`SIMWORKLOADTRACEFILE traces/jobs.2` *Moab will obtain job traces when running in simulation mode from the `<MOABHOMEDIR>/traces/jobs.2` file.*

SPOOLDIR

Format	*<STRING>*
Default	---
Description	Specifies the directory for temporary spool files created by Moab while submitting a job to the RM.
Example	`SPOOLDIR /tmp/moab/spool`

SPOOLDIRKEEPTIME	
Format	*<INTEGER>* (seconds) or [[[DD:]HH:]MM:]SS
Default	---
Description	Specifies the interval to delete spool files and other temporary files that have been left in the spool directory.
Example	`SPOOLDIRKEEPTIME 4:00:00`

SPVIOLATIONWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight to be applied to a job which violates soft usage limit policies (see **Service (SERVICE) Component** on page 292).
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, **JOBATTRLIST**, **MAXTIME**, **NODEFEATURES**, OWNER, **PARTITION**, PERIOD, **PRIORITY**, **QOSLIST**, REQUIREDTPN, RESOURCES, ROLLBACKOFFSET, RSVACCESSLIST, RSVGROUP, STARTTIME, TASKCOUNT, **TIMELIMIT**, **TPN**, **TRIGGER**, or USERLIST **Note:** **HOSTLIST** and **ACL** list values must be comma delimited. For example: **HOSTLIST**=*nodeA*,*nodeB*
Default	---
Description	Specifies attributes of a standing reservation. See Managing Reservations for details.

Example	`SRCFG[fast] STARTTIME=9:00:00 ENDTIME=15:00:00`
	`SRCFG[fast] HOSTLIST=node0[1-4]$`
	`SRCFG[fast] QOSLIST=high,low`
	Moab will create a standing reservation running from 9:00 AM to 3:00 PM on nodes 1 through 4 accessible by jobs with QoS high or low.

STARTCOUNTCAP

Format	*<INTEGER>*
Default	*0*
Description	Specifies the max weighted value allowed from the startcount subfactor when determining a job's priority (see Priority Factors for more information).
Example	`STARTCOUNTWEIGHT 5000` `STARTCOUNTCAP    30000`

STARTCOUNTWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight to be applied to a job's startcount when determining a job's priority (see Priority Factors for more information).
Example	`STARTCOUNTWEIGHT 5000`

STATDIR	
Format	*<STRING>*
Default	*stats*
Description	Specifies the directory in which Moab statistics will be maintained.
Example	`STATDIR /var/adm/moab/stats`

STATPROCMAX

Format	*<INTEGER>*
Default	*1*
Description	Specifies the maximum number of processors requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via mschedctl -m or changeparam while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.

Example	`STATPROCMAX     256`
	`STATPROCSTEPCOUNT 4`
	`STATPROCSTEPSIZE  4`
	Each matrix output will display data in rows for jobs requesting between 4 and 256 processors.
	A **NONE** in services will still allow users to run showq and checkjob on their own jobs.

STATPROCMIN

Format	*<INTEGER>*
Default	*1*
Description	Specifies the minimum number of processors requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.

Example	`STATPROCMIN 4`
	`STATPROCSTEPCOUNT 4`
	`STATPROCSTEPSIZE 4`
	Each matrix output will display data in rows for jobs requesting between 4 and 256 processors.
	A **NONE** in services will still allow users to run showq and checkjob on their own jobs.

STATPROCSTEPCOUNT

Format	*<INTEGER>*
Default	*5*
Description	Specifies the number of rows of processors requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.

Example	`STATPROCMIN 4`
	`STATPROCSTEPCOUNT 4`
	`STATPROCSTEPSIZE 4`
	Each matrix output will display data in rows for jobs requesting between 4 and 256 processors.

STATPROCSTEPSIZE

Format	*<INTEGER>*
Default	*4*
Description	Specifies the processor count multiplier for rows of processors requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.

Example	`STATPROCMIN 4`
	`STATPROCSTEPCOUNT 4`
	`STATPROCSTEPSIZE 4`
	Each matrix output will display data in rows for jobs requesting between 4 and 256 processors.

STATTIMEMAX

Format	*[[DD:]HH:]MM:]SS*
Default	*00:15:00*
Description	Specifies the maximum amount of time requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.

Example	`STATTIMEMAX 02:08:00`
	`STATTIMESTEPCOUNT 4`
	`STATTIMESTEPSIZE 4`
	Each matrix output will display data in columns for jobs requesting between 2 and 128 minutes.

STATTIMEMIN

Format	*[[DD:]HH:]MM:]SS*
Default	*00:15:00*
Description	Specifies the minimum amount of time requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.

Example	`STATTIMEMIN 00:02:00`
	`STATTIMESTEPCOUNT 4`
	`STATTIMESTEPSIZE 4`
	Each matrix output will display data in columns for jobs requesting between 2 and 128 minutes.

STATTIMESTEPCOUNT

Format	*<INTEGER>*
Default	*6*
Description	Specifies the number of columns of time requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.

Example	`STATTIMEMIN 00:02:00`
	`STATTIMESTEPCOUNT 4`
	`STATTIMESTEPSIZE 4`
	Each matrix output will display data in columns for jobs requesting between 2 and 128 minutes.

STATTIMESTEPSIZE

Format	*<INTEGER>*
Default	*4*
Description	Specifies the time multiplier for columns of time requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). It is recommended that you not change any parameters via 'mschedctl -m' or 'changeparam' while Moab is running. Changing any of the parameters invalidates all past data and will start the collection over.

Example	`STATTIMEMIN 00:02:00`
	`STATTIMESTEPCOUNT 4`
	`STATTIMESTEPSIZE 4`
	Each matrix output will display data in columns for jobs requesting between 2 and 128 minutes.

STOPITERATION

Format	*<INTEGER>*
Default	*-1* (don't stop)
Description	Specifies which scheduling iteration Moab will stop and wait for a command to resume scheduling.
Example	`STOPITERATION 10` *Moab should stop after iteration 10 of scheduling and wait for administrator commands.*

STOREJOBSUBMISSION

Format	*<BOOLEAN>*
Default	---
Description	When set to *TRUE*, specifies that Moab will save a job's submit arguments and script to `$MOABHOMEDIR/stats/jobarchive/jobNumber`. If you use TORQUE as your resource manager, you can configure it to store completed job information, and it will store the same information returned by the qstat -f command. For more information, see Job logging in the TORQUE documentation.
Example	`STOREJOBSUBMISSION TRUE`

STRICTPROTOCOLCHECK

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies how Moab reacts to differences in XML protocols when communicating with other Moab peers. If set to *TRUE*, Moab will reject any communication that does not strictly conform to the expected protocol. If set to *FALSE* (the default), Moab will not reject XML that has extra or unknown attributes.
Example	`STRICTPROTOCOLCHECK TRUE` *Moab will reject any XML communication that does not strictly conform to the expected protocol definition.*

SUBMITENVFILELOCATION	
Format	*FILE* or *PIPE*
Default	---
Description	If set to *FILE*, these behaviors are expected: • The environment file is owned by a user with 600 permissions. • Moab writes the environment variables ('\0' delimited) to a random file in Moab's spool directory. • Moab adds the `--export-file=<path_to_file>` on the sbatch command line. • Moab deletes the file after the job completes. If set to *PIPE*, these behaviors are expected: • Moab creates a pipe and passes the read end of the pipe's file descriptor to sbatch. • Moab's parent process writes the environment ('\0' delimited) into the write end of the pipe. Adaptive Computing recommends that you configure this parameter for a more secure environment.
Example	`SUBMITENVFILELOCATION PIPE`

SUBMITFILTER	
Format	*<STRING>*
Default	---
Description	Specifies the directory of a given submit filter script.
Example	`SUBMITFILTER /home/submitfilter/filter.pl`

SUBMITHOSTS	
Format	space delimited list of host names
Default	---
Description	If specified, **SUBMITHOSTS** specifies an explicit list of hosts where jobs can be submitted.
Example	`SUBMITHOSTS hostA hostB`

SUSPENDRESOURCES[<PARID>]

Format	*<RESOURCE>[,...]* Where *<RESOURCE>* is one of the following: *NODE*, *PROC*, *MEM*, *SWAP*, *DISK*
Default	---
Description	List of resources to dedicate while a job is suspended (available in Moab version 4.5.1 and higher).
Example	`SUSPENDRESOURCES[base]  MEM,SWAP,DISK` *While a job is suspended in partition base, the memory, swap and disk for that job will remain dedicated to the job.*

SYSCFG

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: **PRIORITY**, **FSTARGET**, **QLIST**, **QDEF**, **PLIST**, **FLAGS**, or a fairness policy specification.
Default	---
Description	Specifies system-wide default attributes. See the Attribute/Flag Overview for more information.
Example	`SYSCFG PLIST=Partition1 QDEF=highprio` *By default, all jobs will have access to partition Partition1 and will use the QoS highprio.*

SWAPWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight assigned to the virtual memory request of a job.
Example	`SWAPWEIGHT 10`

SYSTEMMAXPROCPERJOB	
Format	*<INTEGER>*
Default	*-1* (NO LIMIT)
Description	Specifies the maximum number of processors that can be requested by any single job.
Example	`SYSTEMMAXPROCPERJOB 256` *Moab will reject jobs requesting more than 256 processors.*

SYSTEMMAXPROCSECONDPERJOB	
Format	*<INTEGER>*
Default	*-1* (NO LIMIT)
Description	Specifies the maximum number of proc-seconds that can be requested by any single job.
Example	`SYSTEMMAXJOBPROCSECOND 86400` *Moab will reject jobs requesting more than 86400 procs seconds. i.e., 64 processors * 30 minutes will be rejected, while a 2 processor * 12 hour job will be allowed to run.*

SYSTEMMAXJOBWALLTIME	
Format	*[[[DD:]HH:]MM:]SS*
Default	*-1* (NO LIMIT)
Description	Specifies the maximum amount of wallclock time that can be requested by any single job.
Example	`SYSTEMMAXJOBWALLTIME 1:00:00:00` *Moab will reject jobs requesting more than 1 day of walltime.*

TARGETQUEUETIMEWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to the time remaining until the queuetime is reached.
Example	`TARGETQUEUETIMEWEIGHT 10`

TARGETWEIGHT	
Format	*<INTEGER>*
Default	*1*
Description	Specifies the weight to be applied to a job's queuetime and expansion factor target components (see Job Prioritization).
Example	`TARGETWEIGHT 1000`

TARGETXFACTORWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to the distance to the target expansion factor.
Example	`TARGETXFACTORWEIGHT  10`

TASKDISTRIBUTIONPOLICY	
Format	One of *DEFAULT*, *PACK*, *RR* (round-robin)
Default	---
Description	Specifies how job tasks should be mapped to allocated resources. *DEFAULT* allows the resource manager to determine how the tasks are placed on the nodes. When *PACK* is used, a node is filled up with tasks before the next node is used. When *RR* is used, tasks are cycled through nodes, one task at a time, until there are no more tasks. See Task Distribution Overview for more information.
Example	`TASKDISTRIBUTIONPOLICY DEFAULT` *Moab should use standard task distribution algorithms.*

THREADPOOLSIZE	
Format	*<INTEGER>*
Default	*2* (MAX: 25)
Description	Specifies how many threads to have available for threaded operations.
Example	`THREADPOOLSIZE 10`

TOOLSDIR	
Format	*<STRING>*
Default	*tools*
Description	Specifies the directory in which Moab tools will be maintained (commonly used in conjunction with Native Resource Managers, and Triggers).
Example	`TOOLSDIR /var/adm/moab/tools`

TRAPFUNCTION	
Format	*<STRING>*
Default	---
Description	Specifies the functions to be trapped.
Example	`TRAPFUNCTION UpdateNodeUtilization|GetNodeSResTime`

TRAPJOB	
Format	*<STRING>*
Default	---
Description	Specifies the jobs to be trapped.
Example	`TRAPJOB pros23.0023.0`

TRAPNODE	
Format	*<STRING>*
Default	---
Description	Specifies the nodes to be trapped.
Example	`TRAPNODE node001|node004|node005`

TRAPRES	
Format	*<STRING>*
Default	---
Description	Specifies the reservations to be trapped.
Example	`TRAPRES interactive.0.1`

TRIGCHECKTIME	
Format	*<INTEGER>* (milliseconds)
Default	*2000*
Description	Each scheduling iteration, Moab will have a period of time where it handles commands and other UI requests. This time period is controlled by RMPOLLINTERVAL. During this time period, known as the UI phase, Moab will periodically evaluate triggers. Usually this only takes a fraction of a second, but if the number of triggers is large it could take up substantially more time (up to several seconds). While Moab is evaluating triggers, it doesn't respond to UI commands. This makes Moab feel sluggish and unresponsive. To remedy this, use the parameter **TRIGCHECKTIME**. This parameter tells Moab to only spend up to X milliseconds processing triggers during the UI phase. After *X* milliseconds has gone by, Moab will pause the evaluating of triggers, handle any pending UI events, and then restart the trigger evaluations where it last left off.
Example	`TRIGCHECKTIME 4000`

TRIGEVALLIMIT

Format	*<INTEGER>*
Default	*1*
Description	Each scheduling iteration, Moab will have a period of time where it handles commands and other UI requests. This time period is controlled by RMPOLLINTERVAL. During this time period, known as the UI phase, Moab will periodically evaluate triggers. The number of times Moab evaluates all triggers in the system is controlled by the TRIGEVALLIMIT parameter. By default, this is set to *1*. This means that Moab will evaluate all triggers at most once during the UI phase. Moab will not leave the UI phase and start other scheduling tasks until ALL triggers are evaluated at least one time. If TrigEvalLimit is set to *5*, then Moab will wait until all triggers are evaluated five times.
Example	`TRIGEVALLIMIT   3`

UJOBWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Weight assigned by jobs per user. *-1* will reduce priority by number of active jobs owned by user.
Example	`UJOBWEIGHT 10`

UMASK

Format	*<INTEGER>*
Default	*0022* (octal) (produces 0644 permissions)
Description	Specifies the file permission mask to use when creating new fairshare, stats, and event files. See the `umask` man page for more details.
Example	`UMASK 0127` *Create statistics and event files which are 'read-write' by owner and 'read' by group only.*

UNSUPPORTEDDEPENDENCIES

Format	Comma delimited string
Default	---
Description	Specifies dependencies that are not supported and should not be accepted by job submissions. A maximum of 30 dependencies is supported.

Example	`# moab.cfg`
	`UNSUPPORTEDDEPENDENCIES before,beforeok,beforenotok,on`
	
	`> msub -l depend=before:105 cmd.sh`
	`ERROR: cannot submit job - error in extension string`

UPROCWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Weight assigned by processors per user. *-1* will reduce priority by number of active procs owned by user.
Example	`UPROCWEIGHT 10`

USAGECONSUMEDWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to per job processor second consumption.
Example	`USAGECONSUMEDWEIGHT 10`

USAGEEXECUTIONTIMEWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight assigned to the total job execution time (measured in seconds since job start). See Preemption Overview.
Example	`USAGEEXECUTIONTIMEWEIGHT 10`

USAGEPERCENTWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to total requested resources consumed.
Example	`USAGEPERCENTWEIGHT 5`

USAGEREMAININGWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight assigned to remaining usage.
Example	`USAGEREMAININGWEIGHT 10`

USAGEWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the weight assigned to the percent and total job usage subfactors.
Example	`USAGEWEIGHT 100`

USEANYPARTITIONPRIO

Format	*<BOOLEAN>*
Default	*FALSE*
Description	The FSTREE data from the first feasible FSTREE will be used when determining a job's start priority, rather than having no FSTREE data considered. Do not set USEANYPARTITIONPRIO if you use per-partition scheduling. Doing so causes to schedule jobs to the first partition listed, even if nodes from another partition will be available sooner.
Example	`USEANYPARTITIONPRIO TRUE`

USECPRSVNODELIST

Format	*<BOOLEAN>*
Default	*TRUE*
Description	Specifies whether Moab should use the checkpointed reservation node list when rebuilding reservations on startup. If this is not used then Moab will use the reservation's specified host expression during rebuilding.
Example	`USECPRSVNODELIST FALSE`

USEDATABASE

Format	INTERNAL
Default	-
Description	Specifies whether Moab should store profile statistics, checkpoint information, and event information in an integrated database. See Layout of Scheduler Components with Integrated Database Enabled for more information.
Example	`USEDATABASE INTERNAL`

USEMOABCTIME

Format	*<BOOLEAN>*
Default	*FALSE*
Description	When Moab finds new jobs on the resource manager, it creates a job inside of Moab for each job in the resource manager. By default, when Moab creates a new job, it uses the time the job was submitted to the resource manager to calculate how long the job has been in the queue (Moab processing time - job creation in resource manager), which is then used in determining the job's priority. In a system where more jobs are submitted to a resource manager than Moab can handle in one iteration, there is the possibility of jobs running out of order. For example, two jobs are both submitted at time 5. The first submitted job is processed first at time 6. So the first job's effective queue duration would be 1 (6-5). On the next iteration, the second job is processed at time 8. So the second job's effective queue duration would be 3 (8-5), indicating that it has been in the queue longer than the other job. Since the later job has a higher effective queue duration it will get a higher priority and could be scheduled to run before earlier submitted jobs. Setting USEMOABCTIME to *TRUE* tells Moab to use the creation time of the job in Moab rather than the creation time in the resource manager. This corrects the possible problem of having later submitted jobs having higher priorities and starting before earlier submitted jobs.
Example	`USEMOABCTIME TRUE`

USEMOABJOBID

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether to use the Moab job ID, or the resource manager's job ID.
Example	`USEMOABJOBID TRUE`

USERCFG[<USERID>]

Format	List of zero or more space delimited *<ATTR>=<VALUE>* pairs where *<ATTR>* is one of the following: General Credential Flags, **CDEF**, **DEFAULT.TPN**, **DEFAULT.WCLIMIT**, **EMAILADDRESS**, ENABLEPROFILING, **FSCAP**, **FSTARGET**, JOBFLAGS, **MAX.WCLIMIT**, **QLIST**, **QDEF**, **NOEMAIL**, **OVERRUN**, **PLIST**, PRIORITY, or a usage limit.
Default	---
Description	Specifies user specific attributes. For general user attribute information, See the Credential Overview. For a description of legal flag values, see flag overview.

Example	`USERCFG[john] MAXJOB=50 QDEF=highprio`
	`USERCFG[john] EMAILADDRESS=john@company.com`
	Up to 50 jobs submitted under the user ID john will be allowed to execute simultaneously and will be assigned the QoS highprio.

USERPRIOCAP

Format	*<INTEGER>*
Default	---
Description	Specifies the priority cap to be applied to the user specified job priority factor. Under Moab, only negative user priorities may be specified. See Credential (Service) Factor.
Example	`USERPRIOWEIGHT 10` `USERPRIOCAP    -10000`

USERPRIOWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight to be applied to the user specified job priority. Under Moab, only negative user priorities may be specified. If this weight is set, users may reduce the priority of some of their jobs to allow other jobs to run earlier. See Credential (Service) Factor and User Selectable Prioritization.
Example	`USERPRIOWEIGHT 10`

USERWEIGHT

Format	*<INTEGER>*
Default	*1*
Description	Specifies the weight to be applied to the user priority of each job. See Credential (CRED) Factor.
Example	`USERWEIGHT 10`

USESYSLOG

Format	*<BOOLEAN>[<FACILITY>]*
Default	*FALSE:daemon*
Description	Specifies whether or not the scheduler will report key events to the system syslog facility. If the *<FACILITY>* is specified, Moab will report events to this syslog facility. See Logging Facilities for more information.
Example	`USESYSLOG TRUE:local3` *Moab will report key events, commands, and failures to syslog using the local3 facility.*

USESYSTEMQUEUETIME

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Specifies whether or not job prioritization should be based on the time the job has been eligible to run, i.e., idle and meets all fairness policies (*TRUE*) or the time the job has been idle (*FALSE*). See Priority Factors for more info. **Note**: This parameter has been superseded by the JOBPRIOACCRUALPOLICY parameter.
Example	`USESYSTEMQUEUETIME FALSE` *The queuetime and expansion factor components of a job's priority will be calculated based on the length of time the job has been in the idle state.*

USEUSERHASH

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Enables searching of the user buffer using the user hash key instead of doing sequential searches of the user buffer.
Example	`USEUSERHASH TRUE`

VMCALCULATELOADBYVMSUM

Format	*<BOOLEAN>*
Default	*FALSE*
Description	When false, vmmigrate using overcommits uses the CPU load from the node to determine if VM's need to be migrated off the hypervisor. When true, overcommit vmmigrates calculates the total node load using the total sum reported by each VM on the hypervisor.
Example	`VMCALCULATELOADBYVMSUM  TRUE`

VMCPURGETIME

Format	*[[[DD:]HH:]MM:]SS*
Default	*5:00*
Description	When a VM completes, Moab stores it in a completed VM table for the specified amount of time. This prevents it from starting again if an RM reports it late. It also prevents a user from creating a VM with the same ID for a certain amount of time.
Example	`VMCPURGETIME 10:00` *Moab holds completed VMs for 10 minutes to prevent a late RM from reporting and restarting it.*

VMMIGRATETOZEROLOADNODES

Format	*<BOOLEAN>*
Default	*FALSE*
Description	Allows VM migrations to occur to and from hypervisors that do not report a CPULoad or memory load.
Example	`VMMIGRATETOZEROLOADNODES TRUE`

VMMIGRATETHROTTLE

Format	*<INTEGER>*
Default	---
Description	Sets the maximum allowable 'VM migrate' jobs at any given time.
Example	`VMMIGRATETHROTTLE 20` *Only 20 VM migrate jobs are allowed in the system at any given time.*

VMMIGRATIONPOLICY

Format	<*STRING*>; values include *CONSOLIDATION* and *OVERCOMMIT*
Default	*NONE*
Description	Choose only one of these values: • *CONSOLIDATION*- If the *CONSOLIDATION* flag is set, Moab consolidates VMs to allow nodes to go idle. This flag also ensures that no hypervisors are overloaded. • *OVERCOMMIT*- If the *OVERCOMMIT* flag is set, VMs to be migrated will be selected from overloaded hypervisors to bring them below the selected thresholds. This flag must be set for the VMOCTHRESHOLD parameter to function.
Example	VMMIGRATIONPOLICY OVERCOMMIT *The OVERCOMMIT VM migration policy is set.* VMMIGRATIONPOLICY CONSOLIDATION *The CONSOLIDATION VM migration policy is set.*

VMMINOPDELAY

Format	*[HH[:MM[:SS]*
Default	---
Description	The minimum time between automatic VM node operations, such as creating, modifying, and destroying VMs. May prevent thrashing.
Example	`VMMINOPDELAY 30`

VMOCTHRESHOLD

Format	*MEM:<0-1>,PROCS:<0-1>,DISK:<0-1>,SWAP:<0-1>,GMETRIC:<metric>:value*
Default	---
Description	Percentage threshold at which Moab begins to migrate virtual machines to other nodes. VMMIGRATIONPOLICY must be set to OVERCOMMIT for this to occur.

Example	`NODECFG[DEFAULT]  VMOCTHRESHOLD=PROC:.7,MEM:.9,GMETRIC:mem_io:6000    # This is the default global policy`
	`NODECFG[node42]  VMOCTHRESHOLD=PROC:.2,MEM:.1,GMETRIC:mem_io:12000  # This is a node-specific policy for node42`
	When a node surpasses .7 (70%) load of CPU or .9 (90%) of memory, Moab begins to migrate virtual machines to other nodes. When node42surpasses .2 (20%) load of CPU or .1 (10%) of memory, Moab begins to migrate virtual machines to other nodes.

VMPROVISIONSTATUSREADYVALUE

Format	*<INTEGER>*
Default	---
Description	Checks a VM for a special value or values (which Moab gets from the resource manager) and, based on the value, tells Moab that a VM was created..
Example	`VMProvisionStatusReadyValue 2` `VMProvisionStatusReadyValue 1-4,6,16`

VMSARESTATIC

Format	*<BOOLEAN>*
Default	*FALSE*
Description	When set to true, informs Moab that it can schedule under the assumption that no VMs will be migrated and no new VMs will be created, and disables Moab from scheduling any VM creations or migrations.
Example	`VMSARESTATIC TRUE`

VMSTALEACTION	
Format	One of the following: *IGNORE*, *CANCELTRACKINGJOB*, or *DESTROY*
Default	*IGNORE*
Description	Specifies the action that is applied to a stale VM, or a VM that the resource manager has not reported to Moab recently (see VMSTALEITERATIONS). • *IGNORE* specifies that Moab will take no action. • *CANCELTRACKING* specifies that Moab will remove the tracking job for stale VMs, but will not remove the actual VM (not recommended). • *DESTROY* specifies that Moab destroys stale VMs. If you specify *DESTROY*, you must also set the ENABLEVMDESTROY parameter to TRUE.
Example	`VMSTALEACTION DESTROY`

VMSTALEITERATIONS	
Format	*<INTEGER>*
Default	*5*
Description	Specifies the number of Moab iterations a VM must be unreported by any resource manager before it is considered "stale." To specify what happens with the VM after it has become stale, see VMSTALEACTION.
Example	`VMSTALEITERATIONS 3` *3 iterations must complete without a resource manager reporting a VM for it to be considered stale. This means that if the RMPOLLINTERVAL maximum is set to 2:00, a VM must be unreported for 6 minutes to be stale.*

VMSTORAGEMOUNTDIR	
Format	*<PATH>*
Default	---
Description	The specified path is used as the default location for storage mounts in all newly created VMs (created via the mvmctl command). This parameter defines the default storage mount directory if one is not specified.
Example	`VMSTORAGEMOUTDIR  /var/spool` *Moab uses /var/spool as a storage mount directory if a storage directory is not submitted (but additional storage is requested) at VM creation.*

VMTRACKING	
Format	*<STRING>*
Default	---
Description	When set to *TRUE*, VMTracking jobs are used to represent VMs in the queue.
Example	`VMTRACKING TRUE`

WALLTIMECAP	
Format	*<DOUBLE>*
Default	*0* (NO CAP)
Description	Specifies the maximum total pre-weighted absolute contribution to job priority which can be contributed by the walltime component. This value is specified as an absolute priority value, not as a percent.
Example	`WALLTIMECAP 10000` *Moab will bound a job's pre-weighted walltime priority component within the range +/- 10000.*

WALLTIMEWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight to be applied to the amount of walltime requested by a job (in seconds) (see Resource (RES) Factor).

Example	`RESWEIGHT      10`
	`WALLTIMEWEIGHT 100`
	Increase the priority of longer duration jobs.

WCACCURACYCAP

Format	*<DOUBLE>*
Default	*0* (NO CAP)
Description	Specifies the maximum total pre-weighted absolute contribution to job priority which can be contributed by the wallclock accuracy component. This value is specified as an absolute priority value, not as a percent.
Example	`WCACCURACYCAP 10000` *Moab will bound a job's pre-weighted wallclock accuracy priority component within the range +/- 10000.*

WCACCURACYWEIGHT

Format	*<INTEGER>*
Default	*0*
Description	Specifies the priority weight to be applied to the job's historical user wallclock accuracy (range 0.0 to 1.0) (see Fairshare (FS) Factor).

Example	`FSWEIGHT         10`
	`WCACCURACYWEIGHT 100`
	Favor jobs with good wallclock accuracies by giving them a priority increase.

WCVIOLATIONACTION

Format	one of *CANCEL* or *PREEMPT*
Default	*CANCEL*
Description	Specifies the action to take when a job exceeds its wallclock limit. If set to *CANCEL*, the job will be terminated. If set to *PREEMPT*, the action defined by PREEMPTPOLICY parameter will be taken. See JOBMAXOVERRUN or Usage-based Limits.

Example	`WCVIOLATIONACTION PREEMPT`
	`PREEMPTPOLICY REQUEUE`
	Moab will requeue jobs which exceed their wallclock limit.

WEBSERVICESURL

Format	*<URL>*
Default	---
Description	If specified, Moab sends data to Moab Web Services (MWS) to be stored in a database. This allows Moab to spend more cycles on scheduling instead of database interaction. The sending occurs via HTTP PUT.
Example	`WEBSERVICESURL http://mws-staging.ac:8080/mws/rm/moab/dump` *Moab sends data that needs to be stored in a database to the specified URL.*

WIKIEVENTS

Format	*<BOOLEAN>*
Default	*TRUE*
Description	When set to true, Moab events are set to native wiki format (ATTR=VALUE pairs) to facilitate easier readability .

Example	`WIKIEVENTS TRUE`
	Moab events will generate output in the format of the following sample:
	`09:26:40 1288279600:5 job 58 JOBEND 58 REQUESTEDNC=1 REQUESTEDTC=3 UNAME=wightman GNAME=wightman WCLIMIT=60 STATE=Completed RCLASS=[batch:1] SUBMITTIME=1288279493 RMEMCMP=>= RDISKCMP=>= RFEATURES=[NONE] SYSTEMQUEUETIME=1288279493 TASKS=1 FLAGS=RESTARTABLE PARTITION=pbs DPROCS=1 ENDDATE=2140000000 TASKMAP=proxy,GLOBAL SRM=pbs EXITCODE=0 SID=2357 NODEALLOCATIONPOLICY=SHARED EFFECTIVEQUEUEDURATION=107`

XFACTORCAP

Format	*<DOUBLE>*
Default	*0* (NO CAP)
Description	Specifies the maximum total pre-weighted absolute contribution to job priority which can be contributed by the expansion factor component. This value is specified as an absolute priority value, not as a percent.
Example	`XFACTORCAP 10000` *Moab will bound a job's pre-weighted XFactor priority component within the range +/- 10000.*

XFACTORWEIGHT	
Format	*<INTEGER>*
Default	*0*
Description	Specifies the weight to be applied to a job's minimum expansion factor before it is added to the job's cumulative priority.
Example	`XFACTORWEIGHT 1000` *Moab will multiply a job's XFactor value by 1000 and then add this value to its total priority.*

XFMINWCLIMIT	
Format	*[[[DD:]HH:]MM:]SS*
Default	*-1* (NO LIMIT)
Description	Specifies the minimum job wallclock limit that will be considered in job expansion factor priority calculations.
Example	`XFMINWCLIMIT 0:01:00` *Jobs requesting less than 1 minute of wallclock time will be treated as if their wallclock limit was set to 1 minute when determining expansion factor for priority calculations.*

Appendix B: Multi-OS Provisioning

- 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

```
> perl -e 'use Storable 2.18'
> perl -MXML::Simple -e 'exit'
> perl -MProc::Daemon -e 'exit'
> perl -MDBD::SQLite -e 'exit'
```

Integrating MSM and xCAT

Copy the `x_msm` table schema to the xCAT schema directory:

```
> cp $MSMHOMEDIR/contrib/xcat/MSM.pm $XCATROOT/lib/perl/xCAT_schema
```

Restart xcatd and check the `x_msm` table is correctly created:

```
> service xcatd restart
```

```
> tabdump x_msm
```

Prepare xCAT images and ensure they provision correctly (see xCAT documentation)

Populate the `x_msm` table with your image definitions:

```
> tabedit x_msm

#flavorname,arch,profile,os,nodeset,features,vmoslist,hvtype,hvgroupname,vmgroupname,co
mments,disable
   "compute","x86_64","compute","centos5.3","netboot","torque",,,,,,
   "science","x86","compute","scientific_linux","netboot","torque",,,,,,
```

- **flavorname** - A user specified name for the image and settings; also an xCAT group name, nodes are added to this group when provisioned
- **arch** - Architecture as used by xCAT
- **profile** - Profile as used by xCAT
- **os** - Operating system as used by xCAT
- **nodeset** - One of netboot|install|statelite
- **features** - Names of xCAT groups that identify special hardware features ('torque' and 'paravirt' are special cases)

- **vmoslist** - Note: Not used. List of flavorname's this image may host as VMs (hypervisor images only)
- **hvtype** - Note: Not used. One of esx|xen|kvm (hypervisor images only)
- **hvgroupname** - Note: Not used. Name of xCAT group nodes will be added to when provisioned to this image
- **vmgroupname** - Note: Not used. Name of xCAT group VMs will be added to when hosted on a hypervisor of this image
- **comments** - User specified comments
- **disable** - Flag to temporarily disable use of this image

Ensure all xCAT group names in the x_msm table exist in the xCAT nodegroup table

```
> tabedit nodegroup
```

Edit as necessary to simulate the following example:

```
   #groupname,grouptype,members,wherevals,comments,disable
   "compute",,,,,
   "esxi4",,,,,
   "esxhv",,,,,
   "esxvmmgt",,,,,
```

After making any necessary edits, run the following command:

```
> nodels compute,esxi4,esxhv,esxvmmgt
   # should complete without error, ok if doesn't return anything
```

MSM Configuration

Edit *$MSMHOMEDIR*/msm.cfg and configure the xCAT plug-in. Below is a generic example for use with TORQUE without virtualization. See the section on configuration parameters for a complete list of parameters and descriptions.

```
   # MSM configuration options
   RMCFG[msm]         PORT=24603
   RMCFG[msm]         POLLINTERVAL=45
   RMCFG[msm]         LOGFILE=/opt/moab/log/msm.log
   RMCFG[msm]         LOGLEVEL=8
   RMCFG[msm]         DEFAULTNODEAPP=xcat

   # xCAT plugin specific options
   APPCFG[xcat]       DESCRIPTION="xCAT plugin"
   APPCFG[xcat]       MODULE=Moab::MSM::App::xCAT
   APPCFG[xcat]       LOGLEVEL=3
   APPCFG[xcat]       POLLINTERVAL=45
   APPCFG[xcat]       TIMEOUT=3600
   APPCFG[xcat]       _USEOPIDS=0
   APPCFG[xcat]       _NODERANGE=moab,esxcompute
   APPCFG[xcat]       _USESTATES=boot,netboot,install
   APPCFG[xcat]       _LIMITCLUSTERQUERY=1
   APPCFG[xcat]       _RPOWERTIMEOUT=120
   APPCFG[xcat]       _DONODESTAT=1
   APPCFG[xcat]       _REPORTNETADDR=1
```

```
   APPCFG[xcat]      _CQXCATSESSIONS=4
```

Configuration Validation

Set up environment to manually call MSM commands:

```
# substitute appropriate value(s) for path(s)
   export MSMHOMEDIR=/opt/moab/tools/msm
   export MSMLIBDIR=/opt/moab/tools/msm
   export PATH=$PATH:/$MSMLIBDIR/contrib:$MSMLIBDIR/bin
```

Verify that MSM starts without errors:

```
> msmd
```

Verify that the expected nodes are listed, without errors, using the value of _NODERANGE from `msm.cfg`.

```
> nodels <_NODERANGE>
```

Verify that the expected nodes, are listed in the cluster query output from MSM:

```
> cluster.query.pl
```

Provision all nodes through MSM for the first time (pick and image name from `x_msm`):

```
> for i in `nodels <_NODERANGE>; do node.modify.pl $i --set os=<image_name>;done
```

Verify the nodes correctly provision and that the correct OS is reported (which may take some time after the provisioning requests are made):

```
> cluster.query.pl
```

Troubleshooting

- **msmctl -a does not report the xCAT plugin** - Check the log file (path specified in `msm.cfg`) for error messages. A common cause is missing Perl modules (Storable, DBD::SQLite, xCAT::Client).
- **cluster.query.pl does not report any nodes** - Check that the xCAT command nodels *<noderange>*, where *<noderange>* is the value configured for _NODERANGE in msm.cfg, outputs the nodes expected.
- **cluster.query.pl does not report OS** - MSM must provision a node to recognize what the current operating system is. It is not sufficient to look up the values in the `nodetype` table because MSM has no way of recognizing whether nodeset and rpower were run with the current values in the `nodetype` table.
- **cluster.query.pl does not report OSLIST, or does not report the expected OSLIST for a node** - Check that the node belongs to the appropriate groups, particularly any listed in the features field of the `x_msm` table for the missing image name.

Deploying Images with TORQUE

When using MSM + xCAT to deploy images with TORQUE, there are some special configuration considerations. Most of these also apply to other workload resource managers.

Note that while the MSM xCAT plugin contains support for manipulating TORQUE directly, this is not an ideal solution. If you are using a version of xCAT that supports prescripts, it is more appropriate to write prescripts that manipulate TORQUE based on the state of the xCAT tables. This approach is also applicable to other workload resource managers, while the xCAT plugin only deals with TORQUE.

Several use cases and configuration choices are discussed in what follows.

Each image should be configured to report its image name through TORQUE. In the TORQUE `pbs_mom` `mom_config` file the opsys value should mirror the name of the image. See 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/etc/moab.cfg`. A sample configuration file, set up and optimized for adaptive computing follows:

```
SCHEDCFG[Moab]          SERVER=gpc-sched:42559
ADMINCFG[1]             USERS=root,egan
LOGLEVEL                7

# How often (in seconds) to refresh information from TORQUE and MSM
RMPOLLINTERVAL           60,60
RESERVATIONDEPTH        10
DEFERTIME               0
TOOLSDIR                /opt/moab/tools

############################################################################
# TORQUE and MSM configuration                                             #
############################################################################
RMCFG[torque]           TYPE=PBS
RMCFG[msm]        TYPE=NATIVE:msm FLAGS=autosync,NOCREATERESOURCE RESOURCETYPE=PROV
RMCFG[msm]        TIMEOUT=60
RMCFG[msm]        PROVDURATION=10:00
AGGREGATENODEACTIONS    TRUE

############################################################################
# ON DEMAND PROVISIONING SETUP                                             #
############################################################################
QOSCFG[od]              QFLAGS=PROVISION
USERCFG[DEFAULT]        QLIST=od
NODEALLOCATIONPOLICY    PRIORITY
NODECFG[DEFAULT]        PRIORITYF=1000*OS+1000*POWER
NODEAVAILABILITYPOLICY  DEDICATED
CLASSCFG[DEFAULT]       DEFAULT.OS=scinetcompute

#################################################################
# GREEN POLICIES                                                #
#################################################################
```

```
NODECFG[DEFAULT]         POWERPOLICY=ONDEMAND
PARCFG[ALL]              NODEPOWEROFFDURATION=20:00
NODEIDLEPOWERTHRESHOLD   600
# END Example moab.cfg
```

Verifying the Installation

When Moab starts it immediately communicates with its configured resource managers. In this case Moab communicates with TORQUE to get compute node and job queue information. It then communicates with MSM to determine the state of the nodes according to xCAT. It aggregates this information and processes the jobs discovered from TORQUE.

When a job is submitted, Moab determines whether nodes need to be provisioned to a particular operating system to satisfy the requirements of the job. If any nodes need to be provisioned Moab performs this action by creating a provisioning system job (a job that is internal to Moab). This system job communicates with xCAT to provision the nodes and remain active while the nodes are provisioning. Once the system job has provisioned the nodes it informs the user's job that the nodes are ready at which time the user's job starts running on the newly provisioned nodes.

When a node has been idle for a specified amount of time (see NODEIDLEPOWERTHRESHOLD), Moab creates a power-off system job. This job communicates with xCAT to power off the nodes and 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  ResourceType: PROV
  Timeout:             30000.00 ms
  Cluster Query URL:   $HOME/tools/msm/contrib/cluster.query.xcat.pl
  Workload Query URL:  exec://$TOOLSDIR/msm/contrib/workload.query.pl
  Job Start URL:       exec://$TOOLSDIR/msm/contrib/job.start.pl
  Job Cancel URL:      exec://$TOOLSDIR/msm/contrib/job.modify.pl
  Job Migrate URL:     exec://$TOOLSDIR/msm/contrib/job.migrate.pl
  Job Submit URL:      exec://$TOOLSDIR/msm/contrib/job.submit.pl
  Node Modify URL:     exec://$TOOLSDIR/msm/contrib/node.modify.pl
  Node Power URL:      exec://$TOOLSDIR/msm/contrib/node.power.pl
  RM Start URL:        exec://$TOOLSDIR/msm/bin/msmd
  RM Stop URL:         exec://$TOOLSDIR/msm/bin/msmctl?-k
  System Modify URL:   exec://$TOOLSDIR/msm/contrib/node.modify.pl
  Environment:
MSMHOMEDIR=/home/wightman/test/scinet/tools//msm;MSMLIBDIR=/home/wightman/test/scinet/
tools//msm
  Objects Reported:    Nodes=10 (0 procs)  Jobs=0
  Flags:               autosync
  Partition:           SHARED
  Event Management:    (event interface disabled)
  RM Performance:      AvgTime=0.10s  MaxTime=0.25s  (38 samples)
  RM Languages:        NATIVE
  RM Sub-Languages:    -
```

To verify nodes are configured to provision use the checknode -v command. Each node will have a list of available operating systems.

```
$ checknode n01
node n01
```

```
State:      Idle  (in current state for 00:00:00)
Configured Resources: PROCS: 4  MEM: 1024G  SWAP: 4096M  DISK: 1024G
Utilized   Resources: ---
Dedicated  Resources: ---
Generic Metrics:    watts=25.00,temp=40.00
Power Policy:       Green (global policy)   Selected Power State: Off
Power State:   Off
Power:      Off
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      compute   Arch:      ---
  OS Option: compute
  OS Option: computea
  OS Option: gpfscompute
  OS Option: gpfscomputea
Speed:      1.00      CPULoad:   0.000
Flags:      rmdetected
RM[msm]:    TYPE=NATIVE:MSM  ATTRO=POWER
EffNodeAccessPolicy: SINGLEJOB
Total Time: 00:02:30  Up: 00:02:19 (92.67%)  Active: 00:00:11 (7.33%)
```

To verify nodes are configured for Green power management, run the mdiag -G command. Each node will show its power state.

```
$ mdiag -G
NOTE:  power management enabled for all nodes
Partition ALL:  power management enabled
  Partition NodeList:
Partition local:  power management enabled
  Partition NodeList:
  node n01 is in state Idle, power state On (green powerpolicy enabled)
  node n02 is in state Idle, power state On (green powerpolicy enabled)
  node n03 is in state Idle, power state On (green powerpolicy enabled)
  node n04 is in state Idle, power state On (green powerpolicy enabled)
  node n05 is in state Idle, power state On (green powerpolicy enabled)
  node n06 is in state Idle, power state On (green powerpolicy enabled)
  node n07 is in state Idle, power state On (green powerpolicy enabled)
  node n08 is in state Idle, power state On (green powerpolicy enabled)
  node n09 is in state Idle, power state On (green powerpolicy enabled)
  node n10 is in state Idle, power state On (green powerpolicy enabled)
Partition SHARED:  power management enabled
```

To submit a job that dynamically provisions compute nodes, run the msub -l os=<*image*> command.

```
$ msub -l os=computea job.sh
yuby.3
$ showq
active jobs------------------------
JOBID              USERNAME      STATE PROCS   REMAINING            STARTTIME
provision-4            root    Running     8    00:01:00  Fri Jun 19 09:12:56
1 active job               8 of 40 processors in use by local jobs (20.00%)
                           2 of 10 nodes active      (20.00%)
eligible jobs----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME
yuby.3             wightman       Idle     8    00:10:00  Fri Jun 19 09:12:55
1 eligible job
blocked jobs-----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME

0 blocked jobs
Total jobs:  2
```

Notice that Moab created a provisioning system job named `provision-4` to provision the nodes. When `provision-4` detects that the nodes are correctly provisioned to the requested OS, the submitted job `yuby.3` runs:

```
$ showq
active jobs------------------------
JOBID              USERNAME      STATE PROCS   REMAINING            STARTTIME
yuby.3             wightman    Running     8    00:08:49  Fri Jun 19 09:13:29
1 active job               8 of 40 processors in use by local jobs (20.00%)
                           2 of 10 nodes active      (20.00%)
eligible jobs----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME

0 eligible jobs
blocked jobs-----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME

0 blocked jobs
Total job:  1
```

The checkjob command shows information about the provisioning job as well as the submitted job. If any errors occur, run the checkjob -v *<jobid>* command to diagnose failures.

xCAT Plug-in Configuration Parameters

Plugin parameters that begin with an underscore character are specific to the xCAT plug-in; others are common to all plug-ins and may either be set in the RMCFG[msm] for all plug-ins, or per plug-in in the APPCFG[*<plugin_name>*].

Description
Module
LogLevel
PollInterval
TimeOut
_NodeRange
_CQxCATSessions
_DORVitals
_PowerString
_DoNodeStat
_DoxCATStats
_LockDir
_HVxCATPasswdKey
_FeatureGroups
_DefaultVMCProc
_DefaultVMDisk
_DefaultVMCMemory
_KVMStoragePath
_ESXStore
_ESXCFGPath
_VMInterfaces
_XenHostInterfaces
_KVMHostInterfaces
_VMSovereign
_UseStates
_ImagesTabName
_VerifyRPower
_RPowerTimeOut
_QueueRPower
_RPowerQueueAge
_RPowerQueueSize
_MaskOSWhenOff
_ModifyTORQUE
_ReportNETADDR
_UseOpIDs
_VMIPRange
_xCATHost
_NoRollbackOnError

Description	
Format	Double quoted string containing brief description of plugin.
Default	---
Description	This information is not visible in Moab, but shows up in msmctl -a.

Module	
Format	Moab::MSM::App::xCAT
Default	---
Description	Name of the plugin module to load.

LogLevel	
Format	1-9
Default	*5*
Description	Used to control the verbosity of logging, 1 being the lowest (least information logged) and 9 being the highest (most information logged). For initial setup and testing, 8 is recommended, then lowering to 3 (only errors logged) for normal operation. Use 9 for debugging, or when submitting a log file for support.

PollInterval	
Format	Integer > 0
Default	*60*
Description	MSM will query xCAT every POLLINTERVAL seconds to update general node status. This number will likely require tuning for each specific system. In general, to develop this number, you should pick a fraction of the total nodes MSM will be managing (1/_CQXCATSESSIONS), and time how long it takes run nodestat, rpower stat, and optionally rvitals on these nodes, and add ~15%. Increasing the POLLINTERVAL will lower the overall load on the xCAT headnode, but decrease the responsiveness to provisioning and power operations.

TimeOut	
Format	Integer value > POLLINTERVAL
Default	*300*
Description	This parameter controls how long MSM will wait for child processed to complete (all xCAT commands are run in child processes). After TIMEOUT seconds, if a child has not returned it will be killed, and an error reported for the operation.

_NodeRange	
Format	Any valid noderange (see the xCAT noderange manpage).
Default	All
Description	When MSM queries xCAT this is the noderange it will use. At sites where xCAT manages other hardware that Moab is not intended to control, it is important to change this.

_CQxCATSessions	
Format	Positive integer > 1
Default	*10*
Description	MSM will divide the node list generated by nodels into this many groups and simultaneously query xCAT for each group. The value may need tuning for large installations, higher values will cause the time to complete a single cluster query to go down, but cause a higher load on the xCAT headnode.

_DORVitals	
Format	*0* or *1*
Default	*0*
Description	When set to *1*, MSM will poll rvitals power and led status (see the xCAT rvitals manpage). This only works with IBM BMCs currently. In order to use this, xCAT should respond without error to the rvitals *<noderange>* watts and rvitals *<noderange>* leds commands. Status is reported as GMETRTIC [watts] and GMETRIC[leds]. See also the **_PowerString** on page 959 configuration parameter.

_PowerString	
Format	single quote delimited string
Default	*'AC Avg Power'*
Description	Only meaningful when used with **_DORVitals** on page 959=*1*. Some BMCs return multiple responses to the rvitals command, or use slightly different text to describe the power metrics. Use this parameter to control what is reported to Moab. You can use *'$MSMLIBDIR/contrib/xcat/dump.xcat.cmd.pl rvitals <node_name> power'* and examine the output to determine what the appropriate value of this string is.

_DoNodeStat	
Format	*0* or *1*
Default	*1*
Description	If set to *0*, MSM will not call nodestat to generated a substate. This can be used to speed up the time it takes to query xCAT, and you do not need the substate visible to Moab.

_DoxCATStats	
Format	*0* or *1*
Default	*0*
Description	If Set to *1*, MSM will track performance statistics about calls to xCAT, and the performance of higher level operations. The information is available via the script *$MSMHOMEDIR*/contrib/xcat/xcatstats.pl. This parameter is useful for tuning the POLLINTERVAL and _CQxCATSessions on page 959 configuration parameters.

_LockDir	
Format	Existing path on MSM host
Default	*$MSMHOMEDIR/lck*
Description	This is a path to where MSM maintains lock files to control concurrency with some Xen and KVM operations.

_HVxCATPasswdKey	
Format	key value in the xCAT passwd table
Default	*vmware*
Description	This is where MSM gets the user/password to communicate with ESX hypervisors.

_FeatureGroups	
Format	Comma delimited string of xCAT group names.
Default	---
Description	MSM builds the OSLIST for a node as the intersection of _FEATUREGROUPS, features specified in `x_msm` for that image, and the nodes group membership. The value 'torque' is special, and indicates that the image uses TORQUE, and the node should be added/removed from TORQUE during provisioning when used in conjunction with the **_ModifyTORQUE** on page 965 parameter.

_DefaultVMCProc	
Format	1-?
Default	*1*
Description	If not explicitly specified in the create request, MSM will create VMs with this many processors.

_DefaultVMDisk	
Format	Positive integer values, minimum is determined by your vm image needs
Default	*4096*
Description	If not explicitly specified in the create request, MSM will create VMs with this much disk allocated.

_DefaultVMCMemory	
Format	Positive integer values, minimum is determined by your vm image needs
Default	*512*
Description	If not specified, MSM will create VMs with this much memory allocated.

_KVMStoragePath	
Format	Existing path on MSM host

_KVMStoragePath	
Default	*/vms*
Description	File backed disk location for stateful KVM VMS will be placed here.

_ESXStore	
Format	Mountable NFS Path
Default	---
Description	Location of ESX stores.

_ESXCFGPath	
Format	Mountable NFS Path
Default	*ESXStore*
Description	Location of ESX VM configuration files.

_VMInterfaces	
Format	Name of bridge device in your VM image
Default	*br0*
Description	Bridge device name passed to libvirt for network configuration of VMs (overrides _ XENHOSTINTERFACES and _KVMHOSTINTERFACES if specified).

_XenHostInterfaces	
Format	Name of bridge device in your VM image
Default	*xenbr0*
Description	Bridge device name passed to libvirt for network configuration of Xen VMs.

_KVMHostInterfaces	
Format	Name of bridge device in your VM image
Default	*br0*
Description	Bridge device name passed to libvirt for network configuration of KVM VMs.

_VMSovereign	
Format	*0* or *1*
Default	*0*
Description	Setting this attribute will cause Moab to reserve VMs' memory and procs on the hypervisor and treat the VM as the workload — additional workload cannot be scheduled on the VM.

_UseStates	
Format	Valid xCAT chain.currstate values (see the xCAT chain manpage)
Default	*boot,netboot,install*
Description	Nodes that do not have one of these values in the xCAT `chain.currstate` field will reported with **STATE**=*Updating*. Use this configuration parameter to prevent Moab from scheduling nodes that are updating firmware, etc.

_ImagesTabName	
Format	Existing xCAT table that contains your image definitions.
Default	*x_msm*
Description	This table specifies the images that may be presented to Moab in a nodes `OSLIST`. The xCAT schema for this table is defined in *$MSMHOMEDIR*`/contrib/xcat/MSM.pm`, which needs to be copied to the *$XCATROOT*`/lib/perl/xCAT_schema` directory.

_VerifyRPower	
Format	*0* or *1*
Default	*0*
Description	If set, MSM will attempt to confirm that rpower requests were successful by polling the power state with rpower stat until the node reports the expected state, or **_RPowerTimeOut** on page 964 is reached. **NOTE**: This can create significant load on the xCAT headnode.

_RPowerTimeOut	
Format	Positive integer values
Default	*60*
Description	Only meaningful when used with **_VerifyRPower** on page 964. If nodes do not report the expected power state in this amount of time, a GEVENT will be produced on the node (or system job).

_QueueRPower	
Format	*0* or *1*
Default	*0*
Description	When set, this parameter will cause MSM to aggregate rpower requests to xCAT into batches. The timing and size of these batches is controlled with the **_RPowerQueueAge** on page 964 and **_RPowerQueueSize** on page 965 parameters. **NOTE**: This can significantly reduce load on the xCAT headnode, but will cause the power commands to take longer, and MSM shutdown to take longer.

_RPowerQueueAge	
Format	Positive integer values
Default	*30*
Description	Only meaningful when used with **_QueueRPower** on page 964. MSM will send any pending rpower requests when the oldest request in the queue exceeds this value (seconds).

_RPowerQueueSize	
Format	Positive integer values
Default	*200*
Description	Only meaningful when used with **_QueueRPower** on page 964. MSM will send any pending rpower requests when the queue depth exceeds this value.

_MaskOSWhenOff	
Format	*0* or *1*
Default	*0*
Description	When set, this parameter will cause MSM to report **OS**=*None* for nodes that are powered off. This may be useful when mixing stateless and stateful images, forcing Moab to request provisioning instead of just powering on a node.

_ModifyTORQUE	
Format	*0* or *1*
Default	*0*
Description	When set, this parameter will cause MSM to add and removes nodes and VMs from TORQUE as required by provisioning. See the **_FeatureGroups** on page 961 parameter as well.

_ReportNETADDR	
Format	*0* or *1*
Default	*0*
Description	When set, this parameter will cause MSM to report NETADDR=<hosts.ip from xCAT>.

_UseOpIDs	
Format	*0* or *1*

_UseOpIDs	
Default	*0*
Description	When set, this parameter will cause errors to be reported as GEVENTs on the provided system job, instead of a node (Moab 5.4 only, with appropriate Moab CFG)

_VMIPRange	
Format	Comma separated list of dynamic ranges for VM (ex '10.10.23.100-200,10.10.24.1-255')
Default	---
Description	Use this parameter to specify a pool of IPs that MSM should assign to VMs at creation time. IPs are selected sequentially from this list as available. Omit this configuration parameter if an external service is managing IP assignment, or if they are all previously statically assigned.

_xCATHost	
Format	*<xcat_headnode>:<xcatd_port>*
Default	*localhost:3001*
Description	Use to configure MSM to communicate with xCAT on another host.

_NoRollbackOnError	
Format	*0* or *1*
Default	*0*
Description	When an error occurs and rollback is activated (as it is by default), rollback causes a reversion to the previous successful request. _NoRollbackOnError is useful for debugging to determine the xCAT state if no rollback occurred. If set to 1 and an error occurs between MSM and xCAT when creating a node, assigning a name (DNS) to a node, or assigning an IP address (DHCP) to a node, then no rollback occurs.

Appendix D: Adjusting Default Limits

Moab is distributed in a configuration capable of supporting multiple architectures and systems ranging from a few processors to several thousand processors. However, in spite of its flexibility, for performance reasons, it still contains a number of default object limits parameters and static structures defined in header files. These limits constrain such things as the maximum number of jobs, reservations, and nodes that Moab can handle and are set to values that provide a reasonable compromise between capability and memory consumption for most sites. However, many site administrators want to increase some of these settings to extend functionality, or decrease them to save consumed memory. The most common parameters are listed in what follows. Parameters listed in the Moab configuration file (`moab.cfg`) can be modified by restarting Moab. To change parameters listed in `moab.h`, please contact technical support.

CLIENTMAXCONNECTIONS	
Location	`moab.cfg` (dynamic parameter)
Default	128
Max tested	---
Description	Maximum number of connections that can simultaneously connect to Moab.

JOBMAXNODECOUNT	
Location	`moab.cfg` (dynamic parameter)
Default	1024
Max tested	8192
Description	Maximum number of compute nodes that can be allocated to a job. After changing this parameter, Moab must be restarted for changes to take effect. The value cannot exceed that of the **MAXNODE** on page 869 parameter (specified in moab.cfg). If you specify a value higher than the limit set for the MAXNODE parameter, the value will match MAXNODE. JOBMAXNODECOUNT can also be specified within configure using --with-maxjobsize=<NODECOUNT>.

MAXGRES	
Location	`moab.cfg` (dynamic parameter)

MAXGRES	
Default	512
Max tested	---
Description	Total number of distinct generic resources that can be managed.

MAXJOB	
Location	`moab.cfg` (dynamic parameter)
Default	4096
Max tested	500,000
Description	Maximum number of jobs that can be evaluated simultaneously. (Can also be specified within **configure** using **--with-maxjobs=<JOBCOUNT>**.)

MAXRSVPERNODE	
Location	`moab.cfg` (dynamic parameter)
Default	48
Max tested	1024
Description	Maximum number of reservations a node can simultaneously support.

MMAX_ATTR	
Location	`moab.h`
Default	128
Max tested	512
Description	Total number of distinct node attributes (PBS node attributes/LL node features) that can be tracked.

MMAX_CLASS	
Location	`moab.h`
Default	256
Max tested	256
Description	Total number of distinct job classes/queues available.

MMAX_FSDEPTH	
Location	`moab.h`
Default	24
Max tested	32
Description	Number of active fairshare windows.

MAXNODE	
Location	`moab.cfg` (dynamic parameter
Default	5120
Max tested	160000
Description	Maximum number of compute nodes supported.

MMAX_PAR	
Location	`moab.h`
Default	32
Max tested	32
Description	Maximum number of partitions supported.

MMAX_QOS	
Location	moab.h
Default	128
Max tested	128
Description	Total number of distinct QoS objects available to jobs.

MMAX_RACK	
Location	moab.h
Default	200
Max tested	200
Description	Total number of distinct rack objects available within cluster.

MMAX_RANGE	
Location	moab.h
Default	2048
Max tested	2048
Description	Total number of distinct timeframes evaluated. **Note**: This is proportional to the size of the cluster and the number of simultaneously active jobs in the cluster. (Can be specified within `./configure` using `--with-maxrange=<RANGECOUNT>`.) Increasing this value will not increase the size of total memory consumed by Moab but may result in minor slowdowns in the evaluation and optimization of reservations.

MMAX_REQ_PER_JOB	
Location	moab.h
Default	5

MMAX_REQ_PER_JOB	
Max tested	64
Description	Total number of unique requirement structures a job can have. Limits the number of `-w` clauses in the mshow -a command. It also limits the number of `-l nodes=X+Y+Z` a normal HPC job can have.

JOBMAXTASKCOUNT	
Location	`moab.cfg` (dynamic parameter
Default	4096
Max tested	16000
Description	Total number of tasks allowed per job.

Moab currently possesses hooks to allow sites to create local algorithms for handling site specific needs in several areas. The `contrib` directory contains a number of sample local algorithms for various purposes. The `MLocal.c` module incorporates the algorithm of interest into the main code. The following scheduling areas are currently handled via the `MLocal.c` hooks.

- Local Job Attributes
- Local Node Allocation Policies
- Local Job Priorities
- Local Fairness Policies

Related topics

- Appendix I: Considerations for Large Clusters

Appendix E: Security

Moab provides role and host based authorization, encryption*, and DES, HMAC, and MD5 based authentication. The following sections describe these features in more detail.

- Authorization
  - Role Based Authorization Security Configuration
- Authentication

    - Mauth Authentication
    - Munge Authentication
    - Server Response Control
    - Interface Development Notes
- Host Security
    - Minimal Host Security Enforcement
    - Medium Host Security Enforcement
    - Strict Host Security Enforcement
- Access Portal Security

Authorization

Role Based Authorization Security Configuration

Moab provides access control mechanisms to limit how the scheduling environment is managed. The primary means of accomplishing this is through limiting the users and hosts that are trusted and have access to privileged commands and data.

With regard to users, Moab breaks access into three distinct levels.

Level 1 Moab Admin (Administrator Access)

Level 1 Moab administrators have global access to information and unlimited control over scheduling operations. By default, they are allowed to control scheduler configuration, policies, jobs, reservations, and all scheduling functions. They are also granted access to all available statistics and state information. Level 1 administrators are specified using the ADMINCFG[1] parameter.

Level 2 Moab Admin (Operator Access)

Level 2 Moab administrators are specified using the ADMINCFG[2] parameter. By default, the users listed under this parameter are allowed to change all job attributes and are granted access to all informational Moab commands.

Level 3 Moab Admin (Help Desk Access)

Level 3 administrators are specified via the ADMINCFG[3] parameter. By default, they are allowed access to all informational Moab commands. They cannot change scheduler or job attributes.

Configuring Role Based Access

Moab allows site specific tuning of exactly which functions are available to each administrator level. Moab also provides two additional administrator levels (**ADMINCFG[4]** and **ADMINCFG[5]**) that may be used for site specific needs.

To configure Moab role based access, use the ADMINCFG parameter.

```
ADMINCFG[1]    USERS=root,john SERVICES=ALL NAME=admin
ADMINCFG[3]    USERS=joe,mary  SERVICES=mdiag,mrsvctl,mcredctl NAME=power
```

```
ADMINCFG[5]    USERS=joy,blake SERVICES=NONE NAME=users
...
```

A *NONE* in services will still allow users to run showq and checkjob on their own jobs.

To determine the role of system users and what commands they can run, use the mcredctl -q role user:<USERID> command.

Using the **SERVICES** attribute of the **ADMINCFG** parameter, access to an arbitrary selection of services can be enabled on a per administrator-level basis. Possible services include the following:

Service	Description
changeparam	Change any scheduling policy or parameter (This command is deprecated. Use mschedctl -m instead).
checkjob	View detailed information for any job.
checknode	View detailed information for any node.
mbal	Perform real-time load-balancing of interactive commands.
mcredctl	View and modify credential attributes.
mdiag	Provide diagnostic reports for resources, workload, and scheduling.
mjobctl	Modify, control, and view jobs.
mnodectl	Modify, control, and view nodes.
mrmctl	Modify, control, and view resource managers.
mrsvctl	Modify, control, and view reservations.
mschedctl	Modify, control, and view scheduler behavior.
mshow	View existing configuration and predicted resource availability.
showstats	View all scheduler and credential statistics.
releaseres	Release all reservations (This command is deprecated. Use mrsvctl -r instead).

Service	Description
runjob	Immediately execute any job (see mjobctl -x).
setqos	Set QoS on any job (This command is deprecated. Use mjobctl -m instead).
setres	Create any reservation (This command is deprecated. Use mrsvctl -c instead).
setspri	Set system priority on any job (This command is deprecated. Use mjobctl -p instead).
showconfig	Show all scheduler configuration parameters (This command is deprecated. Use mschedctl -l instead).
showres	Show detailed information for any reservation.
showstate	Show detailed information for all jobs, including their locations, and display job error messages, if any.

Account and Class/Queue Admins

While the **ADMINCFG** parameter allows organizations to provide controlled access to scheduling objects, it does not allow for distribution along organizational boundaries. For example, a site may set up a level 3 administrator to be able to view statistics, diagnose jobs, and modify job priorities; it does not provide a way to differentiate one type of job from another. If a site administrator wanted to allow control based on the queue or account associated with a job, they would best accomplish this using the credential **MANAGERS** attribute.

A credential manager allows a user to be trusted to administer workload and policies for an associated subgroup of jobs. For example, in the configuration below, a number of queue and account managers are configured.

```
CLASSCFG[orion] MANAGERS=johns
CLASSCFG[xray]  MANAGERS=steve2
CLASSCFG[gamma] MANAGERS=steve2,jpw
ACCOUNTCFG[bio] MANAGERS=charles
```

By default, the specified managers can do anything to a job that the actual job owner could do. By default, this would include the ability to view cumulative and per job statistics, see job details, modify job priorities and holds, cancel and preempt jobs, and otherwise adjust policies and constraints within the associated credential.

Authentication (Interface Security)

Moab supports password-challenge, DES, HMAC, and MD5 based authentication. Authentication protocols may be specified on a per interface basis allowing independent realms of trust with per realm secret keys and even per realm authentication protocols.

Mauth Authentication

Mauth is a tool provided with Moab that provides client authentication services. With mauth enabled, each client request is packaged with the client ID, a timestamp, and an encrypted key of the entire request generated using the shared secret key.

This tool is enabled by providing a secret key. A random key is selected when the Moab `./configure` script is run and may be regenerated at any time by rerunning `./configure` and rebuilding Moab. If desired, this random key may be overridden by specifying a new key in the protected `.moab.key` file as in the example below:

Moab must be shut down before setting a new secret key. Use the `service moab stop` or mschedctl -k commands to shut down Moab.

```
> vi /opt/moab/etc/.moab.key
(insert key)
> cat /opt/moab/etc/.moab.key
XXXXXXXX
# secure file by setting owner read-only permissions
> chmod 400 /opt/moab/etc/.moab.key
# verify file is owned by root and permissions allow only root to read file
> ls -l /opt/moab/etc.moab.key
-r-------- 1 root root 15 2007-04-05 03:47 /opt/moab/etc/.moab.key
```

All directories in the path containing `.moab.key` must be owned by the root or primary Moab user It must not be writable by "other" in its permissions.

If `.moab.key` is used, this protected file will need to be on each host that is authorized to run Moab client commands.

By default, this file will be owned by the user root and its contents will be read by the mauth tool which provides client authorization services. If desired, the ownership of this file can be changed so long as this file is readable by the Moab server and the mauth tool. This can be accomplished if the Moab primary administrator, the owner of mauth, and the owner of `.moab.key` are the same.

By default, it is up to the individual cluster administrators to determine whether to use the `.moab.key` file. For sites with source code, the use of `.moab.key` can be mandated by using `./configure --with-keyfile`.

By default, mauth is located in the install `bin` directory. If an alternate name or alternate file location is desired, this can be specified by setting the **AUTHCMD** attribute of the CLIENTCFG parameter within the `moab.cfg` file as in the following example.

```
CLIENTCFG  AUTHCMD=/opt/sbin/mauth
```

Configuring Peer-Specific Secret Keys

Peer-specific secret keys can be specified using the CLIENTCFG parameter. This key information must be kept secret and consequently can only be specified in the `moab-private.cfg` file. With regard to security, there are two key attributes that can be set. (Other resource managers or clients such as Moab Accounting Manager or a SLURM/Wiki interface can also use the attributes to configure their authentication algorithms. The default, unless otherwise stated, is always *DES*. These attributes are listed in the table below:

AUTH	
Format	one of *ADMIN1*, *ADMIN2*, or *ADMIN3*
Default	---
Description	Specifies the level of control/information available to requests coming from this source/peer.
Example	`CLIENTCFG[RM:clusterB] AUTH=admin1 KEY=14335443`

AUTHTYPE	
Format	one of *DES*, *HMAC*, *HMAC64*, or *MD5*.
Default	*DES*
Description	Specifies the encryption algorithm to use when generating the message checksum.
Example	`CLIENTCFG[AM:mam] AUTHTYPE=HMAC64`

HOST	
Format	*<STRING >*
Default	---
Description	Specifies the host name of the remote peer. Peer requests coming from this host will be authenticated using the specified mechanism. This parameter is optional.
Example	`CLIENTCFG[RM:clusterA] HOST=orx.pb13.com  KEY=banana6`

KEY	
Format	*<STRING >*
Default	---
Description	Specifies the shared secret key to be used to generate the message checksum.
Example	`CLIENTCFG[RM:clusterA] KEY=banana6`

The **CLIENTCFG** parameter takes a string index indicating which peer service will use the specified attributes. In most cases, this string is simply the defined name of the peer service. However, for the special cases of resource and allocation managers, the peer name should be prepended with the prefix `RM:` or `AM:` respectively, as in `CLIENTCFG[AM:mam]` or `CLIENTCFG[RM:devcluster]`.

The first character of any secret key can be viewed by trusted administrators using specific diagnostic commands to analyze Moab interfaces. If needed, increase the length of the secret keys to maintain the desired security level.

Munge Authentication

Moab also integrates with MUNGE, an open source authentication service created by Lawrence Livermore National Laboratory (http://home.gna.org/munge/). MUNGE works with Moab to authenticate user credentials being passed between the Moab client and the Moab server or from Moab server to Moab server.

To set up MUNGE in a cluster, download and install MUNGE on every node in the cluster by following the installation steps found at http://home.gna.org/munge/. The MUNGE secret key must reside on each node in the cluster. Before starting the Moab daemon, the MUNGE daemon must be running on all nodes.

To enable Moab to use MUNGE for authentication purposes, specify the MUNGE executable path in the moab.cfg file using **CLIENTCFG** and **AUTHCMD** as in the following example. The MUNGE executable path must reside in each client's `moab.cfg` file as well.

```
CLIENTCFG AUTHCMD=/usr/bin/munge
```

Moab requires that the MUNGE and UNMUNGE executable names be "munge" and "unmunge" respectively. It also assumes that the UNMUNGE executable resides in the same directory as the MUNGE executable.

Configuring Munge Command Options

Moab also integrates with MUNGE command line options. For example, to set up Moab to use a specific socket that was created when the MUNGE daemon was started, use **CLIENTCFG** and **AUTHCMDOPTIONS** to specify the newly created socket. The **AUTHCMDOPTIONS** attribute, like **AUTHCMD**, must also reside in the client's `moab.cfg` file.

```
CLIENTCFG     AUTHCMD=/usr/bin/munge
CLIENTCFG     AUTHCMDOPTIONS="-S /var/run/munge/munge.socket.2"
```

Server Response Control

If a request is received that is corrupt or cannot be authenticated, Moab will report some limited information to the client indicating the source of the failure, such as "bad key," "malformed header," and so forth. In the case of highly secure environments, or to minimize the impact of sniffing or denial of service attacks, Moab can be configured to simply drop invalid requests. This is accomplished by adding the **DROPBADREQUEST** attribute to the **CLIENTCFG** parameter in the `moab-private.cfg` file as in the following example:

```
CLIENTCFG[DEFAULT] DROPBADREQUEST=TRUE
```

Interface Development Notes

Sample checksum generation algorithm code can be found in the Socket Protocol Description document.

Host Security for Compute Resources

Host level security can vary widely from one site to another with everything from pure on-your-honor based clusters to complete encrypted VLAN based network security and government approved per job scrubbing procedures being used. The following documentation describes some best practices in use throughout the industry.

Minimal Host Security Enforcement

For minimal host security, no additional configuration is required.

Medium Host Security Enforcement

- Login Access
  - PAM — Enable/disable access by modifying `/etc/security/access.conf`.
- Processes
  - Kill all processes associated with job user (dedicated).
  - Kill all processes associated with job session (dedicated/shared). Use `ps -ju <USER>` or `ps -js <SESSID>`.
- IPC (Inter-Process Communication)
  - Remove shared memory, semaphores, and message queues (use ipcs/ipcrm).
  - Remove named pipes.
- Network/Global File System Access
  - Explicitly unmount user home and global file systems.

- Local Temporary File Systems
  - Where possible, mount local file systems read-only.
  - Clear `/tmp`, `/scratch` and other publicly available local file systems.
  - Remove user files with shred; shred is a Linux command that first overwrites files completely before removing them, preventing remnant data from surviving on the hard drive.

Strict Host Security Enforcement

- VLAN creation
- Host rebuild
  - U.S Dept of Energy Disk/File Sanitization (SCRUB)
  - U.S Dept of Defense Scrubbing Software (DOD-5520)

Moab Access Portal Security Overview

The Moab Access Portal (MAP) security model is composed of several different components. First, users will use a Web browser to log in and interact with the Web server running MAP. This communication can be encrypted using industry standard SSL to protect usernames/passwords and other sensitive information that may be accessed by the user. (Instructions on how to set up SSL connections with popular Web servers and servlet engines are readily available on the Internet. A guide for setting up SSL with Apache is available in the MAP documentation.)

When a user logs in via their Web browser, the JSP interface passes this request to a back-end Java infrastructure that then uses an encrypted SSH connection to authenticate the user's credentials at the cluster head node. After the credentials are authenticated and the SSH connection established, further communication between MAP and the cluster 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 G: Integrating Other Resources with Moab

Moab can interface with most popular resource managers, many cluster services, and numerous general protocols. The following links provide additional information.

Compute Resource Managers

- TORQUE - Integration Guide, TORQUE documentation
- SLURM - Integration Guide, http://www.llnl.gov/linux/slurm
- WIKI - WIKI Integration Guide
- Cray XT/TORQUE - Integration Guide (html, pdf), http://www.cray.com

Provisioning Resource Managers

- xCAT - Validating an xCAT Installation for Use with Moab

Hardware Integration

- NUMA - Integration Guide

Compute Resource Managers

- **Moab-TORQUE Integration Guide** on page 980
- **Moab-SLURM Integration Guide** on page 984
- **Installation Notes for Moab and TORQUE for Cray** on page 988

Moab-TORQUE Integration Guide

- Overview
- Integration Steps
  - Install TORQUE
  - Install Moab
  - Configure TORQUE
  - TORQUE/Moab Considerations
- Current Limitations
- Troubleshooting

Install TORQUE

- Install TORQUE

Keep track of the PBS target directory, *$PBSTARGDIR*

Install Moab

- Untar the Moab distribution file.
- Change the directory to the `moab-<version>` directory.
- Run `./configure`.
- Specify the PBS target directory (**$PBSTARGDIR** from step 2.1) when queried by `./configure`.

Moab interfaces to PBS by utilizing a few PBS libraries and include files. If you have a non-standard PBS installation, you may need to modify `Makefile` and change **PBSIP** and **PBSLP** values and references as necessary for your local site configuration.

The ./configure script automatically sets up Moab so that the user running configure will become the default Primary Moab Administrator (*$MOABADMIN*). This can be changed by modifying the **ADMINCFG** [1] **USERS**=*<USERNAME>* line in the Moab configuration file (moab.cfg). The primary administrator is the first user listed in the **USERS** attribute and is the ID under which the Moab daemon runs.

Some Tru64 and IRIX systems have a local libnet library that conflicts with PBS's libnet library. To resolve this, try setting **PBSLIB** to '*${PBSLIBDIR}/libnet.a* **-lpbs**' in the Moab Makefile.

Moab is 64-bit compatible. If PBS/TORQUE is running in 64-bit mode, Moab likewise needs to be built in this manner to use the PBS scheduling API (i.e., for IRIX compilers, add *-64* to **OSCCFLAGS** and **OSLDFLAGS** variables in the Makefile).

General Configuration for All Versions of TORQUE

- Make $MOABADMIN a PBS admin.
  - By default, Moab only communicates with the pbs_server daemons and the *$MOABADMIN* should be authorized to talk to this daemon (See suggestions for more information.).
- *(OPTIONAL)* Set default PBS queue, nodecount, and walltime attributes (See suggestions for more information.).
- *(OPTIONAL - TORQUE Only)* Configure TORQUE to report completed job information by setting the qmgrkeep_completed parameter:>

```
> qmgr -c 'set server keep_completed = 300'
```

PBS nodes can be configured as *time shared* or *space shared* according to local needs. In almost all cases, space shared nodes provide the desired behavior.

PBS/TORQUE supports the concept of virtual nodes. Using this feature, Moab can individually schedule processors on SMP nodes. The online TORQUE documentation describes how to set up the $PBS_HOME/server_priv/nodes file to enable this capability. (For example, *<NODENAME>* **np**=*<VIRTUAL NODE COUNT>*)

Version-Specific Configuration for TORQUE

Do not start the pbs_sched daemon. This is the default scheduler for TORQUE; Moab provides this service.

Moab uses PBS's scheduling port to obtain real-time event information from PBS regarding job and node transitions. Leaving the default qmgr setting of set **server scheduling**=*True* allows Moab to receive and process this real-time information.

Configure Moab

By default, Moab automatically interfaces with TORQUE/PBS when it is installed. Consequently, in most cases, the following steps are not required:

- Specify PBS as the primary resource manager by setting **RMCFG**[*base*] **TYPE**=*PBS* in the Moab configuration file (`moab.cfg`).

If a non-standard PBS installation/configuration is being used, additional Moab parameters may be required to enable the Moab/PBS interface as in the line **RMCFG**[*base*] **HOST**=*$PBSSERVERHOST* **PORT**=*$PBSSERVERPORT*. See the Resource Manager Overview for more information.

Moab's user interface port is set using the SCHEDCFG parameter and is used for user-scheduler communication. This port must be different from the PBS scheduler port used for resource manager-scheduler communication.

TORQUE/Moab Considerations

The default meaning of a node for TORQUE and Moab are not the same. By default, a node is a host in TORQUE. The node may have one or more execution slots (procs) allocated to it in the $TORQUE_HOME/server_priv/nodes file. However, the number of nodes recognized by TORQUE is equivalent to the number of node entries in the *$TORQUE_HOME*`/server_priv/nodes` file. A node specification from qsub such as `-l nodes=2:ppn=2` will direct TORQUE to allocate to execution slots on two separate nodes.

Moab is more liberal in its interpretations of a node. To Moab, the qsub request above would be interpreted to mean allocate four tasks with at least two tasks on a node. Where TORQUE would require two nodes for the request, Moab will place all four tasks on the name node (host) if four execution slots are available.

If a cluster has four nodes with eight processors each, TORQUE still sees only four nodes. Moab sees 32 nodes. However, if a user made a qsub request with `-l nodes=10`, TORQUE would reject the request because there are only four nodes available. To enable TORQUE to accommodate Moab's more liberal node interpretation, the server parameter available_resources.nodect can be set as a server parameter in TORQUE. The value of available_resources.nodect should equal at least the number of execution slots in the cluster.

For our example, cluster available_resources.nodect should be *32*. With this parameter set, the user can now make a request such as `-l nodes=8:ppn=2`. In this example, the user is still limited to a maximum node request of 32.

With available_resources.nodect set in TORQUE, Moab can be directed to honor the default TORQUE behavior by setting JOBNODEMATCHPOLICY to *EXACTNODE*.

PBS Features Not Supported by Moab

Moab supports basic scheduling of all PBS node specifications.

Moab Features Not Supported by PBS

PBS does not support the concept of a job QoS or other extended scheduling features by default. This can be handled using the techniques described in the PBS Resource Manager Extensions section. See the Resource Manager Extensions Overview for more information.

Troubleshooting

On TRU64 systems, the PBS `libpbs` library does not properly export a number of symbols required by Moab. This can be worked around by modifying the Moab `Makefile` to link the PBS `rm.o` object file directly into Moab.

TORQUE/PBS Integration Guide - RM Access Control

Server Configuration

Using the PBS qmgr command, add the Moab administrator as both a *manager* and *operator*.

```
> qmgr
Qmgr: set server managers += <MOABADMIN>@*.<YOURDOMAIN>
Qmgr: set server operators += <MOABADMIN>@*.<YOURDOMAIN>
Qmgr: quit
```

For example:

```
> qmgr
Qmgr: set server managers += staff@*.ucsd.edu
Qmgr: set operators += staff@*.ucsd.edu
Qmgr: quit
```

If desired, the Moab administrator can be enabled as a manager and operator only on the host on which Moab is running by replacing "*.<YOURDOMAIN>" with "<MOABSERVERHOSTNAME>".

Mom Configuration (optional)

If direct Moab to pbs_mom communication is required, the `mom_priv/config` file on each compute node where pbs_mom runs should be set as in the following example:

```
$restricted *.<YOURDOMAIN>
$clienthost <MOABSERVERHOSTNAME>
```

For security purposes, sites may want to run Moab under a non-root user id. If so, and Moab-pbs_mom communication is required, the `mom_priv/config` files must be world-readable and contain the line '`$restricted *.<YOURDOMAIN>`'. (i.e., '`$restricted *.uconn.edu`')

TORQUE/PBS Config - Default Queue Settings

Default Queue

To set the default queue (the queue used by jobs if a queue is not explicitly specified by the user), issue the following:

```
>> qmgr
Qmgr: set system default_queue = <QUEUENAME>
Qmgr: quit
```

Queue Default Node and Walltime Attributes

To set a default of one node and 15 minutes of walltime for a particular queue, issue the following:

```
> qmgr
Qmgr: set queue <QUEUENAME> resources_default.nodect = 1
Qmgr: set queue <QUEUENAME> resources_default.walltime = 00:15:00
Qmgr: quit
```

Default System Wide Node and Walltime Attributes

To set system wide defaults, set the following:

```
> qmgr
Qmgr: set server resources_default.nodect = 1
Qmgr: set server resources_default.walltime = 00:15:00
Qmgr: quit
```

Moab-SLURM Integration Guide

- Overview
- SLURM Configuration Steps
- Moab Configuration Steps
  - Configuration for Standby and Expedite
  - Configuration for the Quadrics Switch
  - Authentication
  - Queue/Class Support
  - Policies
  - Moab Queue and RM Emulation
  - SLURM High Availability

Overview

Moab can be used as the scheduler for the SLURM resource manager. In this configuration, the SLURM handles the job queue and the compute resources while Moab determines when, where and how jobs should be executed according to current cluster state and site mission objectives.

The documentation below describes how to configure Moab to interface with SLURM.

For Moab-SLURM integration, Moab 6.0 or higher and SLURM 2.2 or higher are recommended. From the downloads page, the generic version is needed to install SLURM.

SLURM Configuration Steps

To configure SLURM to utilize Moab as the scheduler, the SchedulerType parameters must be set in the `slurm.conf` config file located in the SLURM `etc` directory (`/usr/local/etc` by default)

```
# slurm.conf
SchedulerType=sched/wiki2
```

The SchedulerType parameter controls the communication protocol used between Moab and SLURM. This interface can be customized using the wiki.conf configuration file located in the same directory and further documented in the SLURM Admin Manual.

Note: To allow sharing of nodes, the SLURM partition should be configured with 'Shared=yes' attribute.

Moab Configuration Steps

By default, Moab is built with WIKI interface support (which is used to interface with SLURM) when running the standard configure and make process.

To configure Moab to use SLURM, the parameter 'RMCFG' should be set to use the **WIKI:SLURM** protocol as in the example below.

```
# moab.cfg

SCHEDCFG[base] MODE=NORMAL
RMCFG[base] TYPE=WIKI:SLURM
...
```

Note: The RMCFG index (set to *base* in the example above) can be any value chosen by the site. Also, if SLURM is running on a node other than the one on which Moab is running, then the **SERVER** attribute of the RMCFG parameter should be set.

Note: SLURM possesses a SchedulerPort parameter which is used to communicate with the scheduler. Moab will auto-detect this port and communicate with SLURM automatically with no explicit configuration required. Do NOT set Moab's SCHEDCFG[] **PORT** attribute to this value, this port controls Moab client communication and setting it to match the SchedulerPort value will cause conflicts. With no changes, the default configuration will work fine.

Note: If the SLURM client commands/executables are not available on the machine running Moab, SLURM partition and other certain configuration information will not be automatically imported from SLURM, thereby requiring a manual setup of this information in Moab. In addition, the SLURM VERSION should be set as an attribute on the RMCFG parameter. If it is not set, the default is version 1.2.0. The following example shows how to set this line if SLURM v1.1.24 is running on a host named Node01 (set using the SERVER attribute).

```
# moab.cfg with SLURM on Host Node01

RMCFG[base] TYPE=WIKI:SLURM SERVER=Node01 VERSION=10124
...
```

Configuration for Standby and Expedite Support

SLURM's 'Standby' and 'Expedite' options are mapped to the Moab QoS feature. By default, when a SLURM interface is detected, Moab will automatically create a 'standby' and an 'expedite' QoS. By default, the 'standby' QoS will be globally accessible to all users and on all nodes and will have a lower than normal priority. Also by default, the 'expedite' QoS will not be accessible by any user, will have no node constraints, and will have a higher than normal priority.

Authorizing Users to Use 'Expedite'

To allow users to request 'expedite' jobs, the user will need to be added to the 'expedite' QoS. This can be accomplished using the MEMBERULIST attribute as in the following example:

```
MEMBERULIST

# allow josh, steve, and user c1443 to submit 'expedite' jobs
QOSCFG[expedite] MEMBERULIST=josh,steve,c1443
...
```

Excluding Nodes for 'Expedite' and 'Standby' Usage

Both 'expedite' and 'standby' jobs can be independently excluded from certain nodes by creating a QoS-based standing reservation.

Specifically, this is accomplished by creating a reservation with a logical-*not* QoS ACL and a host list indicating which nodes are to be exempted as in the following example:

```
MEMBERULIST

# block expedite jobs from reserved nodes
SRCFG[expedite-blocker] QOSLIST=!expedite
SRCFG[expedite-blocker] HOSTLIST=c001[3-7],c200
SRCFG[expedite-blocker] PERIOD=INFINITY

# block standby jobs from rack 13
SRCFG[standby-blocker] QOSLIST=!standby
SRCFG[standby-blocker] HOSTLIST=R:r13-[0-13]
SRCFG[standby-blocker] PERIOD=INFINITY
...
```

Quadrics Integration

If managing a cluster with a Quadrics high speed network, significant performance improvement can be obtained by instructing Moab to allocate contiguous collections of nodes. This can be accomplished by setting the NODEALLOCATIONPOLICY parameter to *CONTIGUOUS* as in the example below:

```
# moab.cfg

SCHEDCFG[cluster1]   MODE=NORMAL SERVER=head.cluster1.org
RMCFG[slurm]         TYPE=wiki:slurm
NODEALLOCATIONPOLICY CONTIGUOUS
...
```

Setting Up Authentication

By default, Moab will not require server authentication. However, if SLURM's `wiki.conf` file (default location is `/usr/local/etc`) contains the AuthKey parameter or a secret key is specified via SLURM's configure using the `--with-key` option, Moab must be configured to honor this setting. Moab configuration is specified by setting the resource manager **AUTHTYPE** attribute to *CHECKSUM* and the **KEY** value in the moab-private.cfg file to the secret key as in the example below.

```
# /usr/local/etc/wiki.conf

AuthKey=4322953
```

```
...
```

```
# moab.cfg

RMCFG[slurm]          TYPE=wiki:slurm AUTHTYPE=CHECKSUM
...
```

```
# moab-private.cfg

CLIENTCFG[RM:slurm]  KEY=4322953
...
```

Note: For the *CHECKSUM* authorization method, the key value specified in the `moab-private.cfg` file must be a decimal, octal, or hexadecimal value, it cannot be an arbitrary non-numeric string.

Queue/Class Support

While SLURM supports the concept of classes and queues, Moab provides a flexible alternative queue interface system. In most cases, sites can create and manage queues by defining partitions within SLURM. Internally, these SLURM partitions are mapped to Moab classes which can then be managed and configured using Moab's CLASSCFG parameter and mdiag -c command.

Policies

By default, SLURM systems only allow tasks from a single job to utilize the resources of a compute node. Consequently, when a SLURM interface is detected, Moab will automatically set the NODEACCESSPOLICY parameter to *SINGLEJOB*. To allow node sharing, the SLURM partition attribute '**Shared**' should be set to *FORCE* in the `slurm.conf` as in the example below:

```
# slurm.conf

PartitionName=batch Nodes=node[1-64] Default=YES MaxTime=INFINITE State=UP
Shared=FORCE
```

Moab Queue and RM Emulation

With a SLURM system, jobs can be submitted either to SLURM or to Moab. If submitted to SLURM, the standard SLURM job submission language must be used. If jobs are submitted to Moab using the msub command, then either LSF*, PBS, or Loadleveler* job submission syntax can be used. These jobs will be translated by Moab and migrated to SLURM using its native job language.

SLURM High Availability

If SLURM high availability mode is enabled, Moab will automatically detect the presence of the SLURM BackupController and utilize it if the primary fails. To verify SLURM is properly configured, issue the SLURM command 'scontrol show config | grep Backup'. To verify Moab properly detects this information, run 'mdiag -R -v | grep FallBack'.

Note: To use SLURM high availability, the SLURM parameter StateSaveLocation must point to a shared directory which is readable and writable by both the primary and backup hosts. See the `slurm.conf` man page for additional information.

Related topics

- SLURM Admin Manual
- SLURM's Moab Integration Guide
- Additional SLURM Documentation
- Wiki Overview

Installation Notes for Moab and TORQUE for Cray

Copyright © 2012 Adaptive Computing Enterprises, Inc.

This document provides information on the steps to install Moab 7.2.0 and TORQUE 4.1.0 on a Cray XT system.

Overview

Moab and TORQUE can be used to manage the batch system for Cray. This document describes how to configure Moab and TORQUE to bring Moab's unmatched scheduling capabilities to the Cray.

New to TORQUE 4.1, TORQUE now handles all communication with ALPS, specifically the pbs_mom. Previously, communication with ALPS was handled by a combination of Moab, scripts and TORQUE. In the new model, Moab treats TORQUE as a regular TORQUE cluster without any special configuration. TORQUE now uses an extra MOM called the alps_reporter MOM to communicate with ALPS regarding configured and available resources. From the information reported by the alps_reporter mom, TORQUE creates a virtual node for each Cray compute node. Previously, TORQUE only reported the login nodes.

Note: For clarity this document assumes that your SDB node is mounting a persistent `/var` file system from the bootnode. If you have chosen not to use persistent `/var` file systems please be aware that the instructions below would have to be modified for your situation.

Upgrade Notes

When upgrading to TORQUE 4.1.0 and using the new Cray model as described in this document, there should be no running jobs. Jobs may be queued but not running.

Torque Installation Notes

Perform the following steps from the *boot node* as root:

> Many of the following examples reflect a specific setup and must be modified to fit your unique configuration.

Download the latest TORQUE release.

Download the latest TORQUE release.

Example 1. Download TORQUE

```
# cd /rr/current/software
# wget http://www.adaptivecomputing.com/resources/downloads/torque/torque-4.1.0.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-4.1.0.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 host name where the TORQUE server will run (`--with-default-server`) if it is different from the host it is being compiled on. The TORQUE server has typically been on the SDB node of your Cray system.

Example 3. Run configure

```
default/:/software # cd torque-4.1.0
default/:/software/torque-4.1.0 # ./configure --prefix=/opt/torque/4.1.0 --with-
server-home=/var/spool/torque --with-default-server=sdb --enable-syslog --disable-gcc-
warnings --with-debug --with-modulefiles=/opt/modulefiles --with-job-create CFLAGS="-
DCRAY_MOAB_PASSTHRU"
```

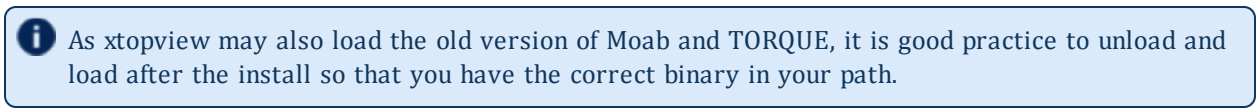
Note: The `--with-job-create` is a change for TORQUE 2.5.9 onwards. This is not necessary on 2.4.16. Sites running TORQUE 2.5.x should upgrade to 2.5.9 or later.

Note: The `-DCRAY_MOAB_PASSTHRU` option tells TORQUE to not validate the qsub -l nodes syntax. For more information, see Submitting Jobs.

Compile and Install TORQUE

You must unload the current module:

```
# module unload moab torque
```



While still in xtopview, compile and install TORQUE into the shared root. Create a link to the installed TORQUE. Exit xtopview.

Example 4. Make and Make Install

```
default/:/software/torque-4.1.0 # make
default/:/software/torque-4.1.0 # make packages
default/:/software/torque-4.1.0 # make install
default/:/software/torque-4.1.0 # ln -sf /opt/torque/4.1.0/ /opt/torque/default
default/:/software/torque-4.1.0 # exit
```

After installing, run `module list` to see what versions you have. If the versions are incorrect, unload and load to confirm you are using the correct versions.

Copy your TORQUE server directory to your moab server host

Example 5. On the boot node, copy the TORQUE home directory to the SDB node's persistent `/var` file system (as exported from the bootnode). This example assumes that the SDB is NID 3 and that you are installing it on the SDB. These instructions need to be modified if the Moab and TORQUE servers are being installed on a different node.

```
# cd /rr/current/var/spool
# cp -pr torque /snv/3/var/spool
```

Set up pbs_server to be Cray compatible

1. Customize the nodes file located in *<TORQUE HOME>*/server_priv/nodes.

   a. Identify the reporter MOM with the reserved feature alps_reporter.

   ```
   sdb alps_reporter
   ```

   We recommend that you set up the SDB node as the ALPS reporter. Setting the NP for this node isn't important because this node will not appear in pbsnodes output and, therefore, will not be scheduled to run jobs.

   b. Identify all login nodes using the reserved feature alps_login.

   ```
   login1 alps_login np=X <other features>
   login2 alps_login np=Y <other features>
   login3 alps_login np=Z <other features>
   ...
   ```

   Identifying these moms as login nodes allows pbs_server to verify that each job has a login node as its mother superior. It also tells pbs_server to place size=0 jobs on one of these login nodes.

2. Set up access and submit permissions from your login nodes.

   ```
   $ qmgr -c 'set server acl_host_enable=true'
   $ qmgr -c 'set server acl_hosts+=login1'
   $ qmgr -c 'set server acl_hosts+=login2'
   $ qmgr -c 'set server acl_hosts+=login3'
   $ qmgr -c 'set server submit_hosts+=login1'
   $ qmgr -c 'set server submit_hosts+=login2'
   $ qmgr -c 'set server submit_hosts+=login3'
   ```

3. Set some helpful server parameters.

   ```
   $ qmgr -c 'set server scheduling = true'
   ```

   This parameter tells pbs_server to notify Moab when pertinent events have happened. If this isn't set, Moab will automatically set it on startup.

   ```
   $ qmgr -c 'set server keep_completed = 300'
   ```

   This tells TORQUE to keep information about completed jobs for 300 seconds (5 minutes) after they have completed. You can customize this number to meet your site's needs.

4. Set the server attribute `cray_enabled` to `True`.

   ```
   $ qmgr -c 'set server cray_enabled = true'
   ```

After using qmgr to set this variable, you will need to restart so that when pbs_server parses the nodes file, it will know it is Cray-enabled.

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 6. Copy in init.d script

```
# xtopview -n <sdb nid>
node/<sdb nid>:/ # cp /software/torque-4.1.0/contrib/init.d/suse.pbs_server
/etc/init.d
node/<sdb nid>:/ # chmod +x /etc/init.d/pbs_server
node/<sdb nid>:/ # chkconfig --add pbs_server
```

Edit the init.d file as necessary -- i.e., change PBS_DAEMON and PBS_HOME as appropriate.

```
# vi /etc/init.d/pbs_server
PBS_DAEMON=/opt/torque/default/sbin/pbs_server PBS_HOME=/var/spool/torque
```

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 7. Copy in init.d script

```
# xtopview
default/:/ # cp /software/torque-4.1.0/contrib/init.d/suse.pbs_mom /etc/init.d
default/:/ # chmod +x /etc/init.d/pbs_mom
default/:/ # chkconfig --add pbs_mom
```

Edit the init.d file as necessary -- i.e. change PBS_DAEMON and PBS_HOME as appropriate, retain core files, etc.

```
# vi /etc/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:

`ulimit -c unlimited` # Uncomment this to preserve core files

Install the trqauthd init.d script on all TORQUE nodes and the SDB (Optional)

Torque provides an init.d script for starting trqauthd as a service.

Example 8. Copy in init.d script

```
# xtopview
default/:/ # cp /software/torque-4.1.0/contrib/init.d/suse.trqauthd /etc/init.d
default/:/ # chmod +x /etc/init.d/trqauthd
default/:/ # chkconfig --add trqauthd
```

Edit the init.d file as necessary -- i.e. change PBS_DAEMON and PBS_HOME as appropriate.

```
# vi /etc/init.d/trqauthd
PBS_DAEMON=/opt/torque/default/sbin/trqauthd PBS_HOME=/var/spool/torque
```

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` file systems mounted from `/snv` on the boot node. Alternatively, a ram var file system 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 host name of the SDB node in the server_name file.

Example 9. Copy out MOM dirs and client server info

```
# cd /rr/current/software/torque-4.1.0/tpackages/mom/var/spool
# for i in 4 64 68: \
     do cp -pr torque /snv/$i/var/spool; \
     echo sdb > /snv/$i/var/spool/torque/server_name; \
     done
```

Note: It is possible that the host name for the SDB node is not set to SDB on your system. Run ssh sdb hostname to determine the host name in use. If the command returns, for example, sdb-p1, modify the "for loop" above to `echo sdb-p1` into the server_name file.

Update the TORQUE MOM config file for the ALPS reporter mom

In the above steps, we identified the ALPS reporter MOM on the pbs_server. We now need to configure the MOM to be the ALPS reporter mom. The ALPS reporter MOM is installed on the SDB. To configure the ALPS reporter mom, set the following in the pbs_mom config file on the SDB:

```
# vi var/spool/torque/mom_priv/config
$reporter_mom true # defaults to false
```

You may also wish to set these variables:

```
$apbasil_path <path_to_apbasil> # defaults to /usr/bin/apbasil if not set
$apbasil_protocol <protocol> # defaults to 1.0 if not set
```

As of CLE 5.0, apbasil is in the `/opt/cray/alps/default/bin/apbasil` directory, not `/usr/bin/apbasil`. Supported apbasil protocols are 1.0, 1.1, and 1.2.

Cray systems do not support GPUs until ALPS version 1.2. Setting `$apbasil_protocol 1.2` in `mom_priv/config` causes the GPU status to appear in the `pbsnodes` output.

Update the TORQUE MOM config file on each login node

Login nodes are service nodes running pbs_moms which are used for submission and launching of job scripts. Login nodes are responsible for creating and confirming ALPS reservations so that the script launched on a login node can access the compute nodes with the aprun command.

Edit the MOM config file so job output is copied to locally mounted directories.

Example 10. Edit the MOM config file

```
# vi var/spool/torque/mom_priv/config

$usecp *:/home/users /home/users
```

```
$usecp *:/scratch /scratch

$login_node true
```

$login_node specifies that this node will create and confirm ALPS reservations.

Note: It may be acceptable to use a `$usecp *:/ /` in place of the sample above. Consult with the site.

You may also wish to set these variables:

```
$apbasil_path <path_to_apbasil> # defaults to /usr/bin/apbasil if not set
$apbasil_protocol <protocol> # defaults to 1.0 if not set
```

As of CLE 5.0, apbasil is in the `/opt/cray/alps/default/bin/apbasil` directory, not `/usr/bin/apbasil`. Supported apbasil protocols are 1.0, 1.1, and 1.2.

Start up the TORQUE MOM Daemons

On the boot node as root:
Example 11. Start up the pbs_moms on the login nodes.

```
# pdsh -w sdb,login[1-3] /opt/torque/default/sbin/pbs_mom
# pdsh -w login[1-3] trqauthd
```

Alternatively, if you installed the `init.d` script, you may run:

```
# pdsh -w sdb,login[1-3] /sbin/service pbs_mom start
# pdsh -w login[1-3] service trqauthd start
```

Startup the TORQUE Server

On the TORQUE server host as root:
Example 12. Start pbs_server

```
# /opt/torque/default/sbin/pbs_server
# /opt/torque/default/sbin/trqauthd
```

Alternatively, if you installed the `init.d` script, you may run:

```
# service pbs_server start
# service trqauthd start
```

Moab Installation Notes

Perform the following steps from the boot node as root:

Download the latest Moab release

Download the latest Moab release from Adaptive Computing Enterprises, Inc.

The correct tarball to install is the plain Moab & TORQUE builds. The XT4 builds are for releases prior to TORQUE 4.1.0 and MWM 7.2.0.

Example 13. Download Moab to the boot node

```
# cd /rr/current/software
wget --post-
data="username=<username>&password=<password>&submit=submit&url=/download/mwm/moab-
7.2.0-linux-x86_64-torque.tar.gz"
https://www.adaptivecomputing.com/myaccount/login.php;
```

Unpack the Moab tarball

Using xtopview, unpack the Moab tarball into the software directory in the shared root.

Example 14. Unpack Moab

```
# xtopview
default/:/ # cd /software
default/:/software # tar -zxvf moab-7.2.0-linux-x86_64-torque.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/$version` 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 15. Run configure

```
default/:/software # cd moab-7.2.0
default/:/software/moab-7.2.0 # ./configure --prefix=/opt/moab/7.0.1 --with-
homedir=/var/spool/moab --with-torque=/opt/torque/default --with-
modulefiles=/opt/modulefiles
```

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 16. Make Install

```
default/:/software/moab-7.2.0 # make install
default/:/software/moab-7.2.0 # ln -sf /opt/moab/7.0.1/ /opt/moab/default
```

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 17. Edit the Moab configuration file

```
# cd /rr/current/var/spool/moab
# vi moab.cfg
SCHEDCFG[moab]       SERVER=sdb:42559
RMCFG[<clustername>] TYPE=TORQUE
NODEACCESSPOLICY     SINGLEJOB
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT]     PRIORITYF=-NODEINDEX

JOBMAXTASKCOUNT      <total number of processors>
MAXNODE              <total number of nodes>
```

By default, ALPS reports the compute nodes in a serialized topology order. TORQUE preserves this ordering by reporting a node_index on each compute node that represents the compute nodes' placement in the ALPS topology ordering. This information is then used by Moab to allocate nodes close to each other in the network. The downside to this is that the nodes can be become fragmented. The NODEALLOCATIONPOLICYPRIORITY parameter used with NODECFG[*DEFAULT*] PRIORITYF=*-NODEINDEX* tells Moab to allocate nodes based on the nodes' node_index reported by TORQUE beginning with the first nodes in the list (-1 x node_index of 1).

It is also possible to use the same node indexes reported by TORQUE to allocate strict contiguous sets of nodes. This is configured by specifying a NODEALLOCATIONPOLICY of *CONTIGUOUS*. In this mode, a job won't run until it can get a strict set of contiguous nodes.

When Moab allocates nodes on the Cray it must only get compute nodes. The purpose of the login nodes are to create and confirm ALPS reservations so that the job script can access the allocated compute nodes. Moab shifts the responsibility of selecting a login node for the allocated compute nodes to TORQUE. Because Moab doesn't allocate a login node with compute nodes, the login nodes must be kept separate from other compute nodes so that Moab doesn't allocate login nodes and compute nodes for the same job. This is accomplished by putting the login nodes in a separate partition. Moab does not allocate jobs across partitions. By default, Moab creates a partition for each RMCFG with the name given in the RMCFG parameter and sticks all the nodes reported by that resource manager in that partition.

With the login and compute nodes now separated, configure all jobs to request the compute partition by default.

Place each login node in a separate partition called login. For example:

```
NODECFG[login1] Partition=login
NODECFG[login2] Partition=login
NODECFG[login3] Partition=login
NODECFG[login4] Partition=login
```

Configure all jobs submitted through msub to request the compute node partition by default.

```
CLIENTCFG[DEFAULT] DEFAULTSUBMITPARTITION=<clustername>
```

Configure all jobs submitted through qsub to request the compute node partition by default.

```
qmgr -c "set server resources_default.partition=<clustername>"
```

Login nodes can be requested to run jobs that don't require Cray compute nodes (for example, compute jobs or data transfer jobs). These jobs can be submitted to the login partition (for example, `qsub -l partition=login`).

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 file system 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 18. Copy out Moab home directory. This example assumes that the SDB is NID 3.

```
# cd /rr/current/var/spool
# cp -pr moab /snv/3/var/spool
```

Copy the Moab configuration file to all of the login nodes

The Moab configuration file (`moab.cfg`) must be copied out to the `/var` file system on the login nodes. The essential parameters that must be in the `moab.cfg` on the login nodes are the **SCHEDCFG** line so the clients can find the server and any client-specific parameters, such as the **CLIENTCFG** parameter.
Example 19. Copy out the configuration files.

```
# cd /rr/current/var/spool/moab
# for i in 4 64 68; do mkdir -p /snv/$i/var/spool/moab/etc /snv/$i/var/spool/moab/log;
cp moab.cfg /snv/$i/var/spool/moab; done
```

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 20. Copy in init.d script to the SDB node from the shared root.

```
# xtopview -n <sdb nid>
node/<sdb nid>:/ # cp /software/moab/moab-7.2/contrib/init.d/moab_sles_init
/etc/init.d/moab
node/<sdb nid>:/ # chkconfig --add /etc/init.d/moab
```

Edit the `init.d` file as necessary -- i.e. retain core files, etc.

Uncomment the following line to retain core dump files

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 21. Loading the Moab module

```
# module load moab
```

Example 22. Exporting the environment variables by hand (in bash)
Example 23. 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
```

Startup the Moab Workload Manager

Start up the Moab daemon.
Example 24. Start Moab

```
# /opt/moab/default/sbin/moab
```

Alternatively, if you installed the `init.d` script, you may run:

```
# service moab start
```

Running Moab and PBS Server outside the Cray network

Previously, Moab and TORQUE had to be run inside the Cray network. With the new model, it is now possible to run Moab and pbs_server outside of the Cray. This provides benefits of having Moab and pbs_server on bigger hardware other than that provided in the service nodes and enables the use of Moab and TORQUE's high-availability features. Also, jobs can be submitted and queued up if the Cray is down. In order to set up Moab and TORQUE to run on an external node, pbs_server must be able to communicate with the login nodes inside the Cray on ports 15002 and 15003 and the login nodes must be able to communicate with the pbs_server on ports 15001.

```
# [root@ext-server /]# telnet login1 15002
Trying XXX.XXX.XXX.XXX...
Connected to login1
Escape character is '^]'.

# [root@ext-server /]# telnet login1 15003
Trying XXX.XXX.XXX.XXX...
Connected to login1
Escape character is '^]'.

# [root@login1 /]# telnet ext-server 15001
Trying XXX.XXX.XXX.XXX...
Connected to ext-server
Escape character is '^]'.
```

System Reservations

System reservations can be done several ways. 1) Just compute nodes can be reserved leaving the login nodes available for executing non-compute jobs.

```
SRCFG[PM] TASKCOUNT=7832
SRCFG[PM] HOSTLIST=!login
SRCFG[PM] PERIOD=DAY DAYS=TUE
SRCFG[PM] FLAGS=OWNERPREEMPT
SRCFG[PM] STARTTIME=8:00:00 ENDTIME=14:00:00
SRCFG[PM] JOBATTRLIST=PREEMPTEE
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"
```

2) Just the login nodes can be reserved leaving just the compute nodes available for execution and

```
SRCFG[PM] TASKCOUNT=192
SRCFG[PM] HOSTLIST=login
SRCFG[PM] PERIOD=DAY DAYS=TUE
SRCFG[PM] FLAGS=OWNERPREEMPT
SRCFG[PM] STARTTIME=8:00:00 ENDTIME=14:00:00
SRCFG[PM] JOBATTRLIST=PREEMPTEE
```

```
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"
```

3) Reserving the whole system, preventing any kind of job from starting.

```
SRCFG[PM] HOSTLIST=ALL
SRCFG[PM] PERIOD=DAY DAYS=TUE
SRCFG[PM] FLAGS=OWNERPREEMPT
SRCFG[PM] STARTTIME=8:00:00 ENDTIME=14:00:00
SRCFG[PM] JOBATTRLIST=PREEMPTEE
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"
```

Setting up the Cray for Use with External Nodes

This feature works with TORQUE 4.1.2 or later.

Set up the Cray as described previously. In Moab, do not place the login nodes in a separate partition. Add the external nodes to the nodes file (*<TORQUEHOME>*/server_priv/nodes) with a feature named external.

```
<hostname>  np=X external
```

In pbsnodes or mdiag -n -v output the login nodes have the feature alps_login and the Cray compute nodes have the cray_compute feature. These are automatically added by pbs_server.

To request a heterogeneous job:

```
> qsub job_script.sh -l nodes=X:cray_compute+Y:external
```

In the example above, Moab assigns X number of Cray compute nodes and Y number of external nodes and passes the information appropriately to pbs_server.

To request a cray only job:

```
> qsub job.script.sh -l nodes=X:cray_compute
```

To request an external only job:

```
> qsub job_script.sh -l nodes=X:external
```

To request a login-only job (such as a job that compiles the code to be run on Cray compute nodes):

```
> qsub job_script.sh -l nodes=X:alps_login
```

In this setup, all job requests must request features for all jobs. If this doesn't happen, Moab might try to schedule the login nodes for running a job or other issues might occur. This should be enforced by setting defaults or by using a submit filter (see the **Applying the msub Submit Filter** on page 218 section for more information)

Writing job scripts for heterogeneous (Cray and external node) jobs

Once configured as above, the same job script is launched on the external nodes and on the Cray. In order for this job to function properly, the job script must detect whether or not it is on the Cray, and then execute the appropriate commands. We recommend that the script inspect the contents of *$PBS_NODEFILE*. This file, whose path is contained in the variable *$PBS_NODEFILE*, contains a list of the host names on which the job is executing, with one host name per line. The first line is the host name of the node on which the script is executing. The script should simply read this line, decide if that is a node external to or inside the Cray, and then execute appropriately.

Submitting Jobs

There are three different ways to submit jobs to the Cray. Each way works in its own way and shouldn't be mixed with other methods.

- `-l nodes=`
- `-l size=`
- `-l mpp*=`

`-l nodes=` is the standard way of submitting jobs to Moab. It is the recommended way since this is the most supported and standard way of submitting jobs among all types of systems run by Moab. One benefit of the `-l nodes=` syntax is that you can submit multi-req jobs (ex., `-l nodes=2:ppn=3+3:ppn=2`). When using the `-l nodes=` syntax, TORQUE should be compiled with the `-DCRAY_MOAB_PASSTHRU` option. By default, `-l nodes=` requests the number of processors, not nodes. If you want `-l nodes=` to request nodes, add JOBNODEMATCHPOLICY *EXACTNODE* to your `moab.cfg`.

`-l size=` was created to be a very simple interface for submitting to the Cray. It requests the number of one-proc tasks to submit on the Cray. Customers that use this option usually have a submit filter that verifies that the number of tasks requested is a multiple of the number of processors per node and rejects the submission if it isn't.

`-l mmp*=` is standard among Cray systems, which is known among Moab/TORQUE and PBS-run systems. Most of the mpp options have an equivalent `-l nodes=` option.

mppwidth	
Format	*<INTEGER>*
Default	---
Description	Number of tasks.
Example	`qsub -l mppwidth=48` *Requests 48 tasks of 1 processor each.*

mppwidth	
Equivalent	`qsub -l nodes=4`

mppnppn	
Format	*<INTEGER>*
Default	---
Description	Number of tasks per node.
Example	`qsub -l mppwidth=48,mppnppn=8` *Requests 48 tasks with 8 tasks per node.*
Equivalent	`qsub -l nodes=48:ppn=8`

mppdepth	
Format	*<INTEGER>*
Default	---
Description	Number of processors per task.
Example	`qsub -l mppwidth=24,mppdepth=2,mppnppn=8` *Requests 24 tasks with 2 processors per task with 8 tasks per node.*

mpphost	
Format	*<STRING>[:<STRING>]...*
Default	---

mpphost	
Description	Partition access list.
Example	`qsub -l mpphost=cray` *Specifies that the job must run on the cray partition.*
Equivalent	`qsub -l partition=cray`

mpparch	
Format	*<STRING>*
Default	---
Description	Required architecture.
Example	`qsub -l mpparch=xt5` *Specifies that the job must run on the xt5 architecture nodes.*
Equivalent	`qsub -l arch=xt5`

mppmem	
Format	*<INTEGER>*[kb|mb|gb]
Default	---
Description	Dedicated memory per task in bytes.
Example	`qsub -l mppmem=200mb` *Specifies that the job requires 200mb per task.*

mppmem	
Equivalent	`qsub -l mem=200mb`

mpplabels	
Format	*<FEATURE>[:<FEATURE>]...*
Default	---
Description	Required list of node features.
Example	`qsub -l mpplabels=featureA:featureB` *Specifies that the job should run on nodes that have both the featureA and featureB features.*
Equivalent	`qsub -l feature=featureA:featureB`

mppnodes	
Format	'+' delimited list of host names
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. A subset means the specified host list is used first to select hosts for the job. If the job requires more hosts than are in the host list, they will be obtained from elsewhere if possible. If the job does not require all of the jobs in the host list, it will use only the ones it needs. A superset means the host list is the *only* source of hosts that should be considered for running the job. If the job can't find the necessary resources in the hosts in this list it should *not* run. No other hosts should be considered in allocating the job.
Example	`qsub -l mppnodes=*512+513` *Specifies that the job should use nodes 512 and 513.*

mppnodes	
Equivalent	`qsub -l hostlist=*512+513`

The mppnppn and mppwidth parameters work with CLASSCFG **MIN.NODE** and **MAX.NODE** to filter classes based on how many nodes the job will use. The other parameters have not been tested.

Launching jobs from designated login nodes (Optional)

It is possible to direct a job to launch from a specific login node. This is done by assigning node features to specific login nodes and requesting these features at submission time with the `-W login_property` option. The **login_property** has no influence on which compute nodes are allocated to the job.

Example 25. Declaring MOM features

For example, if login2 had the himem feature, a job could request that its job launch from login2 rather than login1.

```
# vi /var/spool/torque/server-priv/nodes
login1 alps_login np=200
login2 alps_login np=200 himem
```

```
qsub -W login_property=himem
```

Setting up msub

Point Moab to the qsub binary on the server where Moab is running (ex. sdb).

```
RMCFG[] SUBMITCMD=/opt/torque/default/bin/qsub
```

Setup Moab to schedule nodes when `-l nodes` is requested.

```
JOBNODEMATCHPOLICY EXACTNODE
```

Because Moab uses qsub to submit msub'd jobs, qsub must be configured to not validate the path of the working directory on the sdb as they don't exist on the sdb. (ex. `msub -d /users/jdoe/tmpdir`). Add VALIDATEPATH*FALSE* to the `torque.cfg` on the sdb.

As of TORQUE 2.4.11, the node count and the processors per node count can be obtained in the job's environment by using `$PBS_NUM_NODES` and `$PBS_NUM_PPN` respectively. This aids in mapping the requested nodes to aprun calls. For example, the general format for calling aprun within a job script is: `aprun -n $(($PBS_NUM_NODES * $PBS_NUM_PPN)) -N $PBS_NUM_PPN`

Example submissions:

```
#PBS -l nodes=1:ppn=16
aprun -n 16 -N 16 hostname
```

```
#PBS -l nodes=20
aprun -n 20 hostname
```

```
#PBS -l nodes=20:ppn=16
aprun -n 320 -N 16 hostname
```

```
#PBS -l nodes=2:ppn=16
#PBS -l hostlist=35+36
aprun -n 32 -N 16 hostname
```

```
#PBS -l procs=64
aprun -n 64 hostname
```

```
#run on login nodes only
#PBS -l procs=0
```

Interactive Jobs

By default, interactive jobs run from the login node that they are submitted from, which can be useful if you need that node's particular features. This default behavior can be changed using the TORQUE server parameter interactive_jobs_can_roam. When set to *TRUE*, this parameter allows interactive jobs to run on login nodes other than the one where the jobs were submitted from. For an interactive job submitted from a different node to run on it, a node must have the alps_login property set in the `nodes` file.

```
qmgr -c 'set server interactive_jobs_can_roam = True'
```

You can disable the behavior of `interactive_jobs_can_roam = True` for an individual interactive job by submitting it with the option `-W login_property=<nodeId>`. The job will run on node *<nodeId>* and will not roam.

Job Information

In TORQUE 4.1, Cray compute nodes are treated as TORQUE nodes. This also applies to job allocations. For example, a job can know which nodes were allocated to the job by cat'ing $PBS_NODEFILE.

```
$ qsub -l -l size=64,walltime=5:00
qsub: waiting for job 3043.cray to start
qsub: job 3043.cray ready

$ cat $PBS_NODEFILE | wc -l
64
$ cat $PBS_NODEFILE | uniq
2876
2877
```

You can also view where your job is running by looking at `exec_host` in qsat -f output.

```
$ qstat -f 3043 | grep -A 7 exec_host
 exec_host = 2876/31+2876/30+2876/29+2876/28+2876/27+2876/26+2876/25+2876/2
4+2876/23+2876/22+2876/21+2876/20+2876/19+2876/18+2876/17+2876/16+2876
/15+2876/14+2876/13+2876/12+2876/11+2876/10+2876/9+2876/8+2876/7+2876/
6+2876/5+2876/4+2876/3+2876/2+2876/1+2876/0+2877/31+2877/30+2877/29+28
77/28+2877/27+2877/26+2877/25+2877/24+2877/23+2877/22+2877/21+2877/20+
2877/19+2877/18+2877/17+2877/16+2877/15+2877/14+2877/13+2877/12+2877/1
1+2877/10+2877/9+2877/8+2877/7+2877/6+2877/5+2877/4+2877/3+2877/2+2877
/1+2877/0
```

The login node is not included in $PBS_NODEFILE or exec_host. It can be viewed through qstat -f on the login_node_id field.

```
$ qstat -f 3043 | grep login_node_id
 login_node_id = login6/0
```

pbsnodes will report the job running on compute nodes under jobs = field for compute nodes.

```
$ pbsnodes 2878
 state = job-exclusive,busy
 np = 32
 ntype = cluster
 jobs = 0/3043.cray, 1/3043.cray, 2/3043.cray, 3/3043.cray, 4/3043.cray, 5/3043.cray,
6/3043.cray, 7/3043.cray, 8/3043.cray, 9/3043.cray,
10/3043.cray, 11/3043.cray, 12/3043.cray, 13/3043.cray, 14/3043.cray, 15/3043.cray,
16/3043.cray, 17/3043.cray, 18/3043.cray, 19/3043.cray,
20/3043.cray, 21/3043.cray, 22/3043.cray, 23/3043.cray, 24/3043.cray, 25/3043.cray,
26/3043.cray, 27/3043.cray, 28/3043.cray, 29/3043.cray,
30/3043.cray, 31/3043.cray
 status = rectime=1340836785,node_
index=2814,state=BUSY,totmem=33554432kb,CMEMORY=32768,APROC=0,CPROC=32,name=c1-
0c2s0n2,ARCH=XT
 mom_service_port = 15002
 mom_manager_port = 15003
 gpus = 0
```

pbsnodes will not present Cray compute nodes on the login node's "jobs =" field but will for login only jobs. Cray compute jobs can be seen in the "jobs=" attribute in the "status=" field.

```
$ pbsnodes login6
 state = free
 np =24
 properties = fa,fb
 ntype = cluster
 jobs =
 status =
rectime=1340836830,varattr=,jobs=3043.cray,state=free,netload=3044161434,gres=,loadave
=0.04,ncpus=6,physmem=

7934560kb,availmem=12820180kb,totmem=15934044kb,idletime=1754,nusers=3,nsessions=29,se
ssions=1506 1923 1179 2127 2163 2176
2187 2195 2262 2275 2308 5809 8123 8277 8675 9547 10515 10551 12769 32351 14430 14518
22380 24082 24321 24849 30918 32371
32718,uname=Linux cray 2.6.35-32-generic #67-Ubuntu SMP Mon Mar 5 19:39:49 UTC 2012
x86_64,opsys=linux
 mom_service_port = 34001
 mom_manager_port = 34002
 gpus = 0
```

Provisioning Resource Managers

- **Validating an xCAT Installation for Use with Moab** on page 1006

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.

```
[root@h0 moab]# nodels h1,kvmm1 nodehm.mgt nodehm.power
h1: nodehm.power: ilo
h1: nodehm.mgt: ilo
kvmm1: nodehm.power: kvm
kvmm1: nodehm.mgt: kvm
[root@h0 moab]# rpower h1,kvmm1 off
h1: off
kvmm1: off
[root@h0 moab]# rpower h1,kvmm1 stat
h1: off
kvmm1: off
[root@h0 moab]# rpower h1,kvmm1 boot
h1: on reset
kvmm1: on reset
[root@h0 moab]# rpower h1,kvmm1 stat
h1: on
kvmm1: on
[root@h0 moab]#
```

Verifying Provisioning Images

Verify that all operating system images that Moab uses are configured correctly in xCAT. For stateful images, test that all combinations of operating system, architecture, and profile install correctly.

```
[root@h0 moab]# rinstall -o centos5.3 -a x86_64 -p hyper h1
h1: install centos3.2-x86_64-hyper
h1: on reset
[root@n100 ~]# sleep 15 && nodestat n05
n05: ping install centos5.3-x86_64-hyper
[root@h0 moab]#
```

For stateless images, test that nodes are able to network boot the images.

```
[root@h0 moab]# nodech h5 nodetype.os=centos5.3 nodetype.arch=x86_64
nodetype.profile=hyper
[root@h0 moab]# nodeset h5 netboot
h5: netboot centos5.3-x86_64-hyper
[root@h0 moab]# rpower h5 boot
h5: on reset
[root@h0 moab]# sleep 60 && nodestat h5
h5: pbs, sshd
[root@h0 moab]#
```

Verifying VM Migration

If you use VM migration, verify that xCAT can successfully perform migrations using the rmigrate command.

```
[root@h0 moab]# rmigrate kvmm7 h1
kvmm7: migrated to h1
[root@h0 moab]# ssh h1 virsh list
Id Name State
----------------------------------
33 kvmm1 running
34 kvmm2 running
35 kvmm7 running
```

Related topics

- Native Resource Manager Overview
- Resource Provisioning

Hardware Integration

- **Moab-NUMA Integration Guide** on page 1008

Moab-NUMA Integration Guide

Scheduling a shared-memory NUMA type system (not the same as a modern SMP-based individual compute node, which cannot share memory between compute nodes) requires some special configuration. Additionally, Moab can use NODESETs to guarantee feasibility of large memory jobs and to enforce node allocation based on the system's interconnect network topology.

Configuration

To integrate Moab and NUMA

1. Configure Moab to schedule large memory jobs. Because Moab creates a partition for each resource manager by default, you must configure the cluster controlled by the resource manager to be a shared-memory system to support jobs spanning multiple nodes/blades. To do so, use the PARCFG parameter.

```
RMCFG[sys-uv]  TYPE=TORQUE
PARCFG[sys-uv] FLAGS=SharedMem
```

Cluster `sys-uv` is now configured as a shared-memory system to Moab.

2. Configure NODESETs as shown below.

```
NODESETISOPTIONAL FALSE
NODESETATTRIBUTE FEATURE
NODESETPOLICY ONEOF
NODESETPRIORITYTYPE FIRSTFIT
```

The NODESET parameters tell Moab that performing node allocation using node sets is required, that the node set name is a feature name assigned to compute nodes, that a job must fit within the available nodes of one node set, and that Moab should use the first node set that contains sufficient available nodes to satisfy the job's request.

3. To configure Moab to perform topology-aware node allocation using node sets, you must create a node set definition for each set of nodes that has the same number of maximum network "hops" from any node to every other node within the node set. For an example, see the following sample scenario:

   Use case

   The SGI UV 1000 has a two-socket blade with a physical organization of 16 blades within a blade chassis (SGI term is Intra-Rack Unit or IRU), two blade chassis (IRUs) within a rack, and up to four racks within a single UV system. The UV 1000 interconnect network has a topology that requires zero hops between the two sockets on the same physical blade, one hop between an even-odd blade pair (e.g. blades 0 and 1, 2 and 3, etc.), two hops between all even-numbered or all odd-numbered blades within an IRU, three hops maximum between all blades within an IRU, four hops maximum between all even-numbered blades or all odd-numbered blades within a UV system, and five hops maximum between all blades within a UV system.

   a. Define topology-aware node definitions to parallel the compute nodes reachable within a specific hop count. For the UV 1000, this means the sockets of each blade will belong to six separate node set definitions; i.e., one each for 0, 1, 2, 3, 4, and 5 hops).

   b. Define multiple node sets for different nodes reachable in a specific hop count based on the context of where they are in the network topology; that is, you must create a separate and distinct node set definition for each pair of blades reachable with one hop, for each IRU for its nodes reachable in three hops, etc.

   c. Moab node sets are usually defined as compute node features; that is, each node set defined to Moab should appear as a "feature" name on one or more compute nodes. Which node set/feature names appear on each compute node depends on where the compute node is in the interconnect network topology.

      Since the SGI UV operating system identifies each blade socket as a separate NUMA node, each NUMA node within a UV system is traditionally an individual compute node to Moab (although TORQUE has the ability to redefine a compute node definition by grouping OS NUMA nodes, which some UV installations do to define a blade as a compute node).

      For the sake of illustration, this example assumes each OS NUMA node, which is a UV blade socket, is also a compute node in Moab. This means each compute node (blade socket) will have six feature names assigned, where each feature name must reflect both the compute node's location in the network topology and the hop count the name represents. A feature name is constructed by using the same root name for a hop count and a number for the topology location at the hop-count level.

      For example, the root feature name "blade" represents the zero-hop count and the numbers "0", "1,", etc, represent the consecutively numbered blades throughout the entire UV system, which yields feature names of "blade0" for the first blade in the system, "blade1" for the second blade, etc, to "blade127" for the last blade in a fully populated 4-rack UV system. To illustrate further,

the root feature name "iru" represents the 3-hops count and the numbers "0" through "7" represent the eight IRUs within a full 4-rack UV system.

d. For each compute node, configure the correct feature name for each of the hop counts possible and its location within the topology at the hop-count level (e.g., blade (0 hops), blade pair (1 hop), odd- or even-numbered nodes within an IRU (2 hops), IRU (3 hops), odd- or even-numbered nodes within the UV (4 hops), and UV system (5 hops)). The following example illustrates the feature names assigned to the compute nodes for an SGI UV 1000 system using the following root feature names.

   - blade (0 hops)
   - pair (1 hop)
   - eiru (2 hops for even-numbered blades within an IRU)
   - oiru (2 hops for odd-numbered blades within an IRU)
   - iru (3 hops)
   - esys (4 hops for even-numbered blades within a UV system)
   - osys (4 hops for odd-numbered blades within a UV system)
   - sys (5 hops)

   Note that nodes 0 and 1 are not given any feature names. This is because the operating system instance for the UV system runs on the first blade and in order to not adversely affect OS performance, no jobs should run on the same compute resources as the operating system; hence, these nodes have no node set feature names and therefore will never be chosen to run jobs. In addition, some of the first feature names at a specific hop count-level are omitted (such as pair0) since it makes no sense to define them when the first blade is a substantial part of the nodes making up a node set.

   The node name of a UV system has the same name as the UV system's host name plus the NUMA node's relative socket number.

```
/var/spool/torque/server_priv/nodes:
sys-uv0
sys-uv1
sys-uv2   blade1          oiru0 iru0 osys sys
sys-uv3   blade1          oiru0 iru0 osys sys
sys-uv4   blade2   pair1  eiru0 iru0 esys sys
sys-uv5   blade2   pair1  eiru0 iru0 esys sys
sys-uv6   blade3   pair1  oiru0 iru0 osys sys
sys-uv7   blade3   pair1  oiru0 iru0 osys sys
sys-uv8   blade4   pair2  eiru0 iru0 esys sys
sys-uv9   blade4   pair2  eiru0 iru0 esys sys
sys-uv10  blade5   pair2  oiru0 iru0 osys sys
sys-uv11  blade5   pair2  oiru0 iru0 osys sys
sys-uv12  blade6   pair3  eiru0 iru0 esys sys
sys-uv13  blade6   pair3  eiru0 iru0 esys sys
sys-uv14  blade7   pair3  oiru0 iru0 osys sys
sys-uv15  blade7   pair3  oiru0 iru0 osys sys
sys-uv16  blade8   pair4  eiru0 iru0 esys sys
sys-uv17  blade8   pair4  eiru0 iru0 esys sys
sys-uv18  blade9   pair4  oiru0 iru0 osys sys
sys-uv19  blade9   pair4  oiru0 iru0 osys sys
```

```
sys-uv20  blade10  pair5  eiru0 iru0 esys sys
sys-uv21  blade10  pair5  eiru0 iru0 esys sys
sys-uv22  blade11  pair5  oiru0 iru0 osys sys
sys-uv23  blade11  pair5  oiru0 iru0 osys sys
sys-uv24  blade12  pair6  eiru0 iru0 esys sys
sys-uv25  blade12  pair6  eiru0 iru0 esys sys
sys-uv26  blade13  pair6  oiru0 iru0 osys sys
sys-uv27  blade13  pair6  oiru0 iru0 osys sys
sys-uv28  blade14  pair7  eiru0 iru0 esys sys
sys-uv29  blade14  pair7  eiru0 iru0 esys sys
sys-uv30  blade15  pair7  oiru0 iru0 osys sys
sys-uv31  blade15  pair7  oiru0 iru0 osys sys
sys-uv32  blade16  pair8  eiru1 iru1 esys sys
sys-uv33  blade16  pair8  eiru1 iru1 esys sys
sys-uv34  blade17  pair9  oiru1 iru1 osys sys
sys-uv35  blade17  pair9  oiru1 iru1 osys sys
...
sys-uv62  blade31  pair15 oiru1 iru1 osys sys
sys-uv63  blade31  pair15 oiru1 iru1 osys sys
sys-uv64  blade32  pair16 eiru2 iru2 esys sys
sys-uv65  blade32  pair16 eiru2 iru2 esys sys
...
sys-uv126 blade63  pair31 oiru3 iru3 osys sys
sys-uv127 blade63  pair31 oiru3 iru3 osys sys
sys-uv128 blade64  pair32 eiru4 iru4 esys sys
sys-uv129 blade64  pair32 eiru4 iru4 esys sys
...
sys-uv190 blade95  pair47 oiru5 iru5 osys sys
sys-uv191 blade95  pair47 oiru5 iru5 osys sys
sys-uv192 blade96  pair48 eiru6 iru6 esys sys
sys-uv193 blade96  pair48 eiru6 iru6 esys sys
...
sys-uv252 blade126 pair63 eiru7 iru7 esys sys
sys-uv253 blade126 pair63 eiru7 iru7 esys sys
sys-uv254 blade127 pair63 oiru7 iru7 osys sys
sys-uv255 blade127 pair63 oiru7 iru7 osys sys
```

4. Define the order in which Moab should check node sets for available nodes. Since the NODESETPRIORITYTYPE has a value of *FIRSTFIT*, the node sets must be ordered from smallest to largest so Moab will always choose the node set with the fewest nodes required to satisfy the job's request. This means listing all blades, blade pairs, even and odd IRUs, IRUs, even and odd system, and system, respectively.

```
moab.cfg:
NODESETLIST
blade1,blade2,blade3,…,blade127,pair1,pair2,pair3,…,pair63,eiru0,oiru0,eiru1,oiru1,
…,eiru7,oiru7,iru0,iru1,…,iru7,esys,osys,sys
```

5. Configure Moab to use the *PRIORITY*NODEALLOCATIONPOLICY. This allocation policy causes Moab to allocate enough nodes to fulfill a job's processor and memory requirement.

```
NODEALLOCATIONPOLICY PRIORITY
```

6. Set NODEACCESSPOLICY to *SINGLEJOB* to ensure that Moab will schedule large memory requests correctly and efficiently. This is necessary even when a job uses only the memory of a NUMA node.

```
NODEACCESSPOLICY SINGLEJOB
```

   The policy *SINGLEJOB* tells Moab not to allow jobs to share NUMA resources (cores and memory), which for a shared-memory system is very important for fast job execution. For example, if Moab

scheduled a job to use the cores of a NUMA node where memory is used by another job, both jobs would execute slowly (up to 10 times more slowly).

Job Submission

Jobs can request processors and memory using the `-l nodes=<number of cpus>` and `-l mem=<amount of memory>` syntaxes. You should not have JOBNODEMATCHPOLICY *EXACTNODE* configured on a NUMA system. You must use the `sharedmem` job flag on submission to force the job to run only on a sharedmem partition or cluster and to indicate that the job can span multiple nodes. For example:

```
qsub -l nodes=3,mem=640sgb,flags=sharedmem
```

Appendix H: Interfacing with Moab (APIs)

Moab provides numerous interfaces allowing it to monitor and manage most services and resources. It also possesses flexible interfaces to allow it to interact with peer services and applications as both a broker and an information service. This appendix is designed to provide a general overview and links to more detailed interface documentation.

- H.1 Moab Query and Control APIs
  - Allow external portals and services to obtain information about compute resources, workload, and usage statistics.
- H.2 Resource Management Interfaces
  - Allow Moab to monitor, schedule, and control services and resources.
- H.3 Identity and Credential Management Interfaces
  - Allow monitoring and active management of user configuration, credentials, policies, and usage information.
- H.4 Accounting and Event Interfaces
  - Allow import/export of accounting and event information to external entities.
- H.5 Discovery/Directory Services
- H.6 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 Suite provide a (Moab) workload manager server that supports a broad array of client services. These services can be directly accessed via Moab client commands.

H.1.3 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:

```
> showq --format=xml
<Data>
<Object>queue</Object>
<cluster LocalActiveNodes="1" LocalAllocProcs="1" LocalIdleNodes="0"
LocalIdleProcs="3" LocalUpNodes="1"
  LocalUpProcs="4" RemoteActiveNodes="0" RemoteAllocProcs="0" RemoteIdleNodes="0"
RemoteIdleProcs="0"
  RemoteUpNodes="0" RemoteUpProcs="0" time="1128451812"></cluster>
<queue count="1" option="active">
<job AWDuration="11672" EEDuration="1128451812" Group="[DEFAULT]" JobID="Moab.2"
MasterHost="cw2" PAL="2"
  QOS="bug3" ReqAWDuration="54000" ReqNodes="1" ReqProcs="1" RsvStartTime="1128451812"
RunPriority="0"
  StartPriority="1" StartTime="1128451812" StatPSDed="11886.580000"
StatPSUtl="11886.580000" State="Running"
  SubmissionTime="1128451812" SuspendDuration="0" User="smith"></job>
</queue>
<queue count="1" option="eligible">
<job EEDuration="1128451812" Group="jacksond" JobID="customer.35" QOS="bug"
ReqAWDuration="3600"
  ReqProcs="1" StartPriority="1" StartTime="0" State="Idle"
SubmissionTime="1128451812" SuspendDuration="0"
  User="johnson"></job>
<queue><queue count="0" option="blocked"></queue>
</Data>
```

Common Query/Control Services

- jobs
  - query status - mdiag -j (XML details)
  - submit - msub (XML format)
  - cancel - mjobctl -c
- nodes
  - query status - mdiag -n (XML details)
  - create resource reservation - mrsvctl -c
  - destroy resource reservation - mrsvctl -r

H.2 Resource Management Interfaces

Moab can monitor, schedule, and control services and resources using multiple protocols. These protocols include the following:

- LDAP
- script/flat file
- Resource Manager Specific Interfaces - LSF, SGE, TORQUE, PBSPro, Loadleveler, and so forth

Using the resource manager interfaces, Moab can do the following:

- monitor resources (compute host, network, storage, and software license based resources)
    - load configuration, architecture, and feature information
    - load state, utilization, and workload information
    - load policy and ownership information
- manage resources
    - dynamically reconfigure and reprovision resource hardware (processors, memory, etc.)
    - dynamically reconfigure and reprovision resource software (operating system, application software, file system 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

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 Discovery/Directory Services

Moab can import and export key event information regarding workload, cluster resources, cluster 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.6 Job Submission and Management Interface

Moab provides interfaces to enable the following services:

- Resource Availability Query
    - Determine quantity, state, and configuration of configured resources (idle, busy, and down nodes)
    - Determine quantity and configuration of all available resources (idle nodes)
    - Determine resources available subject now and in the future for potential job
    - Determine best target cluster destination for potential job
    - Determine largest/longest job which could start immediately
    - Determine estimated start time for potential job
    - Determine earliest guaranteed start time for potential job
- Reserve Resources
    - Reserve specific resources for desired time frame
- Submit Job (XML format)
    - Submit job to specific cluster
    - Submit job to global job queue

- Manage Job
  - Hold job
  - Adjust job priority
  - Modify job executable, args, data requirements, job dependencies, duration, hostcount, or other attributes
  - Suspend/resume job
  - Checkpoint/requeue job
  - Cancel job
  - Migrate job
  - Adjust job quality of service (QoS)
- Query Job
  - Determine job state, utilization, or output results for idle, active, or completed job
  - Determine estimated start time
  - Determine guaranteed start time

Appendix I: Considerations for Large Clusters

- I.1 Resource Manager Scaling
- I.2 Handling Large Numbers of Jobs
- I.3 Handling Large Numbers of Nodes
- I.4 Handling Large Jobs
- I.5 Handling Large SMP Systems
- I.6 Server Sizing

There are several key considerations in getting a batch system to scale.

I.1 Resource Manager Scaling

Proper Resource Manager Configuration

- TORQUE
  - General Scaling Overview
- OpenPBS/PBSPro
  - Manage Direct Node Communication with NODEPOLLFREQUENCY

I.2 Handling Large Numbers of Jobs

Reduce command processing time

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

When you set **UIMANAGEMENTPOLICY** *FORK*, Moab forks a copy of itself that will listen for client commands on a separate port, which you must configure with **CLIENTUIPORT**. This forked process responds to checkjob, showbf, showres, and showstart until the main scheduling cycle has finished. After that, it is killed and the normal process resumes responding to client commands. Moab prints a disclaimer at the top of each command that was populated by the forked process stating that the information may be a few seconds stale.

Example 25-1: Sample configuration

```
UIMANAGEMENTPOLICY      FORK
CLIENTUIPORT            41560
```

Moab forks a copy of itself on port 41560, where it will watch for checkjob, showbf, showres, and showstart commands until the main scheduling process completes.

Example 25-2: Sample command output

```
$ checkjob 34

------------------------------------------------------------------
NOTE: The following information has been cached by the remote server
and may be slightly out of date.
------------------------------------------------------------------

job 34

State: Idle
Creds: user:wightman group:company class:batch
WallTime: 00:00:00 of 00:01:00
SubmitTime: Thu May 22 14:17:06
(Time Queued Total: 00:00:18 Eligible: 00:00:18)

TemplateSets: DEFAULT
Total Requested Tasks: 1

Req[0] TaskCount: 1 Partition: ALL

SystemID: scale
SystemJID: 34

IWD: $HOME/test/scale
SubmitDir: $HOME/test/scale
Executable: sleep 60
```

Limited Job Checkpointing

Use the **LIMITEDJOBCP** on page 862 parameter. By default, Moab will checkpoint information about every job it reads from its resource managers. When a cluster routinely runs more than 15000 jobs, they may see some speed-ups by limiting which jobs are checkpointed. When LIMITEDJOBCP is set to *TRUE*,

Moab will only checkpoint jobs that have a hold, a system priority, jobs that have had their QoS modified, and a few other limited attributes. Some minimal statistical information is lost for jobs that are not checkpointed.

Minimize Job Processing Time

Use the **ENABLEHIGHTHROUGHPUT** on page 824 parameter. By default, Moab processes all job attributes, filters, remap classes, job arrays, and other information when a job is submitted. This requires full access to the Moab configuration and significantly increases the processing time Moab needs when jobs are submitted. By setting ENABLEHIGHTHROUGHPUT to *TRUE*, Moab stores the job information in an internal queue and returns the job ID immediately. The internal queue is processed when Moab begins its next scheduling iteration. This enables Moab to process hundreds of jobs per second rather than 20-30 per second. Because the jobs are processed in a separate queue after the job has been returned, it is recommended that MAILPROGRAM be configured. Moab will send an email to the user if a job is rejected.

Because the job is not fully processed, some attributes may change after the job has been submitted. For example, when a job class is remapped, the new class is not reflected until Moab begins its next scheduling iteration. Additionally, job arrays are not instantiated until Moab begins its next scheduling cycle.

If **ENABLEHIGHTHROUGHPUT** on page 824 is *TRUE*, you must set **NODEALLOCATIONPOLICY** on page 872 to *FIRSTAVAILABLE*.

Load all Non-Completed Jobs at Startup

Use the **LOADALLJOBCP** on page 863 parameter. By default, Moab loads non-complete jobs for active resource managers only. By setting LOADALLJOBCP to *TRUE*, Moab will load all non-complete jobs from all checkpoint files at startup, regardless of whether their corresponding resource manager is active.

Reducing Job Start Time

Use the ASYNCSTART parameter. By default, Moab will launch one job at a time and verify that each job successfully started before launching a subsequent job. For organizations with large numbers of very short jobs (less than 2 minutes in duration), the delay associated with confirming successful job start can lead to productivity losses. If tens or hundreds of jobs must be started per minute, and especially if the workload is composed primarily of serial jobs, then the resource manager *ASYNCSTART* flag may be set. When set, Moab will launch jobs optimistically and confirm success or failure of the job start on the subsequent scheduling iteration. Also consider adding the *ASYNCDELETE* flag if users frequently cancel jobs.

Increase TORQUE Timeout

The number of jobs in the queue affects the time needed by TORQUE to get updates to Moab. Adjust the Moab timeout to prevent "End of File" errors on scheduling intervals. This is environment specific, but in general if you have more than 50,000 jobs in the queue you should make this adjustment.

```
RMCFG[torque] TIMEOUT=300
```

This sets the connection timeout to 300 seconds.

Reducing Job Reservation Creation Time

Use the **RMCFG** on page 906 **JOBRSVRECREATE** on page 517 attribute. By default, Moab destroys and re-creates job reservations each time a resource manager updates any aspect of a job. Historically, this stems from the fact that certain resource managers would inadvertently or intentionally migrate job tasks from originally requested nodes to other nodes. To maintain synchronization, Moab would re-create reservations each iteration thus incorporating these changes. On most modern resource managers, these changes never occur, but the effort required to handle this case grows with the size of the cluster and the size of the queue. Consequently, on very large systems with thousands of nodes and thousands of jobs, a noticeable delay is present. By setting **JOBRSVRECREATE** to *FALSE* on resource managers that do not exhibit this behavior, significant time savings per iteration can be obtained.

Optimizing Backfill Time

Use the OPTIMIZEDBACKFILL flag, which speeds up backfill when a system reservation is in use.

Constraining Moab Logging - LOGLEVEL

Use the **LOGLEVEL** on page 865 parameter. When running on large systems, setting **LOGLEVEL** to 0 or 1 is normal and recommended. Only increase **LOGLEVEL** above 0 or 1 if you have been instructed to do so by Moab support.

Preemption

When preemption is enabled Moab can take considerably more time scheduling jobs for every scheduling iteration. Preemption increases the number of options available to Moab and therefore takes more time for Moab to optimally place jobs. If you are running a large cluster or have more than the usual amount of jobs (>10000), consider disabling preemption. If disabling preemption is not possible, consider limiting its scope to only a small subset of jobs (as both preemptors and preemptees).

Handling Transient Resource Manager Failures

Use the **RMCFG** **MAXITERATIONFAILURECOUNT** on page 519 attribute.

Constrain the number of jobs preempted per iteration

Use the JOBMAXPREEMPTPERITERATION parameter.

For very large job count systems, configuration options controlling the maximum supported limits may need to be adjusted including the maximum number of reservations and the maximum number of supported evaluation ranges.

I.3 Scheduler settings

If using Moab, there are a number of parameters which can be set on the scheduler which may improve TORQUE performance. In an environment containing a large number of short-running jobs, the **JOBAGGREGATIONTIME** parameter can be set to reduce the number of workload and resource queries performed by the scheduler when an event based interface is enabled. Setting **JOBAGGREGATIONTIME** instructs the scheduler to ignore events coming from the resource manager and to scheduling at regular intervals, rather than around resource manager events. If the pbs_server daemon is heavily loaded and PBS API timeout errors (i.e. "Premature end of message") are reported within the scheduler, the **TIMEOUT** attribute of the **RMCFG** parameter may be set with a value of between 30 and 90 seconds.

I.3 Handling Large Numbers of Nodes

For very large clusters (>= 10,000 processors) default scheduling behavior may not scale as desired. To address this, the following parameters should be considered:

Parameter	Recommended Settings
RMPOLLINTERVAL	In large node environments with large and long jobs, scheduling overhead can be minimized by increasing RMPOLLINTERVAL above its default setting. If an event-driven resource management interface is available, values of two minutes or higher may be used. Scheduling overhead can be determined by looking at the scheduling load reported by mdiag -S.
LIMITEDNODECP	Startup/shutdown time can be minimized by disabling full node state checkpointing that includes some statistics covering node availability.
SCHEDCFG FLAGS=" FASTRSVSTARTUP on page 1138	When you have reservations on a large number of nodes, it can take Moab a long time to recreate them on startup. Setting the *FASTRSVSTARTUP* scheduler flag greatly reduces startup time.

* For clusters where the number of nodes or processors exceeds 50,000, the maximum stack size for the shell in which Moab is started may need to be increased (as Moab may crash if the stack size is too small). On most Unix/Linux based systems, the command ulimit -s unlimited may be used to increase the stack size limit before starting Moab. This may be placed in your Moab startup script.

See Appendix D for further information on default and supported object limits.

Avoid adding large numbers of NODECFG lines in the `moab.cfg` or `moab.d/*.cfg` files to keep the Moab boot time low.

For example, adding a configuration line to define features for each node in a large cluster (such as `NODECFG[x] Features+=green,purple`) can greatly increase the Moab boot time. If Moab processes 15 node configuration lines per second for a 50,000-node system, it could add approximately 55 minutes of node configuration processing to the Moab boot time.

In this case, it is better to define the node features in the resource manager configuration.

I.4 Handling Large Jobs

For large jobs, additional parameters beyond those specified for large node systems may be required. These include settings for the maximum number of tasks per job, and the maximum number of nodes per job.

I.5 Handling Large SMP Systems

For large-way SMP systems (> 512 processors/node) Moab defaults may need adjustment.

Parameter	Recommended Settings
MAXRSVPERNODE	By default, Moab does not expect more than 48 jobs per node to be running or have future reservations. Increasing this parameter to a value larger than the expected maximum number of jobs per node is advised.

I.6 Server Sizing

See Hardware and Software Requirements for recommendations.

Related topics

- Appendix D: Adjusting Default Limits

Appendix J: Configuring Moab as a Service

Scripts that follow can be used to start up Moab services automatically upon a reboot. To enable a service script, copy the script to `/etc/rc.d/init.d/S97moab`, edit the file to make needed localization changes (adjust binary paths, execution user, etc), and add links to the `rc3.d`and `rc5.d` directories as in the example that follows:

```
> cp mwm.service /etc/rc.d/init.d/S97moab
> vi /etc/rc.d/init.d/S97moab
    (make needed localizations)
> ln -s /etc/rc.d/init.d/S97moab /etc/rc.d/rc3.d
> ln -s /etc/rc.d/init.d/S97moab /etc/rc.d/rc5.d
```

J.1 Moab Workload Manager Service Scripts

- Moab Workload Manager Script
- Moab Workload Manager + TORQUE Script

Appendix K: Migrating from 3.2

Overview

This guide is intended to help facilitate migrating from Maui to Moab. If you do not have Moab yet, you can download a free evaluation version. At a high level, migrating from Maui 3.2 to Moab involves minimal effort. In fact, Moab fully supports all Maui parameters and commands. Migration can consist of nothing more than renaming `maui.cfg` to `moab.cfg` and launching Moab using the moab command. With this migration, the biggest single issue is becoming aware of all the new facilities and capabilities available within Moab. Beyond this, migration consists of a few minor issues that may require attention such as some statistics and priorities.

Another approach of migrating from Maui to Moab is to configure Moab in Monitor mode and run it beside Maui. Maui will continue to perform the scheduling and control workload. Moab will simply monitor the cluster environment using the policies configured in `moab.cfg`. Moab will not have the ability to affect workload, providing a safe and risk-free environment to evaluate Moab without affecting your production environment. You can also have Moab capture resource and workload trace files and allow Moab to simulate what it would have done if it controlled workload. When you feel comfortable with and want to run Moab live on your cluster, all you need to do is change the mode to NORMAL, stop Maui, and restart Moab. Current jobs will remain running and Moab will take over control of scheduling.

As with any migration, we suggest that you back up important files such as the following: `maui.cfg`, `maui.log` and `maui.ck`.

View the Flash demo of migrating from Maui to Moab.

Migrating from Maui to Moab

1. Install Moab Workload Manager. (Installation Instructions)
2. Copy your `maui.cfg` file to the `MOABHOMEDIR/etc` (`/opt/moab/etc`) and rename it `moab.cfg`.
3. Stop Maui.
4. Start Moab.
5. If Applicable: Re-apply those configurations found in the Statistics and Checkpointing section that need adjustment after migration as well as any parameters in `moab.cfg` that point to a Maui file like `maui.log`.

Running Maui and Moab Side-By-Side

1. Install Moab Workload Manager on your cluster. (Installation steps will differ slightly from a typical installation.)
   a. Run ./configure.
   b. Run make.
   c. You will need to set your *MOABHOMEDIR* environment variable to the location where you built Moab by typing `export MOABHOMDIR=[make directory]`.
2. To have Moab use all the same policies as Maui, copy `maui.cfg` to the MOAB`HOMEDIR/etc` and rename it `moab.cfg`.
   - You can also start your `moab.cfg` file from scratch. Just use the `moab.cfg` already in the `MOABHOMEDIR/etc`.
3. Make sure that the port in `moab.cfg` is different than the port used in `maui.cfg`.
4. In the `moab.cfg` file, add the parameter, `SERVERMODE=MONITOR`.
   - If you used the `moab.cfg` from scratch, on the **SCHEDCFG** line add `MODE=MONITOR`.

5. You will need to either put the Moab commands in your environment path (located in `MOABHOMEDIR/bin`) or run the commands from their location if you still want to use the Maui commands in your environment path.
6. Run Moab Workload Manager using the moab command located in `MOABHOMEDIR/bin`.

Other Notes

The following are minor differences between Maui and Moab and changes you may need to make:

File Naming

Moab uses slightly different naming than Maui. The following table displays these changes:

File	Maui	Moab
executable	maui	moab
logs	`maui.log`	`moab.log`
configuration file	`maui.cfg`	`moab.cfg`

Statistics and Checkpointing

Moab supports Maui version 3.2 or higher workload traces (statistics) allowing it to process historical statistics based on these traces as well as generate simulations based on them. No changes are required to use these statistics. See the Simulation Configuration documentation for more information on trace files. You can also view a flash demonstration of the simulation mode.

Moab does not support the Maui 3.2 checkpointing format. Because of this, state information checkpointed under Maui will not be available at the time of the migration. The loss of this information will have the following impact:

- Admin reservations, if any, will need to be re-created.
- Processed credential and scheduler statistics (displayed by showstats) will be lost.
- Admin job system priority configured by the setspri command and QoS assignments configured by the setqos command, if any, will be lost.

Verify Configuration File Compatibility

The command mdiag -C will perform diagnostics on your new configuration file and may prove helpful in identifying any issues.

Environment Variables

Scheduler environment variables are supported under Moab with obvious naming changes. Sample environment variables follow:

Maui	Moab
MAUIHOMEDIR	*MOABHOMEDIR*
MAUIDEBUG	*MOABDEBUG*
MAUICRASHVARIBALE	*MOABCRASHVARIABLE*
MAUIENABLELOGBUFFERING	*MOABENABLELOGBUFFERING*
MAUIRECOVERYACTION	*MOABRECOVERYACTION*
MAUI-COMMANDS-PATH	*MOAB-COMMANDS-PATH*
MAUIENABLELOGBUFFERING	*MOABENABLELOGBUFFERING*

Appendix R: Node Allocation Plug-in Developer Kit

- R.1 Overview
  - R.1.1 Writing the plugin
    - R.1.1.1 API & Data Structures
  - R.1.2 Moab configuration
    - R.1.2.1 Moab.cfg
    - R.1.2.2 Syntax rules
    - R.1.2.3 Troubleshooting

R.1 Overview

Each time Moab schedules a job, it must choose the nodes on which the job will run. Moab uses the Node Allocation policy to select the available nodes to be used. Because there are so many different systems and cluster topologies, you now have the ability to create and use a node allocation plugin for allocating nodes based on your cluster's interconnect topology.

The plugin policy allows you to write your own algorithm to choose which nodes will be used. This algorithm is contained in a shared library that Moab loads at run time.

To obtain the Plug-in Developer Kit (PDK) with the header file and example code, contact your sales representative.

R.1.1 Writing the plugin

A plugin is a shared library that has specific functions and variables that will be called directly from Moab. The plugin conforms to a C language API. The API is specified through an include file: `moab-plugin.h`. This file must be included in the plugin code. The include file provides function definitions, structures and variables that will be used when communicating with Moab.

When you write the plugin, you need to ensure that the plugin code is robust. If the plugin crashes, Moab will crash. You will need to handle your own memory appropriately. If the plugin has memory leaks, Moab will have similar issues. If you want to maintain logs, the plugin will need to be responsible for its own logging.

R.1.1.1 API and Data Structures

The Application Programmer Interface (API) for the Moab Node Allocation Plugin consists of three data items and three entry points that must be supplied to Moab by the plugin.

Plugin Supplied Data	Description
const char *PLUGIN_NAME = "Node Allocation plugin 1.1";	This character pointer is used by Moab when logging information regarding the operation of the plugin.
const char *PLUGIN_TYPE = PLUGIN_TYPE_ NAME_ NODEALLOCATION;	This character pointer is used by Moab to verify the type of plugin. The value of this data is supplied by the `moab-plugin.h` source file. The plugin must set this as shown so that Moab does not attempt to use a plugin incorrectly. Moab uses this to determine whether the plugin API type is correct and to allow Moab to correctly communicate with the plugin.
const char *PLUGIN_VERSION = PLUGIN_API_ VERSION;	This character pointer is used by Moab to verify the API version number. The value of this data is supplied by the `moab-plugin.h` source file. The plugin must set this as shown so that the correct version of the `moab-plugin.h` is supplied to Moab. Moab uses this to determine whether the API version is correct and to allow Moab to correctly communicate with the plugin.

Load Time API	Description
initialize()	`int initialize(const char *name, void **data_handle)` The plugin must supply an initialize() entry point. This entry point is called for each use instance of the plugin. For example, if the plugin is used on two different partitions, the initialize() entry point will be called once for each partition. • Name — The name is the unique identifier which is used to distinguish multiple instances of the plugin and for logging. When configured globally, the name "ALL" will be given. • Data handle — The data_handle points to a location where the plugin should store a pointer to any internal data needed by the plugin between calls to the API. The actual format and structure of the data is up to the plugin. Moab will supply this pointer back to the plugin each time a plugin entry point is called. This data can provide context for the plugin usage instance.

Return codes	The initialize() entry point should return one of two return statuses as defined in `moab-plugin.h`:
	`#define PLUGIN_RC_SUCCESS 0`
	`#define PLUGIN_RC_FAILURE 1`

Gathering node info	The initialize() entry point must gather any information about system nodes, their topology, interconnection, and configuration that it needs to make correct node allocations. Since Moab does not know what information the plugin may need, the plugin must gather this information itself.
Memory considerations	The plugin may allocate memory for temporary or persistent data as needed, but *must* de-allocate or return the memory when finished. Not returning memory can result in memory leaks and unstable operation on the part of Moab.
Multiple access	A given loaded plugin can be used by more than one partition. This means that the plugin must maintain its internal data in such a way that calls to the plugin for the separate partitions do not conflict. It is recommended that internal data be allocated and a pointer to the data be kept in the data_handle described above as opposed to using global or static variables. Any global or static data will be shared between possible multiple instances of the plugin.

Runtime API	Description
node_allocate()	`int node_allocate ( void *data_handle, const char *job_name, int container_count, nalloc_container_t container[])` The plugin must provide a node_allocate() entry point. This entry point is called each time Moab needs to determine where (on what nodes) a job will eventually run. Note that this entry point can be called many times before the job is actually scheduled to run. • Data structures — Moab uses C data structures to pass information and lists of nodes to the plugin and receive them back from the plugin. See `moab-plugin.h` for the definitions of these structures and for information on how they relate to one another.

Runtime API	Description
Operations	A node allocation request consists of one or more requirements. Each of these requirements is provided within a "container" structure. The container has information regarding the requirement to be met, the count and list of all nodes that are available to meet the requirement and a place to return the list of nodes that the plugin has chosen to use for the job.

Command	Moab Job Task Count	Job Node Count	Job Tasks Per Node	Node CFG Procs	Node AVL Procs	Plugin Node Mapped TC	requirement->taskcount	return_node_count
Non-ExactNode								
-l nodes=12	12	0	0	8	8	8	12	2
-l nodes=12:ppn=2	24	0	2	8	8	8	24	3
ExactNode								
-l nodes=4	4	4	0	8	8	1	4	4
-l nodes=4:ppn=2	8	4	2	8	8	2	8	4
-l nodes=12	12	0	0	8	6	6	12	2

The duty of the plugin is to use the information that it has previously gathered (during the initialization) to select from the available nodes those that will best fulfill the requirements.

The basic algorithm is to consume all the task count and memory on each node until the consumed task count is greater than or equal to the container's task_count and memory requirements.

A job's task count is calculated differently based on the JOBNODEMATCHPOLICY parameter. By default, it isn't defined and `-l nodes=#` actually requests the number of tasks without respect to the number of nodes. In this case, the plugin should consume all the tasks of each chosen node until the task count is greater and/or equal to the container's task count requirement. The plugin is for node allocation and not task placement.

Runtime API	Description
	When the JOBNODEMATCHPOLICY EXACTNODE is configured, then `-l nodes=#` means the job wants # of nodes with 1 task per node. In this case, the nodes passed to the plugin will have a task count that is mapped down to what the job can only use on that node. Each node's task count should be consumed on each node until the summed amount is equal to the container's requirement task count requirement. The following table shows how commands are interpreted by Moab and translated to the plugin and what is expected of the plugin.

Errors and return codes	The plugin may internally log any errors encountered and must return a success or error status as defined in `moab-plugin.h`:
	`#define PLUGIN_RC_SUCCESS 0`
	`#define PLUGIN_RC_FAILURE 1`

Multiple access safe	The node_allocate() entry point must support multiple access as described above.

Unload Time API	Description
finish()	`void finish(void *data_handle)` The plugin must supply a finish() entry point. This entry point is called when Moab is preparing to disable and/or unload an instance of the plugin.
Memory/resource cleanup	The plugin must de-allocate and free up any resources acquired either during the initialize () entry point or during any calls to the node_allocate() entry point. When the last entry point returns, there should be no allocated memory or other resources still in use by the plugin instance.
Multiple access safe	The finish() entry point must support multiple access as described above.

R.1.2 Moab configuration

The actual loading of a plugin is accomplished by specifying the plugin in the Moab configuration file, `moab.cfg`.

R.1.2.1 Moab.cfg

We recommend that you store all Moab plugins in the `$MOABHOMEDIR/lib` directory (e.g., `/opt/moab/lib`) as shared libraries (`*.so`). The name of the actual plugin shared library file is up to the plugin developer, which means you must give the correct name in the moab.cfg file to form the absolute plugin filename.

If a plug-in's specified shared library filename starts with a forward slash (`/`), it is an absolute file path name and Moab simply uses it without alteration. For example, if a plugin's specified shared library filename is `/opt/moab/plugins/plugin.so`, Moab will use it as the absolute plugin file path name.

If a plugin's specified shared library filename does not start with a forward slash (`/`), it is a plugin name and Moab forms the plugin's absolute path name by concatenating the Moab home directory, "/lib/lib", the specified plugin name, and ".so" to obtain the absolute path name. For example, if the *$MOABHOMEDIR* environment variable contains `/opt/moab` and the plugin name is `plugin`, Moab will create `/opt/moab/lib/libplugin.so` and use it as the absolute plugin file path name.

R.1.2.2 Syntax rules

In order for Moab to use a plugin for the Node Allocation policy, instead of a built-in Moab policy, you must configure the policy in the moab.cfg file with the value "PLUGIN:" followed by the plugin's shared library file name. The examples below assume the environment variable *$MOABHOMEDIR* has a value of `/opt/moab`. Note the use of relative and absolute plugin shared library file path names in the parameter value and how they affect Moab's construction of the full path name.

Par-tition	Plug-in Name	moab.cfg Parameter	Moab-derived Full Path Name
global	`plugin.so`	NODEALLOCATIONPOLICY *PLUGIN:plugin.so*	`/op-t/moab/lib/libplugin.so`
global	`/us-r/loc-al/plugins/plugin.so`	NODEALLOCATIONPOLICY *PLUGIN:/us-r/local/plugins/plugin.so*	`/us-r/loc-al/plugins/plugin.so`
abc	`plugin.so`	PARCFG[*abc*] NODEALLOCATIONPOLICY =*PLUGIN:plugin.so*	`/op-t/moab/lib/libplugin.so`
xyz	`/us-r/loc-al/plugins/plugin.so`	PARCFG[*xyz*] NODEALLOCATIONPOLICY= *PLUGIN:/us-r/local/plugins/plugin.so*	`/usr/local/plugins/plugin.so`

R.1.2.3 Troubleshooting

There are several commands that can be used to confirm that the Plugin Node Allocation Policy was loaded properly.

mschedctl -l

mschedctl -l is used to print out Moab's in memory configurations. If the plugin policy, with its full path, doesn't show for the configured partition then Moab failed to load the partition. Note that when the NODEALLOCATIONPOLICY is configured globally, it is configured on the "ALL" partition.

```
$ mschedctl -l -v|grep ^NODEALLOCATIONPOLICY
NODEALLOCATIONPOLICY[ALL] PLUGIN:/opt/moab/lib/libfirstavailable.so
NODEALLOCATIONPOLICY[a] PLUGIN:/opt/moab/lib/liblastavailable.so
NODEALLOCATIONPOLICY[b] CONTIGUOUS
NODEALLOCATIONPOLICY[c] PLUGIN:/opt/moab/lib/libfirstavailable.so
NODEALLOCATIONPOLICY[d] [NONE]
```

mdiag -C

mdiag -C is used to validate the moab.cfg configuration. With a plugin node allocation policy, Moab will validate that it can successfully load the plugin and that all of the required symbols are present.

```
$ mdiag -C
...
INFO: line #35 is valid: 'NODEALLOCATIONPOLICY PLUGIN:firstavailable'
INFO: line #36 is valid: 'PARCFG[a]NODEALLOCATIONPOLICY=PLUGIN:lastavailable'
INFO: line #37 is valid: 'PARCFG[b]NODEALLOCATIONPOLICY=CONTIGUOUS'
INFO: line #38 is valid: 'PARCFG[d]NODEALLOCATIONPOLICY=PLUGIN:firstavailable'
```

Appendix S: Scalable Systems Software Specification

- SSS Job Object Specification
- SSS Resource Management and Accounting Protocol Message Format
- SSS Node Object Specification
- SSS Resource Management and Accounting Protocol Wire Protocol

Scalable Systems Software Job Object Specification

SSSJob Object Specification
Draft Release Version 3.1.0
26 April 2011

Scott Jackson, PNNLStringDavid Jackson, Ames Lab
Brett Bode, Ames Lab

Status of This Memo

This document describes the job object to be used by Scalable Systems Software compliant components. It is envisioned for this specification to be used in conjunction with the SSSRMAP protocol with the job object passed in the Data field of Requests and Responses. Queries can be issued to a job-cognizant

component in the form of modified XPATH expressions to the Get field to extract specific information from the job object as described in the SSSRMAP protocol.

Abstract

This document describes the syntax and structure of the SSS job object. A job model is described that is flexible enough to support the specification of very simple jobs as well as jobs with elaborate and complex specification requirements in a way that avoids complex structures and syntax when it is not needed. The basic assumption is that a solitary job specification should be usable for all phases of the job lifecycle and can be used at submission, queuing, staging, reservations, quotations, execution, charging, accounting, etc. This job specification provides support for multi-step jobs, as well as jobs with disparate task descriptions. It accounts for operational requirements in a meta-scheduled environment where the job is executed by multiple hosts in different administrative domains that support different resource management systems.

Table of Contents

- Scalable Systems Software Job Object Specification
- Table of Contents
- 1.0 Introduction
    - 1.1 Goals
    - 1.2 Non-Goals
    - 1.3 Examples
        - 1.3.1 Very Simple Example
        - 1.3.2 Moderate Example
        - 1.3.3 Elaborate Example
- 2.0 Conventions used in this document
    - 2.1 Keywords
    - 2.2 Table Column Interpretations
    - 2.3 Element Syntax Cardinality
- 3.0 The Job Model
- 4.0 JobGroup Element
    - 4.1 JobGroup Properties
        - 4.1.1 Simple JobGroup Properties
        - 4.1.2 Job
        - 4.1.3 JobDefaults
    - 4.2 JobGroup Reference

- 5.0 Job and JobDefaults Element
  - 5.1 Job Properties
    - 5.1.1 Simple Job Properties
    - 5.1.2 Feature Element
    - 5.1.3 OutputFile Element
    - 5.1.4 ErrorFile Element
    - 5.1.5 InputFile Element
    - 5.1.6 NotificationList Element
    - 5.1.7 ResourceLimit Element
    - 5.1.8 Credentials
    - 5.1.9 Environment Element
      - 5.1.9.1 Variable Element
    - 5.1.10 Node Element
    - 5.1.11 TaskDistribution Element
    - 5.1.12 Dependency Element
    - 5.1.13 Consumable Resources
    - 5.1.14 Resource Element
    - 5.1.15 Extension Element
    - 5.1.16 TaskGroup
    - 5.1.17 TaskGroupDefaults
  - 5.2 Job Reference
- 6.0 TaskGroup and TaskGroupDefaults Element
  - 6.1 TaskGroup Properties
    - 6.1.1 Simple TaskGroup Properties
    - 6.1.2 Task
    - 6.1.3 TaskDefaults
  - 6.2 TaskGroup Reference
- 7.0 Task and TaskDefaults Element
  - 7.1 Task Properties
    - 7.1.1 Simple Task Properties
  - 7.2 Task Reference

- 8.0 Property Categories
  - 8.1 Requested Element
  - 8.2 Delivered Element
- 9.0 AwarenessPolicy Attribute
- 10. References
- Appendix A
- Units of Measure Abbreviations

1.0 Introduction

This specification proposes a standard XML representation for a job object for use by the various components in the SSS Resource Management System. This object will be used in multiple contexts and by multiple components. It is anticipated that this object will be passed via the Data Element of SSSRMAP Requests and Responses.

1.1 Goals

There are several goals motivating the design of this representation.

The representation needs to be inherently flexible. We recognize we will not be able to exhaustively include the ever-changing job properties and capabilities that constantly arise.

The representation should use the same job object at all stages of that job's lifecycle. This object will be used at job submission, queuing, scheduling, charging and accounting, hence it may need to distinguish between requested and delivered properties.

The design must account for the properties and structure required to function in a meta environment. It needs to include the capability to support local mapping of properties, global name spaces, etc.

The equivalent of multi-step jobs must be supported. Each step (job) can have multiple logical task descriptions.

Many potential users of the specification will not be prepared to implement the complex portions or fine-granularity that others need. There needs to be a way to allow the more complicated structure to be added as needed while leaving more straightforward cases simple.

There needs to be guidance for how to understand a given job object when higher order features are not supported by an implementation, and which parts are required, recommended and optional for implementers to implement.

It needs to support composite resources.

It should include the ability to specify preferences or fuzzy requirements.

1.2 Non-Goals

Name space considerations and naming conventions for most property values are outside of the scope of this document.

1.3 Examples

Example 25-3: Very Simple Example

This example shows a simple job object that captures the requirements of a simple job.

```
<Job>
  <Id>PBS.1234.0</Id>
  <State>Idle</State>
  <User>scottmo</User>
  <Executable>/bin/hostname</Executable>
  <Processors>16</Processors>
  <Duration>3600</Duration>
</Job>
```

Example 25-4: Moderate Example

This example shows a moderately complex job object that uses features such as required versus delivered properties.

```
<Job>
  <Id>PBS.1234.0</Id>
  <Name>Heavy Water</Name>
  <Project>nwchemdev</Project>
  <User>peterk</User>
  <Application>NWChem</Application>
  <Executable>/usr/local/nwchem/bin/nwchem</Executable>
  <Arguments>-input basis.in</Arguments>
  <InitialWorkingDirectory>/home/peterk</InitialWorkingDirectory>
  <Machine>Colony</Machine>
  <QualityOfService>BottomFeeder</QualityOfService>
  <Queue>batch_normal</Queue>
  <State>Completed</State>
  <StartTime>1051557713</StartTime>
  <EndTime>1051558868</EndTime>
  <Charge>25410</Charge>
  <Requested>
    <Processors op="GE">12</Processors>
    <Memory op="GE" units="GB">2</Memory>
    <Duration>3600</Duration>
  </Requested>
  <Delivered>
    <Processors>16</Processors>
    <Memory metric="Average" units="GB">1.89</Memory>
    <Duration>1155</Duration>
  </Delivered>
  <Environment>
    <Variable name="PATH">/usr/bin:/home/peterk</Variable>
  </Environment>
</Job>
```

Example 25-5: Elaborate Example

This example uses a job group to encapsulate a multi-step job. It shows this protocol's ability to characterize complex job processing capabilities. A component that processes this message is free to retain only that part of the information that it requires. Superfluous information can be ignored by the component or filtered out (by XSLT for example).

```
<JobGroup>
  <Id>workflow1</Id>
  <State>Active</State>
  <Name>ShuttleTakeoff</Name>
  <JobDefaults>
    <StagedTime>1051557859</StagedTime>
    <SubmitHost>asteroid.lbl.gov</SubmitHost>
    <SubmitTime>1051556734</SubmitTime>
    <Project>GrandChallenge18</Project>
    <GlobalUser>C=US,O=LBNL,CN=Keith Jackson</GlobalUser>
    <User>keith</User>
    <Environment>
      <Variable name="LD_LIBRARY_PATH">/usr/lib</Variable>
      <Variable name="PATH">/usr/bin:~/bin:</Variable>
    <Environment>
  </JobDefaults>
  <Job>
    <Id>fr15n05.1234.0</Id>
    <Name>Launch Vector Initialization</Name>
    <Executable>/usr/local/gridphys/bin/lvcalc</Executable>
    <Queue>batch</Queue>
    <State>Completed</State>
    <Machine>SMP2.emsl.pnl.gov</Machine>
    <StartTime>1051557713</StartTime>
    <EndTime>1051558868</EndTime>
    <Quote>http://www.pnl.gov/SMP2#654321</Quote>
    <Charge units="USD">12.75</Charge>
    <Requested>
      <Duration>3600</Duration>
      <Processors>2</Processors>
      <Memory>1024</Memory>
    </Requested>
    <Delivered>
      <Duration>1155</Duration>
      <Processors consumptionRate="0.78">2</Processors>
      <Memory metric="Max">975</Memory>
    </Delivered>
    <TaskGroup>
      <TaskCount>2</TaskCount>
      <TaskDistribution type="TasksPerNode">1</TaskDistribution>
      <Task>
        <Node>node1</Node>
        <Process>99353</Process>
      </Task>
      <Task>
        <Node>node12</Node>
        <Process>80209</Process>
      </Task>
    </TaskGroup>
  </Job>
  <Job>
    <Id>fr15n05.1234.1</Id>
    <Name>3-Phase Ascension</Name>
    <Queue>batch_normal</Queue>
    <State>Idle</State>
    <Machine>Colony.emsl.pnl.gov</Machine>
    <Priority>1032847</Priority>
    <Hold>System</Hold>
    <StatusMessage>Insufficient funds to start job</StatusMessage>
    <Requested>
```

```
      <Duration>43200</Duration>
    </Requested>
    <TaskGroup>
      <TaskCount>1</TaskCount>
      <Name>Master</Name>
      <Executable>/usr/local/bin/stage-coordinator</Executable>
      <Memory>2048<Memory>
      <Resource name="License" type="ESSL2">1</Resource>
      <Feature>Jumbo-Frame</Feature>
    </TaskGroup>
    <TaskGroup>
      <Name>Slave</Name>
      <TaskDistribution type="Rule">RoundRobin</TaskDistribution>
      <Executable>/usr/local/bin/stage-slave</Executable>
      <NodeCount>4</NodeCount>
      <Requested>
        <Processors group="-1">12</Processors>
        <Processors conj="Or" group="1">16</Processors>
        <Memory>512</Memory>
        <Node aggregation="Pattern">fr15n.*</Node>
      </Requested>
    </TaskGroup>
  </Job>
</JobGroup>
```

2.0 Conventions Used in This Document

2.1 Keywords

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC2119.

2.2 Table Column Interpretations

The columns of the property tables in this document have the following meanings:

Element Name	Name of the XML element (xsd;element) see [DATATYPES]
Type	Data type defined by xsd (XML Schema Definition)as: • String — xsd:string (a finite length sequence of printable characters) • Integer — xsd:integer (a signed finite length sequence of decimal digits) • Float — xsd:float (single-precision 32-bit floating point) • Boolean — xsd:boolean (consists of the literals "true" or "false") • DateTime — xsd:int (a 32-bit unsigned long in GMT seconds since the EPOCH) • Duration — xsd:int (a 32-bit unsigned long measured in seconds)
Description	Brief description of the meaning of the property

Element Name	Name of the XML element (xsd;element) see [DATATYPES]
Appearance	An indication of whether the given property must appear in the parent element. It assumes the following meanings: • MUST — This property is REQUIRED when the parent is specified • SHOULD — This property is RECOMMENDED when the parent is specified. • MAY — This property is OPTIONAL when the parent is specified.
Compliance	An indication of the relative importance of supporting the given property. • MUST — A compliant implementation MUST support this property. • SHOULD — A compliant implementation SHOULD support this property. • MAY — A compliant implementation MAY support this property.
Categories	Some properties may be categorized into one of several categories. Letters in this column indicate that the given property can be classified in the following property categories. • R — This property can be encompassed in a Requested element. • D — This property can be encompassed in a Delivered element.

2.3 Element Syntax Cardinality

Selected elements in the element syntax sections use regular expression wildcards with the following meanings:

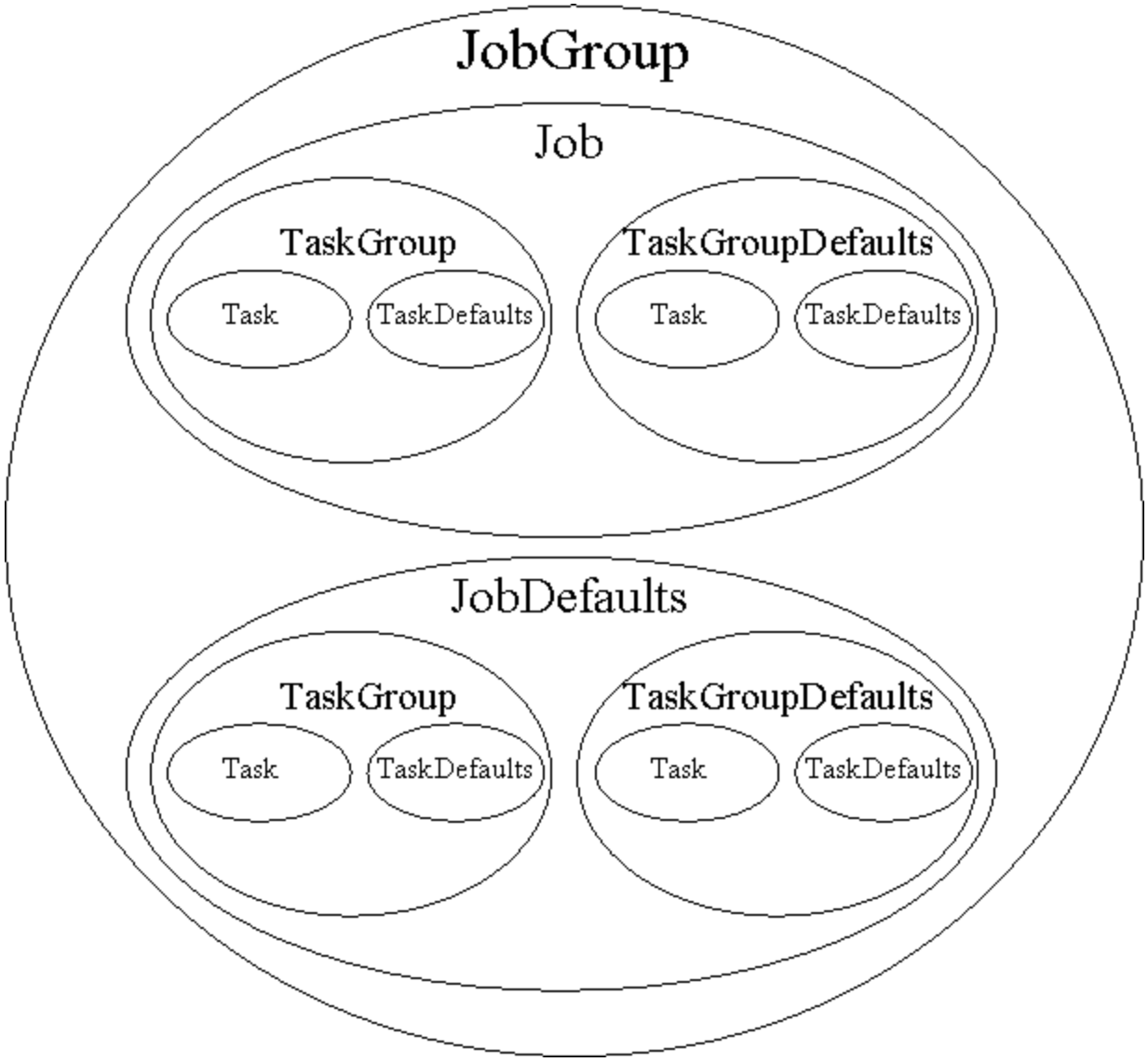
Wildcard	Description
*	Zero or more occurrences
+	One or more occurrences
?	Zero or one occurrences

The absence of one of these symbols implies exactly one occurrence.

3.0 The Job Model

The primary object within the job model is a job. A job can be thought of as a single schedulable entity and will be the object normally seen in job queues.

Image 25-1: JobGroup contains Job and JobDefaults, which contain TaskGroup and TaskGroupDefaults



Jobs with dependencies on other jobs may be submitted in a job group. Jobs within a job group form a DAG (directed acyclic graph) where the nodes are jobs and the edges represent dependencies on the status of previous jobs. A job group will consist of at least one job. A job group can optionally specify job defaults which are a set of job properties to be assumed by all jobs within the job group unless overridden within the job.

A job may consist of multiple tasks, which are the finest grained work unit and represent an endpoint for executing a given process instance. For example, a job that requests 3 nodes and 4 processors will have 4 tasks, two on one node and one on each of two nodes. Tasks may be grouped into task groups, which are logical aggregations of tasks and their common properties. Submit filters, prologs, epilogs, notification scripts, etc. run once only for each job. Whereas task groups function as logical descriptions of tasks and their properties, they also describe the number of such tasks and the nodes that they run on. As an example, a master task group (consisting of a single task) might ask for a node with a MATLAB license,

2GB of memory and an Internet connected network adapter while a slave task group (consisting of 12 tasks) could be targeted for nodes with more CPU bandwidth -- all within the same job and utilizing a common MPI ring. Tasks (and hence taskgroups) can have different executables or environments, specify different consumable resources or node properties. A job, therefore, may specify one or more task group. A job that does not specify an explicit task group is considered as having a single implicit task group. A job can optionally specify task group defaults which are a set of task group properties to be assumed by all task groups within the job unless overridden within a task group.

A task group may specify one or more tasks. A task group that does not specify an explicit task is considered as having a single implicit task. A task group can optionally specify task defaults which are a set of task properties to be assumed by all tasks within the task group unless overridden within a task.

4.0 JobGroup Element

A JobGroup is an optional element that aggregates one or more interdependent jobs. Some resource managers support the submission of job groups (multi-step jobs) and queries on the status of an entire job group.

- A compliant implementation MAY support this element.
- A JobGroup MUST specify one or more JobGroup Properties.
- A JobGroup MUST contain one or more Jobs.
- A JobGroup MAY contain zero or more JobsDefaults.

The following illustrates this element's syntax:

```
C<JobGroup>
  <!-- JobGroup Properties -->+
  <Job/>+
  <JobDefaults/>?
</JobGroup>
```

4.1 JobGroup Properties

JobGroup Properties are properties that apply to the job group as a whole. These include the job group id, jobs and job defaults, and other simple optional job properties.

Simple JobGroup Properties

Simple (unstructured) job group properties are enumerated in the table below.

Table 25-1: Simple JobGroup Properties

Element Name	Type	Description	Appearance	Compliance
CreationTime	DateTime	Date and time that the job group was instantiated	MAY	MAY

Element Name	Type	Description	Appearance	Compliance
Description	String	Description of the job group	MAY	MAY
Id	String	Job group identifier	MUST	MUST
Name	String	Name of the job group	MAY	SHOULD
State	String	State of the job group as a whole. Valid states may include NotQueued, Unstarted, Active, and Completed.	MAY	SHOULD

Job

A job group MUST contain one or more jobs.

See the next section for element details.

JobDefaults

A job group MAY contain zero or one job defaults.

See the next section for element details.

4.2 JobGroup Reference

When a simple reference to a predefined job group is needed in an encapsulating element, a JobGroup element is used with the text content being the job group id:

```
<JobGroup> workflow1</JobGroup>
```

5.0 Job and JobDefaults Element

The Job and JobDefaults elements are of the same structure. A Job element encapsulates a job and may be expressed as a standalone object. A JobDefaults element may only appear within a JobGroup and represents the defaults to be taken by all jobs within the job group. Job properties in Job elements override any properties found in a sibling JobDefaults element.

- A compliant implementation MUST support the Job element.
- A compliant implementation MAY support the JobDefaults element only if it supports the JobGroup element.
- A job MUST specify one or more Job Properties.
- One or more TaskGroup elements MAY appear at this level.
- Zero or one TaskGroupDefaults elements MAY appear at this level.

The following illustrates this element's syntax:

```
<Job>
  <!-- Job Properties -->+
  <TaskGroup/>*
  <TaskGroupDefaults/>?
</Job>
```

5.1 Job Properties

Job Properties apply to a particular job or as default properties to all jobs. They include the job id, job credentials, task groups, task group defaults, and other simple optional properties.

Simple Job Properties

Simple (unstructured) job properties are enumerated in the table below.

Table 25-2: Simple Job Properties

Element Name	Type	Description	Appearance	Compliance	Categories
Application	String	Type of application such as Gaussian or Nwchem	MAY	MAY	
Architecture	String	Type architecture for the nodes on which this job must run	MAY	MAY	RD
Arguments	String	The arguments for the executable	MAY	SHOULD	
Charge	Float	The amount charged for the job	MAY	SHOULD	
Checkpointable	Boolean	Can this job be checkpointed?	MAY	MAY	
CpuDuration	Duration	Number of cpu seconds used by the job	MAY	SHOULD	

Element Name	Type	Description	Appearance	Compliance	Categories
DeadlineTime	DateTime	Date and time that a job must end by	MAY	MAY	
EligibleTime	DateTime	Date and time that a job must start after	MAY	MAY	
EndTime	DateTime	Date and time that a job ended (independent of success or failure)	MAY	MUST	
Executable	String	Executable. This may be an absolute or relative path or a URI.*	MAY	MUST	
ExitCode	Integer	Exit code for the job	MAY	SHOULD	
GlobalJob	String	Globally unique job identifier (possibly in the form of a URI)	MAY	SHOULD	
Hold	String	Hold(s) on the job. There may be multiple instances of this element if there is more than one ld on the job	MAY	SHOULD	
InitialWorking-Directory	String	Initial working directory	MAY	SHOULD	
Interactive	Boolean	Is this an interactive job?	MAY	SHOULD	

Element Name	Type	Description	Appearance	Compliance	Categories
Id	String	A local job identifier assigned to the job by the local resource manager	MUST	MUST	
Name	String	Name of the job	MAY	SHOULD	
State	String	State of the job. Valid states may include Idle, Hold, Running, Suspended, or Completed	MAY	MUST	
Type	String	Type of job. Meaning of this extension property is context specific.	MAY	MAY	
Machine	String	Name of the system or cluster that runs the job	MAY	MUST	RD
Network	String	Type of network adapter required by the job	MAY	MAY	RD
NodeCount	Integer	Number of nodes used by the job	MAY	MUST	RD
OperatingSystem	String	Operating System required by the job	MAY	MAY	RD

Element Name	Type	Description	Appearance	Compliance	Categories
Partition	String	Name of the partition in which the job should run	MAY	MAY	RD
Priority	Integer	Current queue priority (or rank)for the job	MAY	SHOULD	
QualityOfService	String	Name of the Quality of Service (QoS)	MAY	SHOULD	RD
Queue	String	Name of the Queue (or class) that the job runs in	MAY	SHOULD	RD
Quote	String	Identifier for a guaranteed charge rate quote obtained by the job	MAY	MAY	
Reservation	String	Identifier for a reservation used by the job	MAY	MAY	RD
ReservationTime	DateTime	Date and time that a reservation was placed for the job	MAY	MAY	
ResourceManagerType	String	Type of resource manager required to run this job	MAY	MAY	RD
Restartable	Boolean	Can this job be restarted?	MAY	MAY	

Element Name	Type	Description	Appearance	Compliance	Categories
Shell	String	Specified the shell necessary to interpret the job script	MAY	MAY	
StagedTime	DateTime	Date and time that a job was staged to the local resource management system	MAY	MAY	
StartCount	Integer	Number of times the scheduler tried to start the job	MAY	MAY	
StartTime	DateTime	Date and time that the job started	MAY	MUST	
StatusMessage	String	Natural language message that can be used to provide detail on why a job failed, isn't running, etc.	MAY	SHOULD	
SubmitTIme	DateTime	Date and time that a job was submitted	MAY	SHOULD	
SubmitHost	String	FQDN of host where the job was submitted from	MAY	SHOULD	
Suspendable	Boolean	Can this job be suspended?	MAY	MAY	

Element Name	Type	Description	Appearance	Compliance	Categories
SuspendDuration	Integer	Number of seconds the job was in the Suspended state	MAY	MAY	
TimeCategory	String	This allows the specification of shifts like PrimeTime for charging purposes	MAY	MAY	
Duration	Duration	Number of seconds in the Running state	SHOULD	MUST	RD

* The Executable may be a script or a binary executable. If it is already on the target system it may be referenced by an absolute or relative pathname (relative to InitialWorkingDirectory). If it is passed with the job in a File object (see SSSRMAP), it can be referenced by an absolute or relative URI. An absolute URI would specify a URL where the file can be downloaded (like with wget). A relative URI is specified by preceding an identifier by a pound sign, as in

```
<Executable>#Script</Executable>
```

It will be found in a File object included along with the Job object with the Script as an identifier, as in

```
<File id="Script">echo hello world</File>
```

Feature Element

The Feature element connotes an arbitrary named feature of a node.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or one times within a given set of Job Properties.
- This element is of type String.
- This element MAY have an `aggregation` attribute of type String that provides a way to indicate multiple values with a single expression. A compliant implementation MAY support the `aggregation` attribute if the Feature element is supported. Possible values for this attribute include:
    - List — a comma-separated list of features
    - Pattern — a regular expression (perl5) matching desired features

- If an `aggregation` attribute is specified with the value of List, this element MAY also have a `delimiter` attribute of type String that indicates what delimiter is used to separate list elements. The default list delimiter is a comma.
- This element MAY be categorized as a requested or delivered property by being encompassed by the appropriate element.

The following is an example of a feature element:

```
<Feature aggregation="List">feature1,feature2</Feature>
```

OutputFile Element

The `OutputFile` element specifies the name of the file to which the output stream (stdout) from the job will be written.

- This element's character content is the name of the file. If this element is omitted or it is empty, then an appropriate output file is auto-determined by the queuing system.
- This element MAY have a `redirectList` attribute which is a comma-separated list of output redirection attributes of type String. A compliant implementation SHOULD support this attribute if `OutputFile` is supported. Possible values for this attribute include:
    - Append — opens the output file for append
    - Close — closes and discards the output stream
    - Flush — output is written to output file as it is generated
    - Keep — leave the output file on the execution host
    - Merge — merges the output stream into the error stream

Note that when using the `redirectList` attributes, the cumulative effect of the `ErrorFile` and `OutputFile` directives may be order dependent.

The following is an example of an `OutputFile` element:

```
<OutputFile redirectList="Append">~/myjob.out</OutputFile>
```

ErrorFile Element

The `ErrorFile` element specifies the name of the file to which the error stream (stderr) from the job will be written.

- This element's character content is the name of the file. If this element is omitted or it is empty, then an appropriate error file is auto-determined by the queuing system.
- This element MAY have a `redirectList` attribute which is a comma-separated list of error redirection attributes of type String. A compliant implementation SHOULD support this attribute if `ErrorFile` is supported. Possible values for this attribute include:
    - Close — closes and discards the error stream
    - Append — opens the error file for append

- Flush — output is written to output file as it is generated
- Keep — leave the output file on the execution host
- Merge — merges the error stream into the output stream

Note that when using the `redirectList` attributes, the cumulative effect of the `ErrorFile` and `OutputFile` directives may be order dependent.

The following is an example of an `ErrorFile` element:

```
<ErrorFile redirectList="Merge"></ErrorFile>
```

InputFile Element

The `InputFile` element specifies the name of the file from which the input stream (stdin) for the job will be read.

- This element's character content is the name of the file. If this element is omitted or it is empty, then an appropriate input file is auto-determined by the queuing system.
- This element MAY have a `redirectList` attribute which is a comma-separated list of input attributes of type String. A compliant implementation SHOULD support this attribute if `InputFile` is supported. Possible values for this attribute include:
  - Close — closes and discards the input stream

The following is an example of an `InputFile` element:

```
<InputFile redirectList="Close"></InputFile>
```

NotificationList Element

The `NotificationList` element specifies the job-related events or conditions for which a notification will be sent.

- This element's character content is a comma-separated list of events or conditions for which a notification should be sent. Possible values for the elements of this list include:
  - JobStart — send a notification when the job starts
  - JobEnd — send a notification when the job ends
  - All — send notifications for all notifiable events
  - None — do not send notifications for any events
- This element MAY have a `uri` attribute of type String which indicates where the notification is to be sent. A compliant implementation MAY support this attribute if `NotificationList` is supported. The `uri` is in the format: `[scheme://]authority` with the scheme being smtp and the authority being an email address by default.

The following is an example of a `NotificationList` element:

```
<NotificationList uri="smith@business.com">JobStart,JobEnd</NotificationList>
```

ResourceLimitElement

The `ResourceLimit` element represents a resource limit with its name and value.

- This element MUST have a `name` attribute of type String. A compliant implementation MUST support the name attribute if ResourceLimit is supported.
- This element MAY have a `type` attribute of type String that may have the values `Hard` or `Soft`. If the limit is enforced by the operating system, a hard limit is one that cannot be increased once it is set while a soft limit may be increased up to the value of the hard limit. If the type attribute is omitted, both the soft and hard limits are set.
- This element's character content is the resource limit's value.

Some typical names include:

Name	Description
CoreFileSize	Maximum core file size
CpuTime	CPU time in seconds
DataSegSize	Maximum data size
FileSize	Maximum file size
MaxMemorySize	Maximum resident set size
MaxProcesses	Maximum number of processes
MaxSwap	Virtual memory limit
MaxMemLock	Maximum locked-in-memory address space
MaxProcessors	Maximum processors
MaxMemory	Maximum memory
MaxDisk	Maximum disk space
MaxNetwork	Maximum network bandwidth
MaxFileIO	Maximum file i/o

Name	Description
OpenFiles	Maximum number of open files
Stacksize	Maximum stack size

The following is an example of a `ResourceLimit` element:

```
<ResourceLimit name="CPUTime">1000000</ResourceLimit>
```

Credentials

Credentials are a special group of job properties that characterize an authenticated token or id. They can be categorized in both requested and delivered forms.

Credential job properties are enumerated in the table below.

Table 25-3: Credential Job Properties

Element Name	Type	Description	Appearance	Compliance	Categories
Project	String	Name of the Project or Charge Account	MAY	SHOULD	RD
GlobalUser	String	Globally unique user identifier. This may be an X.509 DN for example	MAY	SHOULD	RD
Group	String	Name of the local group id	MAY	MAY	RD
User	String	Name of the local user id for the job	MAY	MUST	RD

Environment Element

The Environment element encapsulates environment variables.

- This element MAY have an export attribute of type Boolean that which if set to `True` indicates that all environment variables in the context of the job submission process should be exported in the job's execution environment.
- A compliant implementation SHOULD support this element.
- An Environment element MAY appear zero or one times within a given set of Job (or TaskGroup)

Properties.

- An Environment element MAY contain one or more Variable elements.

The following illustrates this element's syntax:

```
<Environment>
  <Variable/>+
</Environment>
```

Variable Element

The Variable element represents an environment variable with its name and value.

This element MUST have a `name` attribute of type String. A compliant implementation MUST support the name attribute if Variable is supported. This element's character content is the environment variable's value.

The following is an example of a Variable element:

```
<Variable name="PATH">/usr/bin:/home/sssdemo</Variable>
```

Node Element

The Node element represents a node.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or one times within a given set of Job Properties.
- This element is of type String.
- This element MAY have an `aggregation` attribute of type String that provides a way to indicate multiple values with a single expression. A compliant implementation MAY support the `aggregation` attribute if the Feature element is supported. Possible values for this attribute include:
    - List - a comma-separated list of features
    - Pattern - a regular expression (perl5) matching desired features
    - Range - a range of nodes of the form: <prefix>[5-23,77]
- If an `aggregation` attribute is specified with the value of List, this element MAY also have a `delimiter` attribute of type String that indicates what delimiter is used to separate list elements. The default list delimiter is a comma.
- This element MAY have a `count` attribute of type Integer that indicates the instance count of the specified node(s).
- This element MAY be categorized as a requested or delivered property by being encompassed by the appropriate element.

The following is an example of a Node element:

```
<Node aggregation="Pattern">node[1-5]</Node>
```

TaskDistribution Element

The `TaskDistribution` element describes how tasks are to be mapped to nodes. This mapping may be expressed as a rule name, a task per node ratio or an arbitrary geometry.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or one times in a given set of Job (or TaskGroup) Properties.
- This element is of type String.
- This element MAY have a `type` attribute of type String that provides a hint as to the type of mapping guidance provided. It may have values including `Rule`, `TasksPerNode`, `ProcessorsPerTask` or `Geometry`. A compliant implementation MAY support the `type` attribute if the `TaskDistribution` element is supported.
- It is possible to use `Processors`, `NodeCount` and `TaskCount` elements to specify a set of mutually contradictory task parameters. When this occurs, components are responsible for resolving conflicting requirements.

The following are three examples of a `TaskDistribution` element:

```
<TaskDistribution type="TasksPerNode">2</TaskDistribution>
<TaskDistribution type="Rule">RoundRobin</TaskDistribution>
<TaskDistribution type="Geometry">{1,4}{2}{3,5}</TaskDistribution>
```

Dependency Element

The Dependency element allows a job's execution to depend on the status of other jobs. In a job group (multi-step job), some jobs may delay execution until the failure or success of other jobs creating in general a Directed Acyclic Graph relationship between the jobs. This element's content is of type String and represents the job that the current job is dependent upon. Since a job may have two or more dependencies, this element may appear more than once in a given job scope. A compliant implementation SHOULD support this element if job groups are supported.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or more times in a given set of Job (or TaskGroup) Properties.
- This element is of type String and contains the JobId that the current job is dependent upon.
- This element MAY have a `condition` attribute of type String that indicates the basis for determining when the current job executes in relation to the specified job. A compliant implementation MUST support this attribute if this element is supported.
  Possible values for this attribute include:
  - OnSuccess this job should run after the referenced job only if it completes successfully (this is the default if the `type` attribute is omitted)
  - OnFailure this job should run after the referenced job only if it fails
  - OnExit this job should run after the referenced job exits

- If the `condition` attribute is equal to OnExit, this element MAY have a `code` attribute of type Integer that indicates the exit code that will trigger this job to run. If the code attribute is omitted, then the current job should run after the referenced job for any exit status.
- This element MAY have a `designator` attribute of type String that indicates that indicates the property of the job that identifies it as the dependent job. A compliant implementation MAY support this attribute if this element is supported.
  Possible values for this attribute include:
  - JobId the job this job is dependent upon is specified by JobId (this is the default if the designator attribute is omitted)
  - JobName the job(s) this job is dependent upon are specified by JobName

The following is an example of a Dependency element:

```
<Dependency condition=”OnSuccess” designator=”JobId”>PBS.1234.0</Dependency>
```

Consumable Resources

Consumable Resources are a special group of properties that can have additional attributes and can be used in multiple contexts. In general a consumable resource is a resource that can be consumed in a measurable quantity.

- A consumable resource MAY have a `context` attribute of type String that indicates the sense in which the resource is used. A compliant implementation MAY support this attribute. Possible values for this attribute include:
  - Configured — run this task only on nodes having the specified configured resources
  - Available — run this task only on nodes having the specified available resources. (this is the default if the `context` attribute is omitted)
  - Used — the task used the indicated resources (this is analogous to being including in a Delivered block)
  - Dedicated — the indicated amount of the resource should be dedicated to the task
- A consumable resource MAY have a `units` attribute that is of type String that specifies the units by which it is being measured. If this attribute is omitted, a default unit is implied. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a `metric` attribute that is of type String that specifies the type of measurement being described. For example, the measurement may be a Total, an Average, a Min or a Max. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a `duration` attribute of type Duration that indicates the amount of time for which that resource was used. This need only be specified if the resource was used for a different amount of time than the duration for the job. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a `consumptionRate` attribute of type Float that indicates the average percentage that a resource was used over its duration. For example, an overbooked SMP

running 100 jobs across 32 processors may wish to scale the usage and charge by the average fraction of processor usage actually delivered. A compliant implementation MAY support this attribute if the element is supported.

- A consumable resource MAY have a `dynamic` attribute of type Boolean that indicates whether the resource allocated for this job should be allowed to grow or shrink dynamically. For example, if processors is specified with dynamic equal to True, the job may be dynamically allocated more processors as they become available. The growth bounds can be indicated via the `op` attribute which is inherited when a consumable resource element is encapsulated within a *Requested* element. A compliant implementation MAY support this attribute if the element is supported.

A list of simple consumable resources is listed in the table below.

Table 25-4: Simple Consumable Resources

Element Name	Type	Description	Appearance	Compliance	Categories
Disk	Float	Amount of disk	MAY	SHOULD	RD
Memory	Float	Amount of memory	MAY	SHOULD	RD
Network	Float	Amount of network	MAY	MAY	RD
Processors	Integer	Number of processors	MAY	MUST	RD
Swap	Float	Amount of virtual memory	MAY	MAY	RD

The following are two examples for specifying a consumable resource:

```
<Memory metric="Max" units="GB">483</Memory>
<Processors duration="1234" consumptionRate="0.63">4</Processors>
```

Resource Element

In addition to the consumable resources enumerated in the above table, an extensible consumable resource is defined by the Resource element.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or more times within a given set of job (or task group) properties.
- Like the other consumable resources, this property MAY be categorized as a requested or delivered property by being encompassed in the appropriate element.
- This element is of type Float.

- This element shares the *same properties and attributes as the other consumable resources* but it requires an additional `name` (and optional type) attribute to describe it.
- It MUST have a `name` attribute of type String that indicates the type of consumable resource being measured. A compliant implementation MUST support this attribute if the element is supported.
- It MAY have a `type` attribute of type String that distinguishes it within a general resource class. A compliant implementation SHOULD support this attribute if the element is supported.

The following are two examples for specifying a Resource element:

```
<Resource name="License" type="MATLAB">1</Resource>
<Resource name="Telescope" type="Zoom2000" duration="750" metric="KX">10</Resource>
```

Extension Element

The Extension element provides a means to pass extensible properties with the job object.

Some applications may find it easier to use a named extension property than discover and handle elements they do not understand or anticipate by name.

- A compliant implementation MAY support this element.
- This element MUST have a `name` attribute of type String that gives the extension property's name. A compliant implementation MUST support this attribute if this element is supported.
- This element MAY have a `type` attribute of type String that characterizes the context within which the property should be understood. A compliant implementation SHOULD support this attribute if this element is supported.
- This element's character content, which is of type String, is the extension property's value.

The following is an example of an Extension element:

```
<Extension type="Scheduler" name="Restartable">true</Extension>
```

TaskGroup

A job MAY specify one or more task groups.

See the next section for element details.

TaskGroupDefaults

A job MAY specify zero or more task group defaults.

See the next section for element details.

5.2 Job Reference

When a simple reference to a predefined job is needed in an encapsulating element, a Job element is used with the text content being the job id:

```
<Job> job123</Job>
```

6.0 TaskGroup and TaskGroupDefaults Element

The `TaskGroup` and `TaskGroupDefaults` elements have the same structure. A `TaskGroup` element aggregates tasks. A TaskGroupDefaults element may only appear within a Job (or JobDefaults) and represents the defaults to be taken by all task groups within the job. Task group properties in `TaskGroup` elements override any properties found in a sibling `TaskGroupDefaults` element.

- A compliant implementation MAY support the `TaskGroup` element.
- A compliant implementation MAY support the `TaskGroupDefaults` element.
- A task group MUST specify one or more TaskGroup Properties.
- One or more Task elements MAY appear at this level.
- Zero or one TaskDefaults elements MAY appear at this level.

The following illustrates this element's syntax:

```
<TaskGroup>
  <!-- TaskGroup Properties -->+
  <!-- Job Properties -->*
  <Task>+
  <TaskDefaults>?
</TaskGroup>
```

6.1 TaskGroup Properties

TaskGroup Properties apply to a particular task group or as default properties to encompassed task groups. These properties include the task group id, its tasks, task defaults, and other simple task group properties.

Simple TaskGroup Properties

Simple (unstructured) task group properties are enumerated in Table 6.

Table 25-5: Simple TaskGroup Properties

Element Name	Type	Description	Appearance	Compliance	Categories
TaskCount	Integer	Number of tasks in this taskgroup	MAY	MUST	
Id	String	A task group identifier unique within the job	MAY	MAY	

Element Name	Type	Description	Appearance	Compliance	Categories
Name	String	A task group name (such as Master)	MAY	SHOULD	

Task

A task group MAY specify zero or more tasks.

See the next section for element details.

TaskDefaults

A task group MAY specify zero or more task defaults.

See the next section for element details.

6.2 TaskGroup Reference

When a simple reference to a predefined task group is needed in an encapsulating element, a TaskGroup element is used with the text content being the task group id:

```
<TaskGroup> tg1</TaskGroup>
```

7.0 Task and TaskDefaults Element

The `Task` and `TaskDefaults` elements have the same structure. A `Task` element contains information specific to a task (like the process id or the host it ran on). A `TaskDefaults` element may only appear within a `TaskGroup` (or `TaskGroupDefaults`) element and represents the defaults for all tasks within the task group. Task properties in `Task` elements override any properties found in a sibling `TaskDefaults` element.

- A compliant implementation MAY support the `TaskGroup` element.
- A compliant implementation MAY support the `TaskGroupDefaults` element.
- A task group MUST specify one or more `TaskGroup` Properties.
- One or more `Task` elements MAY appear at this level.
- Zero or one `TaskDefaults` elements MAY appear at this level.

The following illustrates this element's syntax:

```
<Task>
  <!-- Task Properties -->+
  <!-- Job Properties -->*
</Task>
```

7.1 Task Properties

Task Properties are properties that apply to a particular task or as default properties to encompassed tasks. These properties include the task id and other task properties.

Simple Task Properties

Simple (unstructured) task properties are enumerated in the table below.

Table 25-6: Simple Task Properties

Element Name	Type	Description	Appearance	Compliance	Categories
Node	String	Name of the node this task ran on	MAY	MUST	
Session	Integer	Session id for the task group or job	MAY	MAY	
Id	String	A task identifier unique within the taskgroup	MAY	MAY	

7.2 Task Reference

When a simple reference to a predefined task is needed in an encapsulating element, a `Task` element is used with the text content being the task id:

```
<Task>1</Task>
```

8.0 Property Categories

Certain properties need to be classified as being in a particular category. This is done when it is necessary to distinguish between a property that is requested versus a property that was delivered. When no such distinction is necessary, it is recommended that the property not be enveloped in one of these elements. In general, a property should be enveloped in a category element only if it is expected that the property will need to be attributed to more than one property category, or if it needs to make use of some of the special attributes inherited from the category.

8.1 Requested Element

A requested property reflects properties as they were requested. A disparity might occur between the requested value and the value delivered if a preference was expressed, if multiple options were specified, or if ranges or pattern matching was specified.

- A compliant implementation SHOULD support this element.

The following illustrates the syntax of this element:

```
<Requested>
  <!-- Requested Properties -->+
</Requested>
```

The following describes the attributes and elements for the example above:

`/Requested`

This element is used to encapsulate requested properties.

`/Requested/<Requested Property>`

Requested properties appear at this level.

Requested Properties inherit some additional attributes.

- A requested property MAY have an `op` attribute of type String that indicates a conditional operation on the value. A compliant implementation SHOULD support this attribute. Valid values for the `op` attribute include EQ meaning equals (which is the default), NE meaning not equal, LT meaning less than, GT meaning greater than, LE meaning less than or equal to, GE meaning greater than or equal to, Match which implies the value is a pattern to be matched.
- A requested property MAY have a `conj` attribute of type String that indicates a conjunctive relationship with the previous element. A compliant implementation MAY support this attribute. Valid values for the conj attribute include And (which is the default), Or, Nand meaning and not, and Nor meaning or not.
- A requested property MAY have a `group` attribute of type Integer that indicates expression grouping and operator precedence much like parenthetical groupings. A compliant implementation MAY support this attribute. A positive grouping indicates the number of nested expressions being opened with the property while a negative grouping indicates the number of nested expressions being closed with the property.
- A requested property MAY have a `preference` attribute of type Integer that indicates a preference for the property along with a weight (the weights are taken as a ratio to the sum of all weights in the same group). A compliant implementation MAY support this attribute. If a group of positive valued preference alternatives are specified, at least one of the preferences must be satisfied for the job to run. If a group of negative valued preferences are specified, the preferences will try to be met according to their weights but the job will still run even if it can't satisfy any of the preferred properties. (Weight ranking can be removed by making all weights the same value (1 or -1 for example).
- A requested property MAY have a `performanceFactor` attribute of type Float that provides a hint to the scheduler of what performance tradeoffs to make in terms of resources and start time. A compliant implementation MAY support this attribute.

The following are four examples of using Requested Properties:

```
<Requested>
```

```
  <Processors op="GE">8</Processors>
  <Processors op="LE">16</Processors>
  <Duration>3600</Duration>
</Requested>
<Requested>
  <NodeCount>1</NodeCount>
  <Node aggregation="Pattern">fr15.*</Node>
<Requested>
<Requested>
  <User group="1">scottmo</User>
  <Account group="-1">mscfops</Account>
  <User conj="Or" group="1">amy</User>
  <Account group="-1">chemistry</Account>
</Requested>
<Requested>
  <Memory preference="2">1024</Memory>
  <Memory preference="1">512</Memory>
</Requested>
```

8.2 Delivered Element

A delivered property reflects properties as they were actually utilized, realized or consumed. It reflects the actual amounts or values that are used, as opposed to a limit, choice or pattern as may be the case with a requested property.

- A compliant implementation SHOULD support this element.

The following illustrates the syntax of this element:

```
<Delivered>
  <!-- Delivered Properties -->+
</Delivered>
```

The following describes the attributes and elements for the example above:

`/Delivered`

This element is used to encapsulate delivered properties.

`/Delivered/<Delivered Property>`

Delivered properties appear at this level.

Delivered Properties inherit some additional attributes.

- A delivered property MAY have a `group` attribute of type Integer that indicates expression grouping and operator precedence much like parenthetical groupings. A compliant implementation MAY support this attribute. A positive grouping indicates the number of nested expressions being opened with the property while a negative grouping indicates the number of nested expressions being closed with the property. The purpose of this attribute would be to logically group delivered properties if they were used in certain aggregations (like a job that spanned machines).

The following are the same four examples distinguishing the delivered amounts and values:

```
<Delivered>
```

```
  <Processors>12</Processors>
  <Duration>1234</Duration>
</Delivered>
<Delivered>
  <Node>fr15n03</Node>
</Delivered>
<Delivered>
  <User>scottmo</User>
  <Account>mscfops</Account>
</Delivered>
<Delivered>
  <Memory>1024</Memory>
</Delivered>
```

9.0 AwarenessPolicy Attribute

A word or two should be said about compatibility mechanisms. With all the leeway in the specification with regard to implementing various portions of the specification, problems might arise if an implementation simply ignores a portion of a job specification that is critical to the job function in certain contexts. Given this situation, it might be desirable in some circumstances for jobs to be rejected by sites that fail to fully support that job's element or attributes. At other times, it might be desirable for a job to run, using a best-effort approach to supporting unimplemented features. Consequently, we define an `awarenessPolicy` attribute which can be added as an optional attribute to the Job element or any other containment or property element to indicate how the property (or the default action for the elements that the containment element encloses) must react when the implementation does not understand an element or attribute.

An awareness policy of `Reject` will cause the server to return a failure if it receives a client request in which it does not support an associated element name or attribute name or value. It is reasonable for an implementation to ignore (not even look for) an element or attribute that would not be critical to its function as long as ignoring this attribute or element would not cause an incorrect result. However, any element or attribute that was present that would be expected to be handled in a manner that the implementation does not support must result in a failure.

An awareness policy of `Warn` will accept the misunderstood element or attribute and continue to process the job object on a best effort basis. However a warning MUST be sent (if possible) to the requestor enumerating the elements and attributes that are not understood.

An awareness policy of `Ignore` will accept the unsupported element or attribute and continue to process the job object on a best effort basis. The action could be to simply ignore the attribute.

- This name of this attribute is `awarenessPolicy`.
- This attribute is of type String.
- This attribute can have values of `Reject`, `Warn` or `Ignore`.
- A compliant implementation MAY support this attribute.

- An implementation that does not support an attribute MUST reject any job object which contains elements or attributes that it does not support. Furthermore, it SHOULD return a message to the requestor with an indication of the element or attribute name it did not understand.
- This attribute MAY be present in a property or containment element.
- If an implementation does support the attribute, but it is absent, the default value of `Reject` is implied.
- Individual elements in the job object may override the containing object's awareness policy default by including this attribute. For example, a job might specify an `awarenessPolicy` of Reject at its root (the Job element) but may want to allow a particular subset of elements or attributes to be ignored if not understood. Conversely, a job with a default awarenessPolicy of Ignore might want to classify a subset of its optional elements as Reject if they are indispensable to its correct interpretation. An implementation can opt to check or not check for this attribute at any level it wants but must assume a Reject policy for any elements it does not check.

10.0 References

ISO 8601

ISO (International Organization for Standardization). Representations of dates and times, 1988-06-15. http://www.iso.ch/markete/8601.pdf

DATATYPES

XML Schema Part 2: Datatypes. Recommendation, 02 MAY 2001. http://www.w3.org/TR/xmlschema-2/

Appendix A

Units of Measure Abbreviations

Abbreviation	Definition	Quantity
B	byte	1 byte
KB	Kilobyte	2^10 bytes
MB	Megabyte	2^20 bytes

Abbreviation	Definition	Quantity
GB	Gigabyte	2^30 bytes
TB	Terabyte	2^40 bytes
PB	Petabyte	2^50 bytes
EB	Exabyte	2^60 bytes
ZB	Zettabyte	2^70 bytes
YB	Yottabyte	2^80 bytes
NB	Nonabyte	2^90 bytes
DB	Doggabyte	2^100 bytes

Scalable Systems Software Resource Management and Accounting Protocol (SSSRMAP) Message Format

Resource Management Interface Specs
Release v. 3.0.4
18 JUL 2005

Scott Jackson
Brett Bode
David Jackson
Kevin Walker

Status of This Memo

This is a specification defining an XML message format used between Scalable Systems Software components. It is intended that this specification will continue to evolve as these interfaces are implemented and thoroughly tested by time and experience.

Abstract

This document is a specification describing a message format for the interaction of resource management and accounting software components developed as part of the Scalable Systems Software Center. The SSSRMAP Message Format defines a request-response syntax supporting both functional and object-oriented messages. The protocol is specified in XML Schema Definition. The message elements

defined in this specification are intended to be framed within the Envelope and Body elements defined in the SSSRMAP Wire Protocol specification document.

Table of Contents

- 1.0 Introduction
- 2.0 Conventions Used in this Document
  - 2.1 Keywords
  - 2.2 XML Case Conventions
  - 2.3 Schema Definitions
- 3.0 Encoding
  - 3.1 Schema Header and Name spaces
  - 3.2 Element Descriptions
    - 3.2.1 The Request Element
    - 3.2.2 The Object Element
    - 3.2.3 The Get Element
    - 3.2.4 The Set Element
    - 3.2.5 The Where Element
    - 3.2.6 The Option Element
    - 3.2.7 The Data Element
    - 3.2.8 The File Element
    - 3.2.9 The Count Element
    - 3.2.10 The Response Element
    - 3.2.11 The Status Element
    - 3.2.12 The Value Element
    - 3.2.13 The Code Element
    - 3.2.14 The Message Element
  - 3.3 Modified XPATH Expressions
    - 3.3.1 Sample Modified XPATH expressions
  - 3.4 Examples
    - 3.4.1 Sample Requests
    - 3.4.2 Sample Responses

- 4.0 Error Reporting
- 5.0 References

1.0 Introduction

A major objective of the Scalable Systems Software [SSS] Center is to create a scalable and modular infrastructure for resource management and accounting on terascale clusters including resource scheduling, node daemon support, comprehensive usage accounting and user interfaces emphasizing portability to terascale vendor operating systems. Existing resource management and accounting components feature disparate APIs (Application Programming Interfaces) requiring various forms of application coding to interact with other components.

This document proposes a common message format expressed in an XML request-response syntax to be considered as the foundation of a standard for communications between and among resource management and accounting software components. In this document this standard is expressed in two levels of generality. The features of the core SSSRMAP protocol common to all resource management and accounting components in general are described in the main body of this document. The aspects of the syntax specific to individual components are described in component-specific binding documents.

2.0 Conventions Used in This Document

2.1 Keywords

The keywords “MUST”, “MUST NOT”, “REQUIIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC2119 [RFC2119].

2.2 XML Case Conventions

In order to enforce a consistent capitalization and naming convention across all SSSRMAP specifications “Upper Camel Case” (UCC) and “Lower Camel Case” (LCC) Capitalization styles shall be used. UCC style capitalizes the first character of each word and compounds the name. LCC style capitalizes the first character of each word except the first word. [XML_CONV][FED_XML]

1. SSSRMAP XML Schema and XML instance documents SHALL use the following conventions:
   - Element names SHALL be in UCC convention (example: <UpperCamelCaseElement/>.
   - Attribute names SHALL be in LCC convention (example: <UpperCamelCaseElement lowerCamelCaseAttribute=”Whatever”/>.
2. General rules for all names are:
   - Acronyms SHOULD be avoided, but in cases where they are used, the capitalization SHALL remain (example: XMLSignature).
   - Underscores (_), periods (.) and dashes (-) MUST NOT be used (example: use JobId instead of JOB.ID, Job_ID or job-id).

2.3 Schema Definitions

```
SSSRMAP Schema Definitions appear like this
```

In case of disagreement between the schema file and this specification, the schema file takes precedence.

3.0 Encoding

Encoding tells how a message is represented when exchanged. SSSRMAP data exchange messages SHALL be defined in terms of XML schema [XML_SCHEMA].

3.1 Schema Header and Name Spaces

The header of the schema definition is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:sssrmap="http://scidac.org/ScalableSystems/SSSRMAP"
  targetNamespace="http://www.scidac.org/ScalableSystems/SSSRMAP"
  elementFormDefault="qualified">
```

3.2 Element Descriptions

The following subsections describe the elements that make up SSSRMAP messages. SSSRMAP messages are transmitted in the Body and Envelope elements as described in the SSSRMAP Wire Protocol specification [WIRE_PROTOCOL].

The Request Element

The `Request` element specifies an individual request. An object-oriented request will have at least one `Object` element while a functional request will not have one. Depending on context, the `Request` element MAY contain one or more `Get` elements or one or more Set elements and any number of `Where` elements. `Option`, `Data`, `File` or `Count` elements may also be included. If a component supports it, chunking may be requested where large response data is possible. Setting the chunking attribute to "True" requests that the server break a large response into multiple chunks (each with their own envelope) so they can be processed in separate pieces.

Only an `action` attribute is required. All other attributes are optional.

Attribute	Description
action	Specifies the action or function to be performed
actor	The authenticated user sending the request
id	Uniquely maps the request to the appropriate response
chunking	Requests that segmentation be used for large response data if set to "True"
chunkSize	Requests that the segmentation size be no larger than the specified amount

```
<complexType name="RequestType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="sssrmap:Object" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:Option" minOccurs="0" maxOccurs="unbounded"/>
    <choice minOccurs="0" maxOccurs="1">
      <element ref="sssrmap:Get" minOccurs="1" maxOccurs="unbounded"/>
      <element ref="sssrmap:Set" minOccurs="1" maxOccurs="unbounded"/>
    </choice>
    <element ref="sssrmap:Where" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:Data" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:Count" minOccurs="0" maxOccurs="1"/>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </choice>
  <attribute name="action" type="string" use="required"/>
  <attribute name="actor" type="string" use="required"/>
  <attribute name="id" type="string" use="optional"/>
  <attribute name="chunking" type="sssrmap:BoolType" use="optional"/>
  <attribute name="chunkSize" type="positiveInteger" use="optional"/>
</complexType>

<element name="Request" type="sssrmap:RequestType"/>
```

The Object Element

The `Object` element is used in an object-oriented request to specify the object receiving the action. It is possible to have multiple `Object` elements in a request if an implementation supports multi-object queries.

The object class name is specified as text content. All attributes are optional.

- `join` – the type of join to be performed with the preceding object
    - A `join` attribute of “Inner” specifies an inner join. This is the default.
    - A `join` attribute of “FullOuter” specifies a full outer join.
    - A `join` attribute of “LeftOuter” specifies a left outer join.
    - A `join` attribute of “RightOuter” specifies a right outer join.
    - A `join` attribute of “Cross” specifies a cross join.
    - A `join` attribute of “Union” specifies a union join.

```
<complexType name="ObjectType">
  <simpleContent>
    <extension base="string">
      <attribute name="join" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Object" type="sssrmap:ObjectType"/>
```

The Get Element

The `Get` element is used to indicate the data fields to be returned in a query. Get is typically used within requests with `action="query"`. Multiple Get elements cause the fields to be returned in the order specified. If no `Get` elements are specified, the query will return a default set of fields.

Only a `name` attribute is required. All other attributes are optional.

Attribute	Description
name	The name of the data field to be returned. This MUST be of the form of a "Modified XPATH expression" as described in a later section.
op	The operator to be used to aggregate or perform an operation on the returned values. • An *op* attribute of "Sort" specifies an ascending sort operation • An *op* attribute of "Tros" specifies a descending sort operation • An *op* attribute of "Sum" returns the sum (only valid for numeric values) • An *op* attribute of "Max" returns the maximum value • An *op* attribute of "Min" returns the minimum value • An *op* attribute of "Count" returns the number of values • An *op* attribute of "Average" returns the average of the values • An *op* attribute of "GroupBy" signifies that aggregates are grouped by this field
object	Specifies the object for which you want the named attribute in a multi-object query.
units	The units in which to return the value (if applicable)

```
<complexType name="GetType">
  <attribute name="name" type="string" use="required"/>
  <attribute name="object" type="string" use="optional"/>
  <attribute name="op" type="sssrmap:GetOperatorType" use="optional"/>
  <attribute name="units" type="string" use="optional"/>
</complexType>

<element name="Get" type="sssrmap:GetType"/>

<simpleType name="GetOperatorType">
  <restriction base="string">
    <enumeration value="Sort"/>
    <enumeration value="Tros"/>
    <enumeration value="Count"/>
    <enumeration value="Sum"/>
    <enumeration value="Max"/>
    <enumeration value="Min"/>
    <enumeration value="Average"/>
    <enumeration value="GroupBy"/>
  </restriction>
</simpleType>
```

The Set Element

The `Set` element is used to specify the object data fields to be assigned values. Set is typically used within requests with `action="Create"` or `action="Modify"`. The use of `Get` or `Set` elements within a request is mutually exclusive.

The assignment value (to which the field is being changed) is specified as the text content. A Set element without a value may be used as an assertion flag. Only the `name` attribute is required. All other attributes are optional.

Attribute	Description
name	The name of the field being assigned a value. This MUST be of the form of a "Modified XPATH expression" as described in a later section.
op	The operator to be used in assigning a new value to the name. If an `op` attribute is not specified and a value is specified, the specified value will be assigned to the named field ("assign"). • An `op` attribute of "Assign" assigns value to the named field • An `op` attribute of "Inc" increments the named field by the value • An `op` attribute of "Dec" decrements the named field by the value
units	The units corresponding to the value being set

```
<complexType name="SetType">
  <simpleContent>
    <extension base="string">
      <attribute name="name" type="string" use="required"/>
      <attribute name="op" type="sssrmap:SetOperatorType" use="optional"/>
      <attribute name="units" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Set" type="sssrmap:SetType"/>

<simpleType name="SetOperatorType">
  <restriction base="string">
    <enumeration value="Assign"/>
    <enumeration value="Inc"/>
    <enumeration value="Dec"/>
  </restriction>
</simpleType>
```

The Where Element

A `Request` element may contain one or more `Where` elements that specify the search conditions for which objects the action is to be performed on.

The condition value (against which the field is tested) is specified as the text content. A `Where` element without a value may be used as a truth test. Only the `name` attribute is required. All other attributes are optional.

Attribute	Description
name	The name of the data field to be tested. This MUST be of the form of a "Modified XPATH expression" as described in a later section.

Attribute	Description
op	The operator to be used to test the name against the value. If an op attribute is not specified and a value is specified, the field will be tested whether it is equal to the value ("EQ"). • An `op` attribute of "EQ" specifies an equality comparison • An `op` attribute of "LT" specifies a "less than" comparison • An `op` attribute of "GT" specifies a "greater than" comparison • An `op` attribute of "LE" specifies a "less than or equal to" test • An `op` attribute of "GE" specifies a "greater than or equal to" test • An `op` attribute of "NE" specifies a "not equal to" test • An `op` attribute of "Match" specifies a regular expression matching comparison
conj	Indicates whether this test is to be ANDed or ORed with the immediately preceding where condition • A `conj` attribute of "And" specifies an "and" conjunction • A `conj` attribute of "Or" specifies an "or" condition • A `conj` attribute of "AndNot" specifies an "and not" conjunction • A `conj` attribute of "OrNot" specifies an "or not" condition
group	Indicates an increase or decrease of parentheses grouping depth • A positive number indicates the number of left parentheses to precede the condition, i.e. `group="2"` represents "((condition". • A negative number indicates the number of right parentheses to follow the condition, i.e. `group="-2"` represents "condition))".
object	Specifies the object for the first operand in a multi-object query.
subject	Specifies the object for the second operand in a multi-object query.
units	Indicates the units to be used in the value comparison

```
<complexType name="WhereType">
  <simpleContent>
    <extension base="string">
      <attribute name="name" type="string" use="required"/>
      <attribute name="op" type="sssrmap:OperatorType" use="optional"/>
      <attribute name="conj" type="sssrmap:ConjunctionType" use="optional"/>
      <attribute name="group" type="integer" use="optional"/>
      <attribute name="units" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Where" type="sssrmap:WhereType"/>

<simpleType name="WhereOperatorType">
  <restriction base="string">
    <enumeration value="EQ"/>
    <enumeration value="GT"/>
    <enumeration value="LT"/>
    <enumeration value="GE"/>
    <enumeration value="LE"/>
    <enumeration value="NE"/>
    <enumeration value="Match"/>
  </restriction>
</simpleType>
```

The Option Element

The `Option` element is used to indicate processing options for the command. An option might be used to indicate that command usage or special formatting is desired, or that the command is to be invoked with particular options.

The option value is specified as the text content. An `Option` element without a value may be used as an assertion flag. Only the `name` attribute is required. All other attributes are optional.

Attribute	Description
name	The name of the field being assigned a value
op	The operator to be used to disassert the option • An `op` attribute of “Not” specifies that the option is not asserted
conj	Indicates whether this test is to be ANDed or ORed with the immediately preceding where condition • A `conj` attribute of “And” specifies an “and” conjunction • A `conj` attribute of “Or” specifies an “or” condition • A `conj` attribute of “AndNot” specifies an “and not” conjunction • A `conj` attribute of “OrNot” specifies an “or not” condition

```
<complexType name="OptionType">
  <simpleContent>
    <extension base="string">
      <attribute name="name" type="string" use="required"/>
      <attribute name="op" type="sssrmap:OptionOperatorType" use="optional"/>
      <attribute name="conj" type="sssrmap:ConjunctionType" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Option" type="sssrmap:OptionType"/>

<simpleType name="OptionOperatorType">
  <restriction base="string">
    <enumeration value="Not"/>
  </restriction>
</simpleType>
```

The Data Element

A `Request` or `Response` element may have one or more `Data` elements that allow the supplying of context-specific data. A request might pass in a structured object via a `Data` element to be acted upon. Typically a query will result in a response with the data encapsulated within a `Data` element.

The following attributes are optional:

Attribute	Description
name	Object name describing the contents of the data
type	Describing the form in which the data is represented • A `type` attribute of "XML" indicates the data has internal xml structure and can be recursively parsed by an XML parser • A `type` attribute of "Binary" indicates the data is an opaque dataset consisting of binary data • A `type` attribute of "String" indicates the data is an ASCII string • A `type` attribute of "Int" indicates the data is an integer • A `type` attribute of "Text" indicates the data is in formatted human-readable text • A `type` attribute of "HTML" indicates the data is represented in HTML

```
<complexType name="DataType">
  <sequence>
    <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="name" type="string" use="optional"/>
  <attribute ref="sssrmap:Type" use="optional"/>
</complexType>

<element name="Data" type="sssrmap:DataType"/>
```

The File Element

A `Request` or `Response` element may have one or more `File` elements of type String that allow the inclusion of files. The files may be either text or binary and may be referenced by objects inside the Data element. A file may be compressed using the gzip algorithm [ZIP]. A binary file or a compressed file must

be base64 encoded as defined in XML Digital Signatures ("`http://www.w3.org/2000/09/xmldsig#base64`"). Metadata describing the modes and properties of the resulting file are passed as parameters. The text or base64 encoded file data forms the string content of the `File` element.

The following attributes are optional:

Attribute	Description
id	Specifies an identifier that allows the file to be referenced from within another object. If more than one File elements are specified, this attribute is REQUIRED in each of them.
name	Specifies the name to give the file upon creation on the target system. This can be an absolute or relative pathname (relative to the InitialWorkingDirectory).
owner	Indicates what owner the file should be changed to. By default it will be changed to the UserId that the authenticated actor maps to on the target system. Note that this function should succeed only if the requestor has the privileges to do so (i.e. authenticated as root).
group	Indicates what group the file should be changed to. By default it will be set to the primary groupid of the UserId that the authenticated actor maps to on the target system. Note that this function should succeed only if the requestor has the proper privileges.
mode	Indicates the permissions the file should possess. By default it will be set according to the default umask for the UserId that the authenticated actor maps to on the target system. Note that this function should not set permissions for the file that exceed the privileges for the actor. These permissions can be specified using either an octal number or symbolic operations (as accepted by the GNU chmod(1) command).
compressed	Indicates whether the file has been compressed • A `compressed` attribute of "True" indicates the file has been compressed. • A `compressed` attribute of "False" indicates the file has not been compressed. This is the default.
encoded	Indicates whether the file has been base64 encoded • An `encoded` attribute of "True" indicates the file has been encoded. • An `encoded` attribute of "False" indicates the file has not been encoded. This is the default.

```
<complexType name="FileType">
  <sequence>
    <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="name" type="string" use="optional"/>
  <attribute name="owner" type=="string" use="optional"/>
  <attribute name="group" type="string" use="optional"/>
  <attribute name="mode" type="string" use="optional"/>
  <attribute name="compressed" type="boolean" use="optional"/>
  <attribute name="encoded" type="boolean" use="optional"/>
</complexType>

<element name="file type="sssrmap:FileType"/>
```

The Count Element

A single `Count` element may be included within a `Request` or `Response` and is context-specific. This can be used to represent the number of objects acted upon or returned.

```
<element name="Count" type="positiveInteger"/>
```

The Response Element

The `Response` element specifies an individual response. It MUST contain a `Status` element. It MAY also contain `Count` and any number of `Data` or `File` elements. If chunking has been requested and is supported by the server, a large response may be broken up into multiple chunks (each with their own envelope). The `chunkNum` attribute can be used to indicate which chunk the current one is. The `chunkMax` attribute can be used to determine when all the chunks have been received (all chunks have been received if `chunkNum=chunkMax` or `chunkMax=0`).

It MAY have any of the following attributes:

Attribute	Description
id	Uniquely maps the response to the corresponding request
chunkNum	Integer indicating the current chunk number [1 is implied when this attribute is missing or blank]
chunkMax	Integer indicating the number of chunks expected [-1 means unknown but more chunks to follow; 0 means unknown but this is the last chunk; 0 is implied if this attribute is missing or blank]

```
<complexType name="ResponseType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="sssrmap:Status" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:Count" minOccurs="0" maxOccurs="1"/>
    <element ref="sssrmap:Data" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:File" minOccurs="0" maxOccurs="unbounded"/>
    <any minOccurs="0" maxOccurs="unbounded" namespace="##other"/>
  </choice>
  <attribute name="object" type="string" use="optional"/>
  <attribute name="action" type="string" use="optional"/>
  <attribute name="id" type="string" use="optional"/>
  <attribute name="chunkNum" type="integer" use="optional"/>
  <attribute name="chunkMax" type="integer" use="optional"/>
</complexType>

<element name="Response" type="sssrmap:ResponseType"/>
```

The Status Element

A `Response` element MUST contain a single `Status` element that indicates whether the reply represents a success, warning or failure. This element is composed of the child elements `Value`, `Code` and `Message`. Of these, `Value` and `Code` are required, and `Message` is optional.

```
<complexType name="StatusType">
  <choice minOccurs="1" maxOccurs="unbounded">
    <element ref="sssrmap:Value" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:Code" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:Message" minOccurs="0" maxOccurs="1"/>
    <any minOccurs="0" maxOccurs="unbounded" namespace="##other"/>
  </choice>
</complexType>

<element name="Status" type="sssrmap:StatusType"/>
```

The Value Element

The `Value` element is of type String and MUST have a value of “Success”, “Warning” or “Failure”.

```
<simpleType name="StatusValueType">
  <restriction base="string">
    <enumeration value="Success"/>
    <enumeration value="Warning"/>
    <enumeration value="Failure"/>
  </restriction>
</simpleType>

<element name="Value" type="sssrmap:StatusValueType"/>
```

The Code Element

A `Response` element must contain a single `Code` element that specifies the 3-digit status code for the response. Refer to the next section on Error Reporting for a description and listing of supported status codes.

```
<simpleType name="CodeType">
  <restriction base="string">
    <pattern value="[0-9]{3}"/>
  </restriction>
</simpleType>

<element name="Code" type="sssrmap:CodeType"/>
```

The Message Element

A `Response` element may contain a single `Message` element that is context specific to the success or failure response. The message should be an error message if status is false. If present for a successful response, it may be used as a human readable message for a user interface.

```
<element name="Message" type="string"/>
```

3.3 Modified XPATH Expressions

The `name` attribute used within the `Get`, `Set` and `Where` Elements MUST have the form of a modified XPATH expression as defined in this section. Usually this will just be the simple name of the object property. Some complex objects, such as the SSS Job Object and the SSS Node Object, however, are

represented in a structured way with nested elements. In order to define a consistent and flexible way to access and manipulate these objects as well as keeping the flat XML objects simple and straightforward, SSSRMAP specifies that a "Modified XPATH" syntax be used.

In essence, "Modified XPATH" is defined to be an XPATH [XPATH] expression with the exception that the "//" may be omitted from the beginning of the expression when a document search is desired. Thus, on the server side, a standard XPATH routine can be used by prepending "//" to any expression that does not begin with a "/".

The response data should always include all of the structure of the queried object necessary to place the requested data in its proper context.

See the XPATH specification for a full description of XPATH. The XPath 1.0 Recommendation is `http://www.w3.org/TR/1999/REC-xpath-19991116`. The latest version of XPath 1.0 is available at http://www.w3.org/TR/xpath.

Sample Modified XPATH Expressions

Consider the following hypothetical object(s) (which might be returned within a Data element).

```
<Job>
  <JobId>PBS.1234.0</JobId>
  <Requested>
    <Memory op="GE">512</Memory>
    <Processors>2</Processors>
    <WallDuration>P3600S</WallDuration>
  </Requested>
  <Utilized>
    <Memory metric="Average">488</Memory>
    <WallDuration>P1441S</WallDuration>
  </Utilized>
</Job>
```

To get everything above for this job you would not need a `Get` element:

```
<Request action="Query">
  <Object>Job</Object>
  <Where name="JobId">PBS.1234.0</Where>
</Request>
```

If you used `<Get name="JobId"/>` you would get back:

```
<Job>
  <JobId>PBS.1234.0</JobId>
</Job>
```

If you used `<Get name="Memory"/>` (or `name="/Job/*/Memory"`) you would get:

```
<Job>
  <Requested>
    <Memory op="GE">512</Memory>
  </Requested>
  <Utilized>
    <Memory metric="Average">488</Memory>
  </Utilized>
</Job>
```

If you used <Get name="Requested/Memory"/> (or name="/Job/Requested/Memory") you would get:

```
<Job>
  <Requested>
    <Memory op="GE">512</Memory>
  </Requested>
</Job>
```

If you used <Get name="Memory[@metric='Average']"/> (or name="Memory[@metric]") you would get:

```
<Job>
  <Utilized>
    <Memory metric="Average">488</Memory>
  </Utilized>
</Job>
```

3.4 Examples

Sample Requests

Requesting a list of nodes with a certain configured memory threshold (batch format):

```
<Request action="Query" id="1">
  <Object>Node</Object>
  <Get name="Name" />
  <Get name="Configured/Memory" />
  <Where name="Configured/Memory" op="GE" units="MB">512</Where>
</Request>
```

Activating a couple of users:

```
<Request action="Modify">
  <Object>User</Object>
  <Set name="Active">True</Set>
  <Where name="Name">scott</Where>
  <Where name="Name" conj="Or"/>brett</Where>
</Request>
```

Submitting a simple job:

```
<Request action="Submit">
  <Object>Job</Object>
  <Data>
    <Job>
      <User>xdp</User>
      <Account>youraccount</Account>
      <Command>myprogram</Command>
      <InitialWorkingDirectory>/usr/home/scl/xdp</InitialWorkingDirectory>
      <RequestedNodes>4</RequestedNodes>
      <RequestedWCTime>100</RequestedWCTime>
    </Job>
  </Data>
</Request>
```

Sample Responses

A response to the available memory nodes query (batch format)

```
<Response id="1">
  <Status>
    <Value>Success</Value>
    <Code>000</Code>
  </Status>
  <Count>2</Count>
  <Data>
    <Node>
      <Name>fr01n01</Name>
      <Configured>
        <Memory>512</Memory>
      </Configured>
    </Node>
    <Node>
      <Name>fr12n04</Name>
      <Configured>
        <Memory>1024</Memory>
      </Configured>
    </Node>
  </Data>
</Response>
```

Two users successfully activated

```
<Response>
  <Status>
    <Code>000</Code>
    <Message>Two users were successfully modified</Message>
  </Status>
  <Count>2</Count>
</Response>
```

A failed job submission:

```
<Response>
  <Status>
    <Value>Failure</Value>
    <Code>711</Code>
    <Message>Invalid account specified. The job was not submitted.</Message>
  </Status>
</Response>
```

4.0 Error Reporting

SSSRMAP requests will return a status and a 3-digit response code to signify success or failure conditions. When a `request` is successful, a corresponding `response` is returned with the `status` element set to Success and the `code` element set to "000". When a `request` results in an error detected by the server, a `response` is returned with the `status` element set to Failure and a 3-digit error code in the `code` element. An optional human-readable `message` may also be include in a failure response providing context-specific detail about the failure. The default message language is US English. (The status flag makes it easy to signal success or failure and allows the receiving peer some freedom in the amount of parsing it wants to do on failure [BXXP]).

Success codes:

Code	Response Text in US English
0xx	Request was successful
000	General Success
010	Help/usage reply
020	Status reply
030	Subscription successful
035	Notification successful (Ack)
040	Registration successful
050-079	Component-defined
080-099	Application-defined

Warning codes:

Code	Response Text in US English
1xx	Request was successful but includes a warning
100	General warning (examine message for details)
102	Check result (Did what you asked but may not have been what you intended -- or information is suspect)
110	Wire Protocol or Network warning
112	Redirect
114	Protocol warning (something was wrong with the protocol, but best effort guesses were applied to fulfill the request)

Code	Response Text in US English
120	Message Format warning
122	Incomplete specification (request missing some essential information -- best effort guess applied)
124	Format warning (something was wrong with the format but best effort guesses were applied to fulfill the request)
130	Security warning
132	Insecure request
134	Insufficient privileges (Response was sanitized or reduced in scope due to lack of privileges)
140	Content or action warning
142	No content (The server has processed the request but there is no data to be returned
144	No action taken (nothing acted upon -- i.e. deletion request did not match any objects)
146	Partial content
148	Partial action taken
150-179	Component-defined
180-199	Application-defined

Wire protocol codes:

Code	Response Text in US English
2xx	A problem occurred in the wire protocol or network
200	General wire protocol or network error
210	Network failure

Code	Response Text in US English
212	Cannot resolve host name
214	Cannot resolve service port
216	Cannot create socket
218	Cannot bind socket
220	Connection failure
222	Cannot connect
224	Cannot send data
226	Cannot receive data
230	Connection rejected
232	Timed out
234	Too busy
236	Message too large
240	Framing failure
242	Malformed framing protocol
244	Invalid payload size
246	Unexpected end of file
250-279	Component-defined
280-299	Application-defined

Message format codes:

Code	Response Text in US English
3xx	A problem occurred in the message format
300	General message format error
302	Malformed XML document
304	Validation error(XML Schema)
306	Name space error
308	Invalid message type (Something other than Request or Response in Body
310	General syntax error in request
311	Object incorrectly (or not) specified
312	Action incorrectly (or not) specified
313	Invalid Action
314	Missing required element or attribute
315	Invalid Object (or Object-Action combination
316	Invalid element or attribute name
317	Illegal value for element or attribute
318	Illegal combination
319	Malformed Data
320	General syntax error in response
321	Status incorrectly (or not)specified

Code	Response Text in US English
322	Code incorrectly (or not)specified
324	Missing required element or attribute
326	Invalid element or attribute name
327	Illegal value for element or attribute
328	Illegal combination
329	Malformed Data
340	Pipelining failure
342	Request identifier is not unique
344	Multiple messages not supported
346	Mixed messages not supported (Both requests and responses in same batch)
348	Request/response count mismatch
350-379	Component-defined
380-399	Application-defined

Security codes:

Code	Response Text in US English
4xx	A security requirement was not fulfilled
400	General security error
410	Negotiation failure
412	Not understood

Code	Response Text in US English
414	Not supported
416	Not accepted
420	Authentication failure
422	Signature failed at client
424	Authentication failed at server
426	Signature failed at server
428	Authentication failed at client
430	Encryption failure
432	Encryption failed at client
434	Decryption failed at server
436	Encryption failed at server
438	Decryption failed at client
440	Authorization failure
442	Authorization failed at client
444	Authorization failed at server
450-479	Component-defined
480-499	Application-defined

Event management codes:

Code	Response Text in US English
5xx	Failure conditions in event messaging
500	General Event Management failure
510	Subscription failed
520	Notification failed
550-579	Component-defined
580-599	Application-defined

Reserved codes:

Code	Response Text in US English
6xx	Reserved for future use

Server application codes:

Code	Response Text in US English
7xx	A server-side application-specific error occurred
700	General failure
710	Not supported
712	Not understood
720	Internal error
730	Resource unavailable (insufficient resources -- software, hardware or a service I rely upon is down)
740	Business logic

Code	Response Text in US English
750-779	Component-defined
780-799	Application-defined

Client application codes:

Code	Response Text in US English
8xx	A client-side application-specific error occurred
800	General failure
810	Not supported
812	Not understood
820	Internal error
830	Resource unavailable
840	Business logic
850-879	Component-defined
880-899	Application-defined

Miscellaneous codes:

Code	Response Text in US English
9xx	Miscellaneous failures
999	Unknown failure

5.0 References

[BEEP] M. Rose, "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

[FED_XML] "U.S. Federal XML Guidelines".

[HMAC] H. Krawczyk, M. Bellare, R. Canetti, "HMAC, Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[HTTP] "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616, June 1999.

[RFC2119] S. Bradner, "Key Words for Use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[RFC3117] M. Rose, "On the Design of Application Protocols", Informational RFC 3117, November 2001.

[SHA-1] U.S. Department of Commerce/National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-1.

[SSS] "Scalable Systems Software", http://www.scidac.org/ScalableSystems

[WIRE_PROTOCOL] S. Jackson, B. Bode, D. Jackson, K. Walker, "Systems Software Resource Management and Accounting Protocol (SSSRMAP) Wire Protocol", SSS Resource Management and Accounting Documents, January 2004.

[XML] Bray, T., et al, "Extensible Markup Language (XML) 1.0 (Second Edition)", 6 October 2000.

[XML_CONV] "I-X and <I-N-CA> XML Conventions".

[XML_DSIG] D. Eastlake, J. Reagle Jr., D. Solo, "XML Signature Syntax and Processing", W3C Recommendation, 12 February 2002.

[XML_ENC] T. Imamura, B. Dillaway, E. Smon, "XML Encryption Syntax and Processing", W3C Candidate Recommendation, 4 March 2002.

[XML_SCHEMA] D. Beech, M. Maloney, N. Mendelshohn, "XML Schema Part 1: Structures Working Draft", April 2000.

[XPath 1.0] J. Clark, S. DeRose, "XML Path Language (XPath) Version 1.0", 16 November 1999.

[XRP] E. Brunner-Williams, A. Damaraju, N. Zhang, "Extensible Registry Protocol (XRP)", Internet Draft, expired August 2001.

[ZIP] J. Gailly, M. Adler, "The gzip home page", http://www.gzip.org/

Scalable Systems Software Node Object Specification

SSS Node Object Specification
Release Version 3.1.0
26 April 2011

Scott Jackson, PNNL
David Jackson, Ames Lab
Brett Bode, Ames Lab

Status of This Memo

This is a specification of the node object to be used by Scalable Systems Software compliant components. It is envisioned for this specification to be used in conjunction with the SSSRMAP protocol with the node object passed in the Data field of Requests and Responses. Queries can be issued to a node-cognizant component in the form of modified XPATH expressions to the Get field to extract specific information from the node object as described in the SSSRMAP protocol.

Abstract

This document describes the syntax and structure of the SSS node object. This node model takes into account various node property categories such as whether it represents a configured, available or utilized property.

Table of Contents

- Scalable Systems Software Node Object Specification
- Table of Contents
- 1.0 Introduction
  - 1.1 Goals
  - 1.2 Examples
    - 1.2.1 Simple Example
    - 1.2.2 Elaborate Example
- 2.0 Conventions Used in This Document
  - 2.1 Keywords
  - 2.2 Table Column Interpretations
  - 2.3 Element Syntax Cardinality
- 3.0 The Node Model
- 4.0 Node Element
  - 4.1 Uncategorized Node Properties
    - 4.1.1 Simple Node Properties
    - 4.1.2 Extension Element
  - 4.2 Property Categories
    - 4.2.1 Configured Element
    - 4.2.2 Available Element
    - 4.2.3 Utilized Element
  - 4.3 Categorized Node Properties
    - 4.3.1 Consumable Resources
    - 4.3.2 Resource Element
- Appendix A
- Units of Measure Abbreviations

1.0 Introduction

This specification proposes a standard XML representation for a node object for use by the various components in the SSS Resource Management System. This object will be used in multiple contexts and by multiple components. It is anticipated that this object will be passed via the Data Element of SSSRMAP Requests and Responses.

1.1 Goals

There are several goals motivating the design of this representation.

It needs to be inherently flexible. We recognize we will not be able to exhaustively include the ever-changing node properties and capabilities that constantly arise.

The same node object should be used at all stages of its lifecycle. This object needs to distinguish between configured, available and utilized properties of a node.

Its design takes into account the properties and structure required to function in a meta environment. It should eventually include the capability of resolving name space and locality issues, though the earliest versions will ignore this requirement.

One should not have to make multiple queries to obtain a single piece of information — i.e. there should not be two mutually exclusive ways to represent a node resource.

Needs to support resource metric as well as unit specifications.

1.2 Examples

Simple Example

This example shows a simple expression of the Node object.

```
<Node>
  <Id>Node64</Id>
  <Configured>
    <Processors>2</Processors>
    <Memory>512</Memory>
  </Configured>
</Node>
```

Elaborate Example

This example shows a more elaborate Node object.

```
<Node>
  <Id>64</Id>
  <Name>Netpipe2</Name>
  <Feature>BigMem</Feature>
  <Feature>NetOC12</Feature>
  <Opsys>AIX</Opsys>
  <Arch>Power4</Arch>
  <Configured>
    <Processors>16</Processors>
    <Memory units="MB">512</Memory>
    <Swap>512</Swap>
  </Configured>
  <Available>
```

```
    <Processors>7</Processors>
    <Memory metric="Instantaneous">143</Memory>
  </Available>
  <Utilized>
    <Processors wallDuration="3576">8</Processors>
    <Memory metric="Average" wallDuration="3576">400</Memory>
  </Utilized>
</Node>
```

2.0 Conventions Used in This Document

2.1 Keywords

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC2119.

2.2 Table Column Interpretations

In the property tables, the columns are interpreted to have the following meanings:

Property	Description
Element Name	Name of the XML element (xsd:element)
Type	Data type defined by xsd (XML Schema Definition) as: • String — xsd:string(a finite length sequence of printable characters) • Integer — xsd:integer(a signed finite length sequence of decimal digits) • Float — xsd:float (single-precision 32-bit floating point) • Boolean — xsd:boolean (consists of the literals "true" or "false") • DateTime — xsd:dateTime (discreet time values are represented in ISO 8601 extended format `CCYY-MM-DDThh:mm:ss` where `CC` represents the century, `YY` the year, `MM` the month and `DD` the day. The letter `T` is the date/time separator and `hh`, `mm`, `ss` represent hour, minute and second respectively. This representation may be immediately followed by a `Z` to indicate Coordinated Universal Time (UTC) or, to indicate the time zone, i.e. the difference between the local time and Coordinated Universal Time, immediately followed by a sign, + or –, followed by the difference from UTC.) • Duration — xsd:duration (a duration of time is represented in ISO 8601 extended format PnYnMnDTnHnMnS, where nY represents the number of years, nM the number of months, nD the number of days, `T` is the date/time separator, nH the number of hours, nM the number of minutes and nS the number of seconds. The number of seconds can include decimal digits to arbitrary precision.)
Description	Brief description of the meaning of the property

Property	Description
Appearance	This column indicates whether the given property has to appear within the parent element. It assumes the following meanings: • MUST — This property is REQUIRED when the parent is specified. • SHOULD — A compliant implementation SHOULD support this property. • MAY — A compliant implementation MAY support this property.
Compliance	The column indicates whether a compliant implementation has to support the given property. • MUST — A compliant implementation MUST support this property. • SHOULD — A compliant implementation SHOULD support this property. • MAY — A compliant implementation MAY support this property.
Categories	Some properties may be categorized into one of several categories. Letters in this column indicate that the given property can be classified in the following property categories. • C — This property can be encompassed in a Configured element. • A — This property can be encompassed in an Available element. • U — This property can be encompassed in a Utilized element.

2.3 Element Syntax Cardinality

The cardinality of elements in the element syntax sections may make use of regular expression wildcards with the following meanings:

Wildcard	Description
*	Zero or more occurrences
+	One or more occurrences
?	Zero or one occurrences

The absence of one of these symbols implies one and only one occurrence.

3.0 The Node Model

The primary element within the node model is a node. One can speak of some node properties as being a configured, available or utilized property of the node.

4.0 Node Element

The Node element is the root element of a node object and is used to encapsulate a node.

- A node object MUST have exactly one Node element.
- A compliant implementation MUST support this element.

- A node MUST specify one or more Node Properties.

4.1 Uncategorized Node Properties

Uncategorized Node Properties are properties that apply to the node as a whole and do not need to be distinguished between being configured, available or utilized. These include the node id and other optional node properties.

Simple Node Properties

Simple (unstructured) node properties are enumerated in the table below.

Table 25-7: Simple Node Properties

Element Name	Type	Description	Appearance	Compliance
Id	String	Node identifier	MUST	MUST
Name	String	Node name or pattern	MAY	MAY
OpSys	String	Operating System	MAY	SHOULD
Arch	String	Architecture	MAY	SHOULD
Description	String	Description of the node	MAY	MAY
State	String	State of the node. Valid states may include Offline, Configured, Unknown, Idle, and Busy.	SHOULD	MUST
Features	String	Arbitrary named features of the node (comma-delimited string)	MAY	SHOULD

Extension Element

The `Extension` element provides a means to pass extensible properties with the node object. Some applications may find it easier to deal with a named extension property than discover and handle elements for which they do not understand or anticipate by name.

- A compliant implementation MAY support this element.
- This element MUST have a name attribute that is of type String and represents the name of the extension property. A compliant implementation MUST support this attribute if this element is supported.
- This element MAY have a type attribute that is of type String and provides a hint about the context within which the property should be understood. A compliant implementation SHOULD

support this attribute if this element is supported.

- The character content of this element is of type String and is the value of the extension property.

The following is an example of an `Extension` element:

```
<Extension type="Chemistry" name="Software">NWChem</Extension>
```

4.2 Property Categories

Certain node properties (particularly consumable resources) need to be classified as being in a particular category. This is done when it is necessary to distinguish between a property that is configured versus a property that is available or utilized. For example, a node might be configured with 16 processors. At a particular time, 8 might be utilized, 7 might be available and 1 disabled. When a node property must be categorized to be understood properly, the property MUST be enveloped within the appropriate Property Category Element.

Configured Element

A configured node property reflects resources pertaining to the node that could in principle be used though they may not be available at this time. This information could be used to determine if a job could ever conceivably run on a given node.

- A compliant implementation MUST support this element.

The following is an example of using Configured Properties:

```
<Configured>
  <Processors>16</Processors>
  <Memory units="MB">512</Memory>
</Configured>
```

Available Element

An available node property refers to a resource that is currently available for use.

- A compliant implementation SHOULD support this element.

The following is an example of specifying available properties:

```
<Available>
  <Processors>7</Processors>
  <Memory units="MB">256</Memory>
</Available>
```

Utilized Element

A utilized node property reflects resources that are currently utilized.

- A compliant implementation SHOULD support this element.

The following is an example of specifying utilized properties:

```
<Utilized>
  <Processors>8</Processors>
  <Memory metric="Average">207</Memory>
</Utilized>
```

4.3 Categorized Node Properties

Consumable Resources

Consumable Resources are a special group of node properties that can have additional attributes and can be used in multiple categories. In general a consumable resource is a resource that can be consumed in a measurable quantity.

- A consumable resource MUST be categorized as being a configured, available or utilized node property by being a child element of a Configured, Available or Utilized element respectively.
- A consumable resource MAY have a units attribute that is of type String that specifies the units by which it is being measured. If this attribute is omitted, a default unit is implied. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a metric attribute that is of type String that specifies the type of measurement being described. For example, the measurement may be a Total, an Average, a Min or a Max. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a wallDuration attribute of type Duration that indicates the amount of time for which that resource was used. This need only be specified if the resource was used for a different amount of time than the wallDuration for the step. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a consumptionRate attribute of type Float that indicates the average percentage that a resource was used over its wallDuration. For example, an overbooked SMP running 100 jobs across 32 processors may wish to scale the usage and charge by the average fraction of processor usage actually delivered. A compliant implementation MAY support this attribute if the element is supported.

A list of simple consumable resources is listed in the table below.

Table 25-8: Consumable Resource Node Properties

Element Name	Type	Description	Appearance	Compliance	Categories
Processors	Integer	Number of processors	MAY	MUST	CAU
Memory	Float	Amount of memory	MAY	SHOULD	CAU
Disk	Float	Amount of disk	MAY	SHOULD	CAU
Swap	Float	Amount of virtual memory	MAY	MAY	CAU
Network	Float	Amount of network	MAY	MAY	CAU

The following are two examples for specifying a consumable resource:

```
<Memory metric="Max" units="GB">483</Memory>
<Processors wallDuration="1234" consumptionRate="0.63">4</Processors>
```

Resource Element

In addition to the consumable resources enumerated in the above table, an extensible consumable resource is defined by the Resource element.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or more times within a given set of node properties.
- Like the other consumable resources, this property MUST be categorized as a configured, available or utilized property by being encompassed in the appropriate elements.
- This element is of type Float.
- It shares the other same properties and attributes as the other consumable resources but it requires an additional name (and optional type) attribute to describe it.
- This element MUST have a name attribute of type String that indicates the type of consumable resource being measured. A compliant implementation MUST support this attribute if the element is supported.
- This element MAY have a type attribute of type String that distinguishes it within a general resource class. A compliant implementation SHOULD support this attribute if the element is supported.

The following are two examples for specifying a `Resource` element:

```
<Resource name="License" type="MATLAB">1</Resource>
<Resource name="Telescope" type="Zoom2000" wallDuration="750"
metric="KX">10</Resource>
```

4.4 Node Reference

When a simple reference to a predefined node is needed in an encapsulating element, a Node element is used with the text content being the node id:

```
<Node>node1</Node>
```

- This element MAY have an aggregation attribute of type String that provides a way to indicate multiple values with a single expression. A compliant implementation MAY support the aggregation attribute if the Feature element is supported. Possible values for this attribute include:
    - List a comma-separated list of features
    - Pattern a regular expression (perl5) matching desired features
    - Range a range of nodes of the form: `<prefix>[5-23,77]`
- If an aggregation attribute is specified with the value of List, this element MAY also have a delimiter attribute of type String that indicates what delimiter is used to separate list elements. The default list delimiter is a comma.

- This element MAY have a count attribute of type Integer that indicates the instance count of the specified node(s).

The following is another example of a `Node` element:

```
<Node aggregation="Pattern">node[1-5]</Node>
```

Appendix A

Units of Measure Abbreviations

Abbreviation	Definition	Quantity
B	byte	1 byte
KB	Kilobyte	2^10 bytes
MB	Megabyte	2^20 bytes
GB	Gigabyte	2^30 bytes
TB	Terabyte	2^40 bytes
PB	Petabyte	2^50 bytes
EB	Exabyte	2^60 bytes
ZB	Aettabyte	2^70 bytes
YB	Yottabyte	2^80 bytes
NB	Nonabyte	2^90 bytes
DB	Doggabyte	2^100 bytes

Scalable Systems Software Resource Management and Accounting Protocol (SSSRMAP) Wire Protocol

Resource Management Interface Specs
Release v. 3.0.3
13 May 2004

Scott Jackson
Brett Bode
David Jackson
Kevin Walker

Status of This Memo

This is a specification defining a wire level protocol used between Scalable Systems Software components. It is intended that this specification will continue to evolve as these interfaces are implemented and thoroughly tested by time and experience.

Abstract

This document is a specification describing a connection-oriented XML-based application layer client-server protocol for the interaction of resource management and accounting software components developed as part of the Scalable Systems Software Center. The SSSRMAP Wire Protocol defines a framing protocol that includes provisions for security. The protocol is specified in XML Schema Definition and rides on the HTTP protocol.

Table of Contents

- Scalable Systems Software Resource Management and Accounting Protocol (SSSRMAP) Wire Protocol
- Table of Contents
- 1.0 Introduction
- 2.0 Conventions Used in this Document
    - 2.1 Keywords
    - 2.2 XML Case Conventions
    - 2.3 Schema Definitions
- 3.0 Encoding
    - 3.1 Schema Header and Name spaces
    - 3.2 The Envelope Element
    - 3.3 The Body Element
- 4.0 Transport Layer
- 5.0 Framing
    - 5.1 Message Header Requirements
    - 5.2 Message Chunk Format
    - 5.3 Reply Header Requirements

    - 5.4 Reply Chunk Format
    - 5.5 Message and Reply Tail Requirements and Multiple Chunks
    - 5.6 Examples
        - 5.6.1 Sample SSSRMAP Message Embedded in HTTP Request
        - 5.6.2 Sample SSSRMAP Reply Embedded in HTTP Response
- 6.0 Asynchrony
- 7.0 Security
    - 7.1 Security Token
        - 7.1.1 The SecurityToken Element
        - 7.1.2 Security Token Types
            - 7.1.2.1 Symmetric Key
            - 7.1.2.2 Asymmetric Key
            - 7.1.2.3 Password
            - 7.1.2.4 Cleartext
            - 7.1.2.5 Kerberos
            - 7.1.2.6 GSI (X.509)
        - 7.1.3 Example
    - 7.2 Authentication
        - 7.2.1 The Signature Element
        - 7.2.2 The DigestValue Element
        - 7.2.3 The SignatureValue Element
        - 7.2.4 Signature Example
    - 7.3 Confidentiality
        - 7.3.1 The EncryptedData Element
        - 7.3.2 The EncryptedKey Element
        - 7.3.3 The CipherValue Element
        - 7.3.4 Encryption Example
- 8.0 Acknowledgements
- 9.0 References

1.0 Introduction

A major objective of the Scalable Systems Software [SSS] Center is to create a scalable and modular infrastructure for resource management and accounting on terascale clusters including resource scheduling, node daemon support, comprehensive usage accounting and user interfaces emphasizing portability to terascale vendor operating systems. Existing resource management and accounting components feature disparate APIs (Application Programming Interfaces) requiring various forms of application coding to interact with other components.

This document proposes a wire level protocol expressed in an XML envelope to be considered as the foundation of a standard for communications between and among resource management and accounting software components. Individual components additionally need to define the particular XML binding necessary to represent the message format for communicating with the component.

2.0 Conventions Used in this Document

2.1 Keywords

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC2119.

2.2 XML Case Conventions

In order to enforce a consistent capitalization and naming convention across all SSSRMAP specifications "Upper Camel Case" (UCC) and "Lower Camel Case" (LCC) Capitalization styles shall be used. UCC style capitalizes the first character of each word and compounds the name. LCC style capitalizes the first character of each word except the first word. [XML_CONV][FED_XML]

1. SSSRMAP XML Schema and XML instance documents SHALL use the following conventions:
   - Element names SHALL be in UCC convention (example: `<UpperCamelCaseElement/>`.
   - Attribute names SHALL be in LCC convention (example: `<UpperCamelCaseElement lowerCamelCaseAttribute="Whatever"/>`.
2. General rules for all names are:
   - Acronyms SHOULD be avoided, but in cases where they are used, the capitalization SHALL remain (example: XMLSignature).
   - Underscores (_), periods (.) and dashes (-) MUST NOT be used (example: use JobId instead of JOB.ID, Job_ID or job-id).

2.3 Schema Definitions

```
SSSRMAP Schema Definitions appear like this
```

In case of disagreement between the schema file and this specification, the schema file takes precedence.

3.0 Encoding

Encoding tells how a message is represented when exchanged. SSSRMAP data exchange messages SHALL be defined in terms of XML schema [XML_SCHEMA].

3.1 Schema Header and Name spaces

The header of the schema definition is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<schema
  xmlns="http://www.w3.org/201/XMLSchema"
  xmlns:sssrmap="http://www.scidac.org/ScalableSystems/SSSRMAP"
  targetNamespace="http://www.scidac.org/ScalableSystems/SSSRMAP"
  elementFormDefault="qualified">
```

3.2 The Envelope Element

SSSRMAP messages and replies are encapsulated in the `Envelope` element. There are two possibilities for the contents of this element. If the contents are unencrypted, this element MUST contain a `Body` element and MAY contain a `Signature` element (refer to the section on Security). If the contents are encrypted, this element MUST contain exactly one `EncryptedData` element (refer to the section on Security). The `Envelope` element MAY contain name space and other xsd-specific information necessary to validate the document against the schema. In addition, it MAY have any of the following attributes which may serve as processing clues to the parser:

Attribute	Description
type	A message type providing a hint as to the body contents such as “Request” or “Notification”
component	A component type such as “QueueManager” or “LocalScheduler”
name	A component name such as “OpenPBS” or “Maui”
version	A component version such as “2.2p12” or “3.2.2”

```
<complexType name=EnvelopeType">
  <choice minOccurs="1" maxOccurs="1">
    <choice minOccurs="1" maxOccurs="2">
      <element ref="sssrmap:Signature" minOccurs="0" maxOccurs="1"/>
      <element ref="sssrmap:Body" minOccurs="1" maxOccurs="1"/>
    </choice>
    <element ref="sssrmap:EncryptedData" minOccurs="1" maxOccurs="1"/>
  </choice>
  <attribute name="type" type="string" use="optional"/>
  <attribute name="component" type="string" use="optional"/>
  <attribute name="name" type="string" use="optional"/>
  <attribute name="version" type="string" use="optional"/>
</complexType>

<element name="Envelope" type="sssrmap:EnvelopeType"/>
```

3.3 The Body Element

- SSSRMAP messages and replies are encapsulated in the `Body` element. This element MUST contain exactly one `Request` or `Response` element.

```
<complexType name="BodyType">
  <choice minOccurs="1" maxOccurs="1">
    <element ref="sssrmap:Request" minOccurs="0" maxOccurs="1"/>
    <element ref="sssrmap:Response" minOccurs="0" maxOccurs="1"/>
    <any minOccurs="0" maxOccurs="1" namspace="##other"/>
  </choice>
</complexType>

<element name="Body" type="sssrmap:BodyType"/>
```

4.0 Transport Layer

This protocol will be built over the connection-oriented reliable transport layer TCP/IP. Support for other transport layers could also be considered, but native support for TCP/IP can be found on most terascale clusters and automatically handles issues such as reliability and connection fullness for the application developer implementing the SSSRMAP protocol.

5.0 Framing

Framing specifies how the beginning and ending of each message is delimited. Given that the encoding will be expressed as one or more XML documents, clients and servers need to know when an XML document has been fully read in order to be parsed and acted upon.

SSSRMAP uses the HTTP 1.1 [HTTP] protocol for framing. HTTP uses a byte-counting mechanism to delimit the message segments. HTTP chunked encoding is used. This allows for optional support for batched messages, large message segmentation and persistent connections.

5.1 Message Header Requirements

The HTTP request line (first line of the HTTP request header) begins with POST and is followed by a URI and the version of the HTTP protocol that the client understands. It is suggested for this protocol that the URI consist of a single slash, followed by the protocol name in uppercase (i.e. /SSSRMAP), though this field is not checked and could be empty, a single slash or any URI.

The Content-Type must be specified as test/xml. Charset may be optionally specified and defaults to US-ASCII. It is recommended that charset be specified as “utf-8” for maximum interoperability.

The Transfer-Encoding must be specified as chunked. The Content-Length must NOT be specified as the chunk size is specified in the message chunk.

Other properties such as User-Agent, Host and Date are strictly optional.

5.2 Message Chunk Format

A message chunk consists of a chunk size in hexadecimal format (whose value is the number of bytes in the XML message not including the chunk size and delimiter) delimited by a CR/LF “\r\n” and followed by the message payload in XML that consists of a single XML document having a root element of `Envelope`.

5.3 Reply Header Requirements

The HTTP response line (first line of the HTTP response header) begins with HTTP and a version number, followed by a numeric code and a message indicating what sort of response is made. These response codes and messages indicate the status of the entire response and are as defined by the HTTP

standard. The most common response is 200 OK, indicating that the message was received and an appropriate response is being returned.

The Content-Type must be specified as text/xml. Charset may be optionally specified and defaults to US-ASCII. It is recommended that charset be specified as “utf-8” for maximum interoperability.

The Transfer-Encoding MUST be specified as chunked. The Content-Length must NOT be specified.

Other properties such as Server, Host and Date are strictly optional.

5.4 Reply Chunk Format

A reply chunk consists of a chunk size in hexadecimal format (whose value is the number of bytes in the XML reply not including the chunk size and delimiter) delimited by a CR/LF “\r\n” and followed by the reply payload in XML that consists of a single XML document having a root element of `Envelope`.

5.5 Message and Reply Tail Requirements and Multiple Chunks

This specification only requires that single chunks be supported. A server may optionally be configured to handle requests with persistent connections (multiple chunks). It will be the responsibility of clients to know whether a particular server supports this additional functionality. After all chunks have been sent, a connection is terminated by sending a zero followed by a carriage return-linefeed combination (0\r\n) and closing the connection.

5.6 Examples

Sample SSSRMAP Message Embedded in HTTP Request

```
POST /SSSRMAP HTTP/1.1\r\n
Content-Type: text/xml; charset=”utf-8”\r\n
Transfer-Encoding: chunked\r\n
\r\n
9A\r\n
<Envelope …/>
0\r\n
```

Sample SSSRMAP Reply Embedded in HTTP Response

```
HTTP/1.1 200 OK\r\n
Content-Type: text/xml; charset=”utf-8”\r\n
Transfer-Encoding: chunked\r\n
\r\n
2B4\r\n
<Envelope …/>
0\r\n
```

6.0 Asynchrony

Asynchrony (or multiplexing) allows for the handling of independent exchanges over the same connection. A widely-implemented approach is to allow pipelining (or boxcarring) by aggregating requests or responses within the body of the message or via persistent connections and chunking in HTTP 1.1. Pipelining helps reduce network latency by allowing a client to make multiple requests of a server, but requires the requests to be processed serially [RFC3117]. Parallelism could be employed to further reduce server latency by allowing multiple requests to be processed in parallel by multi-threaded applications.

Segmentation may become necessary if the messages are larger than the available window. With support for segmentation, the octet-counting requirement that you need to know the length of the whole message before sending it can be relegated to the segment level – and you can start sending segments before the whole message is available. Segmentation is facilitated via "chunking" in HTTP 1.1.

The current SSSRMAP strategy supports only a single request or response within the Body element. A server may optionally support persistent connections from a client via HTTP chunking. Segmentation of large responses is also optionally supported via HTTP chunking. Later versions of the protocol could allow pipelined requests and responses in a single Body element.

7.0 Security

SSSRMAP security features include capabilities for integrity, authentication, confidentiality, and non-repudiation. The absence or presence of the various security features depend upon the type of security token used and the protection methods you choose to specify in the request.

For compatibility reasons, SSSRMAP specifies six supported security token types. Extensibility features are included allowing an implementation to use alternate security algorithms and security tokens. It is also possible for an implementation to ignore security features if it is deemed nonessential for the component. However, it is highly RECOMMENDED that an implementation support at least the default security token type in both authentication and encryption.

7.1 Security Token

A security token may be included in either the Signature block, and/or in the EncryptedData block (both described later) as an implicit or explicit cryptographic key. If this element is omitted, the security token is assumed to be a secret key shared between the client and the server.

The SecurityToken Element

This element is of type String. If the security token conveys an explicit key, this element's content is the value of the key. If the key is natively expressed in a binary form, it must be converted to base64 encoding as defined in XML Digital Signatures ("http://www.w3.org/2000/09/xmldsig#base64"). If the type is not specified, it is assumed to be of type "Symmetric".

It may have any of the following optional attributes:

Attribute	Description
type	The type of security token (described subsequently) • A `type` attribute of "Symmetric" specifies a shared secret key between the client and server. This is the default. • A `type` attribute of "Asymmetric" specifies the use of public private key pairs between the client and server. • A `type` attribute of "Password" encrypts and authenticates with a user password known to both the client and server. • A `type` attribute of "Cleartext" allows the passing of a cleartext username and password and depends on the use of a secure transport (such as SSL or IPSec). • A `type` attribute of "Kerberos5" specifies a kerberos token. • A `type` attribute of "X509v3" specifies an X.509 certificate.
name	The name of the security token which serves as an identifier for the actor making the request (useful when the key is a password, or when the key value is implicit as when a public key is named but not included)

```
<complexType name="SecurityTokenType" mixed="true">
  <simpleContent>
    <extension base="string">
      <attribute name="type" type="string" use="optional">
      <attribute name="name" type="string" use="optional">
    </extension>
  </simpleContent>
</complexType>

<element name="SecurityToken" type="sssrmap:SecurityTokenType"/>
```

Security Token Types

SSSRMAP defines six standard security token types:

Symmetric Key

The default security token specifies the use of a shared secret key. The secret key is up to 128-bits long and known by both client and server. When using a symmetric key as a security token, it is not necessary to specify the `type` attribute with value "Sym metric" because this is assumed when the attribute is absent. The `name` attribute should be specified indicating the actor issuing the request. If the user provides a password to be sent to the server for authentication, then the password is encrypted with the secret key using a default `method="kw-tripledes"` (XML ENCRYPTION http://www.w3.org/2001/04/xmlenc#kw-tripledes), base64 encoded and included as the string content of the `SecurityToken` element. If the client authenticated the user, then the `SecurityToken` element is empty. The same symmetric key is used in both authentication and encryption.

Asymmetric Key

Public and private key pairs can be used to provide non-repudiation of the client (or server). The client and the server must each have their own asymmetric key pairs. This mode is indicated by specifying the type attribute as "Asymmetric". The `name` attribute should be specified indicating the actor issuing the request. If the user provides a password to be sent to the server for authentication, then the password is encrypted with the server's public key using a default `method="rsa-1_5"` (XML ENCRYPTION

http://www.w3.org/2001/04/xmlenc#rsa-1_5), base64 encoded and included as the string content of the *SecurityToken* element. If the client authenticated the user, then the `SecurityToken` element is empty .The sender's private key is used in authentication (signing) while the recipient's public key is used for encryption.

Password

This mode allows for a username password combination to be used under the assumption that the server also knows the password for the user. This security token type is indicated by specifying a value of "Password" for the *type* attribute. The password itself is used as the cryptographic key for authentication and encryption. The *name* attribute contains the user name of the actor making the request. The *SecurityToken* element itself is empty.

Cleartext

This security mode is equivalent to passing the username and password in the clear and depends upon the use of a secure transport (such as SSL or IPSec). The purpose of including this security token type is to enable authentication to occur from web browsers over SSL or over internal LANs who use IPSec to encrypt all traffic. The password (or a hash of the password like in /etc/passwd) would have to be known by the server for authentication to occur. In this mode, neither encryption nor signing of the hash is performed at the application layer. This mode is indicated by specifying a value of "Cleartext" for the *type* attribute. The `name` attribute contains the user name of the actor making the request and the string content of the `SecurityToken` element is the unencrypted plaintext password.

Kerberos

The use of a Kerberos version 5 token is indicated by specifying "Kerberos5" in the `type` attribute. The `name` attribute is used to contain the kerberos user id of the actor making the request. The `SecurityToken` element contains two subelements. The *Authenticator* element contains the authenticator encoded in base64. A `Ticket` element contains the service-granting ticket, also base64 encoded.

GSI (X.509)

The Grid Security Infrastructure (GSI) which is based on public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol can be indicated by specifying a `type` attribute of "X509v3". The name attribute contains the userid used that the actor was mapped to in the local system. The string content of the `SecurityToken` element is the GSI authentication message including the X.509 identity of the sender encoded in base64.

Example

```
<SecurityToken type="Asymmetric" name="scottmo">
MIIEZzCCA9CggAwIBAgIQEmtJZc0rqrKh5i...
</SecurityToken>
```

7.2 Authentication

Authentication entails how the peers at each end of the connection are identified and verified. Authentication is optional in an SSSRMAP message or reply. SSSRMAP uses a digital signature scheme for authentication that borrows from concepts in XML Digital Signatures [XML_DSIG]. In addition to authentication, the use of digital signatures also ensures integrity of the message, protecting exchanges from third-party modification.

When authentication is used, a `Signature` element is prepended as the first element within the `Envelope` element. All of the security modes will create a digest of the data for integrity checking and store this in base64 encoding in a `DigestValue` element as a child of the `Signature` element. The digital signature is created by encrypting the hash with the appropriate security token and storing this value in a `SignatureValue` element as a child of the `Signature` element. The security token itself is included as a child of the `Security` element within a `SecurityToken` element.

There are a number of procedural practices that must be followed in order to standardize this approach. The digest (or hash) is created over the contents of the `Envelope` element (not including the Element tag or its attributes). This might be over one or more `Request` or `Notify` elements (or `Response` or `Ack` elements) and necessarily excludes the `Signature` Element itself. (Note that any data encryption is performed after the creation of the digital signature and any decryption is performed before authenticating so the `EncryptedData` element will not interfere with this process. Hence, the signature is always based on the (hashed but) unencrypted data). For the purposes of generating the digest over the same value, it is assumed that the data is first canonicalized to remove extraneous whitespace, comments, etc according to the XML Digital Signature algorithm ("http://www.w3.org/TR/2001/REC-xml-c14n-20010315") and a transform is applied to remove name space information. As a rule, any binary values are always transformed into their base64 encoded values when represented in XML.

The Signature Element

The `Signature` element MUST contain a `DigestValue` element that is used for integrity checking. It MUST also contain a `SecurityToken` element that is used to indicate the security mode and token type, and to verify the signature. It MUST contain a SignatureValue element that contains the base64 encrypted value of the signature wrought on the hash UNLESS the security token type indicates Cleartext mode where a signature would be of no value with the encryption key being sent in the clear -- in this case we use the password itself for authentication).

```
<complexType name="SignatureType">
  <choice minOccurs="2" maxOccurs="3">
    <element ref="sssrmap:DigestValue" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:SignatureValue" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:SecurityToken" minOccurs="0" maxOccurs="1"/>
  </choice>
</complexType>

<element name="Signature" type="sssrmap:SignatureType"/>
```

The DigestValue Element

The `DigestValue` element contains the cryptographic digest of the message data. As described above, the hash is generated over the `Body` element. The data to be hashed must first be canonicalized and appropriately transformed before generating the digest since typically an application will read in the XML document into an internal binary form, then marshal (or serialize) the data into a string which is passed as input to the hash algorithm. Different implementations marshal the data differently so it is necessary to convert this to a well-defined format before generating the digest or the clients will generate different digest values for the same XML. The SHA-1 [SHA-1] message digest algorithm (http://www.w3.org/2000/09/xmldsig#sha1) SHALL be used as the default method for generating the digest. A *method* attribute is defined as an extensibility option in case an implementation wants to be able to specify alternate message digest algorithms.

It MAY have a method attribute:

Attribute	Description
method	The message digest algorithm. • A `method` attribute of "sha1" specifies the SHA-1 message digest algorithm. This is the default and is implied if this attribute is omitted.

```
<complexType name="DigestValueType">
  <simpleContent>
    <extension base="string">
      <attribute name="method" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="DigestValue" type="sssrmap:DigestValueType"/>
```

The SignatureValue Element

The SignatureValue element contains the digital signature that serves the authentication (and potentially non-repudiation) function. The string content of the SignatureValue element is a base64 encoding of the encrypted digest value. The HMAC algorithm [HMAC] based on the SHA1 message digest (http://www.w3.org/2000/09/xmldsig#hmac-sha1) SHALL be used as the default message authentication code algorithm for user identification and message integrity. A `method` attribute is defined as an extensibility option in case an implementation wants to be able to specify alternate digital signature algorithms.

It MAY have a method attribute:

Attribute	Description
method	The digest signature algorithm. • A `method` attribute of "hmac-sha1" specifies the HMAC SHA-1 digital signature algorithm. This is the default and is implied if this attribute is omitted.

```
<complexType name="SignatureValueType">
  <simpleContent>
    <extension base="string">
      <attribute name="method" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="SignatureValue" type="sssrmap:SignatureValueType"/>
```

Signature Example

Pre-authentication:

```
<Envelope>
  <Body>
    <Request action="Query" actor="kenneth">
      <Object>User</Object>
      <Get name="EmailAddress"></Get>
      <Where name="Name">scott</Where>
    </Request>
  </Body>
</Envelope>
```

Post-authentication:

```
<Envelope>
  <Signature>
    <DigestValue>
      LyLsF0Pi4wPU...
    </DigestValue>
    <SignatureValue>
      DJbchm5gK...
    </SignatureValue>
    <SecurityToken type="Asymmetric" name="kenneth">
      MIIEZzCCA9CggAwIBAgIQEmtJZc0rqrKh5i...
    </SecurityToken>
  </Signature>
  <Body>
    <Request action="Query" actor="kenneth">
      <Object>User</Object>
      <Get name="EmailAddress"></Get>
      <Where name="Name">scottmo</Where>
    </Request>
  </Body>
</Envelope>
```

7.3 Confidentiality

Confidentiality involves encrypting the sensitive data in the message, protecting exchanges against third-party interception and modification. Confidentiality is optional in an SSSRMAP message or reply. When confidentiality is required, SSSRMAP sessions use block cipher encryption with concepts borrowed from the emerging XML Encryption [XML_ENC] standard.

When confidentiality is used, encryption is performed over all child elements of the `Envelope` element, i.e. on the message data as well as any signature (The encrypted data is not signed -- rather the signature is encrypted). This data is replaced in-place within the envelope with an `EncryptedData` element. The data is first compressed using the gzip algorithm [ZIP]. Instead of encrypting this compressed data with the security token directly, a 192-bit random session key is generated by the sender and used to perform symmetric encryption on the compressed data. This key is itself encrypted with the security token and included with the encrypted data as the value of the `EncryptedKey` element as a child of the `EncryptedData` element. The ciphertext resulting from the data being encrypted with the session key is passed as the value of a `CipherValue` element (also a child of the `EncryptedData` element). As in the case with authentication, the security token itself is included as a child of the `Security` element within a `SecurityToken` element.

The EncryptedData Element

When SSSRMAP confidentiality is required, the `EncryptedData` element MUST appear as the only child element in the Envelope element. It directly replaces the contents of these elements including the data

and any digital signature. It MUST contain an `EncryptedKey` element that is used to encrypt the data. It MUST contain a `CipherValue` element that holds the base64 encoded ciphertext. It MAY also contain a `SecurityToken` element that is used to indicate the security mode and token type. If the `SecurityToken` element is omitted, a Symmetric key token type is assumed. Confidentiality is not used when a security token type of "Cleartext" is specified since it would be pointless to encrypt the data with the encryption key in the clear.

```
<complexType name="EncryptionDataType">
  <choice minOccurs="0" maxOccurs="1">
    <element ref="sssrmap:EncryptedKey" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:CipherValue" minOccurs="1" maxOccurs="1"/>
    <element ref="sssmap:SecurityToken" minOccurs="1" maxOccurs="1"/>
  </choice>
</complexType>

<element name="EncryptedData" type="sssrmap:EncryptedDataType"/>
```

The EncryptedKey Element

The `EncryptedKey` element is a random session key encrypted with the security token. This approach is used for a couple of reasons. In the case where public key encryption is used, asymmetric encryption is much slower than symmetric encryption and it makes sense to use a symmetric key for encryption and pass along it along by encrypting it with the recipient's public key. It is also useful in that the security token which does not change very often (compared to the session key which changes for every connection) is used on a very small sampling of data (the session key), whereas if it was used to encrypt the whole message an attacker could more effectively exploit an attack against the ciphertext. The CMS Triple DES Key Wrap algorithm "kw-tripledes" SHALL be used as the default method for key encryption. The session key is encrypted using the security token, base64 encoded and specified as the string content of the `EncryptedKey` element. A `method` attribute is defined as an extensibility option in case an implementation wants to be able to specify alternate key encryption algorithms.

It is REQUIRED that an implementation use a cryptographically secure Pseudo-Random number generator. It is RECOMMENDED that the session key be cryptographically generated (such as cyclic encryption, DES OFB, ANSI X9.17 PRNG, SHA1PRNG, or ANSI X12.17 (used by PGP)).

It MAY have a method attribute:

Attribute	Description
method	The key encryption algorithm. • A `method` attribute of "kw-tripledes" specifies the CMS Triple DES Key Wrap algorithm. This algorithm is specified by the XML Encryption [XML_ENC] URI "`http://www.w3.org/2001/04/xmlenc#kw-tripledes`". It involves two Triple DES encryptions, a random and known Initialization Vector (IV) and a CMS key checksum. A 192-bit key encryption key is generated from the security token, lengthened as necessary by zero-padding. No additional padding is performed in the encryptions. This is the default and is implied if this attribute is omitted.

```
<complexType name="EncryptedKeyType">
  <simpleContent>
    <extension base="string">
      <attribute name="method" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="EncryptedKey" type="sssrmap:EncryptedKeyType"/>
```

The CipherValue Element

The CipherValue element contains the message (and possibly signature) data encrypted with the random session key. The ciphertext is compressed using the gzip algorithm [ZIP], encrypted by the designated method, base64 encoded and included as the string content of the CipherValue element. The Triple DES algorithm with Cipher Block Chaining (CBC) feedback mode SHALL be used as the default method for encryption. A *method* attribute is defined as an extensibility option in case an implementation wants to be able to specify alternate data encryption algorithms.

It MAY have a method attribute:

Attribute	Description
method	The data encryption algorithm. • A `method` attribute of "tripledes-cbc" specifies the Triple DES algorithm with Cipher Block Chaining (CBC) feedback mode. This algorithm is specified by the XML Encryption [XML_ENC] URI identifier "http://www.w3.org/2001/04/xmlenc#tripledes-cbc". It specifies the use of a 192-bit encryption key and a 64-bit Initialization Vector (IV). Of the key bits, the first 64 are used in the first DES operation, the second 64 bits in the middle DES operation, and the third 64 bits in the last DES operation. The plaintext is first padded to a multiple of the block size (8 octets) using the padding scheme described in [XMLENC] for Block Encryption Algorithms (Padding per PKCS #5 will suffice for this). The resulting cipher text is prefixed by the IV. This is the default and is implied if this attribute is omitted.

```
<complexType name="CipherValueType">
  <simpleContent>
    <extension base="string">
      <attribute name="method" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="CipherValue" type="sssrmap:CipherValueType"/>
```

Encryption Example

In this example, a simple request is demonstrated without a digital signature for the sake of emphasizing the encryption plaintext replacement.

Pre-encryption:

```
<Envelope>
  <Body>
    <Response>
      <Status>true</Status>
```

```
      <Code>000</Code>
      <Count>1</Count>
      <Data>
        <User>
          <EmailAddress>Scott.Jackson@pnl.gov</EmailAddress>
        </User>
      </Data>
    </Response>
  </Body>
</Envelope>
```

Post-encryption:

```
<Envelope>
  <EncryptedData>
    <EncryptedKey>
      NAkE9iQofYhyOfiHZ29kkEFVJ30CAwEAAaMSM...
    </EncryptedKey>
    <CipherValue>
      mPCadVfOMx1NzDaKMHNgFkR9upTW4kgBxyPW...
    </CipherValue>
    <SecurityToken type="Asymmetric" name="kenneth">
      MIIEZzCCA9CggAwIBAgIQEmtJZc0rqrKh5i...
    </SecurityToken>
  </EncryptedData>
</Envelope>
```

8.0 Acknowledgements

9.0 References

[RFC2119] S. Bradner, "Key Words for Use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[BEEP] M. Rose, "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

[HMAC] H. Krawczyk, M. Bellare, R. Canetti, "HMAC, Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[SHA-1] U.S. Department of Commerce/National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-1.

[SSS] "Scalable Systems Software", http://www.scidac.org/ScalableSystems

[HTTP] "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616, June 1999.

[XML_CONV] "I-X and <I-N-CA> XML Conventions".

[FED_XML] "U.S. Federal XML Guidelines".

[RFC3117] M. Rose, "On the Design of Application Protocols", Informational RFC 3117, November 2001.

[XML_DSIG] D. Eastlake, J. Reagle Jr., D. Solo, "XML Signature Syntax and Processing", W3C Recommendation, 12 February 2002.

[XML_ENC] T. Imamura, B. Dillaway, E. Smon, "XML Encryption Syntax and Processing", W3C Candidate Recommendation, 4 March 2002.

[XRP] E. Brunner-Williams, A. Damaraju, N. Zhang, "Extensible Registry Protocol (XRP)", Internet Draft, expired August 2001.

[XML] Bray, T., et al, "Extensible Markup Language (XML) 1.0 (Second Edition)", 6 October 2000.

[XML_SCHEMA] D. Beech, M. Maloney, N. Mendelshohn, "XML Schema Part 1: Structures Working Draft", April 2000.

[ZIP] J. Gailly, M. Adler, "The gzip home page", http://www.gzip.org/

Appendix W: Moab Resource Manager Language Interface Overview

The Moab RM Language (formerly called WIKI) is the language that some resource managers use to communicate with Moab, specifically a native RM. Generally each line represents a single resource or workload in Moab. The line contains the name of the resource or workload followed by a set of *<attr>=<val>* pairs. Although the Moab RM language follows the same data format for all RMs, each RM type receives and returns it differently. For instructions and examples on using Moab RM language with SLURM or a native RM, see **W.2 Managing Resources with SLURM** on page 1122 and **Managing Resources Directly with the Native Interface** on page 566 respectively.

- W.1 Moab Resource Manager Language Data Format
- W.2 Managing Resources with SLURM
- W.3 Moab RM Language Socket Protocol Description

W.1 Moab Resource Manager Language Data Format

- W.1.1 Query Resources Data Format
- W.1.2 Query Workload Data Format

W.1.1 Query Resources Data Format

NAME	FORMAT	DEFAULT	DESCRIPTION
ADISK	*<INTEGER>*	0	Available local disk on node (in MB)
AFS	`<fs id="X" size="X" io="Y" rcount="X" wcount="X" ocount="X"></fs>[...]`	0	Available file system state

NAME	FORMAT	DEFAULT	DESCRIPTION
AMEMORY	*<INTEGER>*	0	Available/free RAM on node (in MB)
APROC	*<INTEGER>*	1	Available processors on node
ARCH	*<STRING>*	---	Compute architecture of node
ARES	one or more comma delimited *<NAME>*:*<VALUE>* pairs (i.e. `MATLAB:6,COMPILER:100`)	---	Arbitrary consumable resources currently available on the node
ASWAP	*<INTEGER>*	0	Available swap on node (in MB)
CCLASS	one or more bracket enclosed *<NAME>*:*<COUNT>* pairs (i.e. `[batch:5][sge:3]`)	---	Run classes supported by node. Typically, one class is 'consumed' per task. Thus, an 8 processor node may have 8 instances of each class it supports present, i.e. `[batch:8][interactive:8]`
CDISK	*<INTEGER>*	0	Configured local disk on node (in MB)
CFS	*<STRING>*	0	Configured file system state
CMEMORY	*<INTEGER>*	0	Configured RAM on node (in MB)
CONTAINERNODE	*<STRING>*	---	The physical machine that is hosting the virtual machine. Only valid on VMs.
CPROC	*<INTEGER>*	1	Configured processors on node
CPULOAD	*<DOUBLE>*	0.0	One minute BSD load average
CPUSPEED	*<INTEGER>*	---	The node's processor speed in MHz

NAME	FORMAT	DEFAULT	DESCRIPTION
CRES	one or more comma delimited *<NAME>*:*<VALUE>* pairs (i.e. MATLAB:6,COMPILER:100)	---	Arbitrary consumable resources supported and tracked on the node, i.e. software licenses or tape drives
CSWAP	*<INTEGER>*	0	Configured swap on node (in MB)
FEATURE	one or more colon delimited *<STRING>*'s (i.e. WIDE:HSM)	---	Generic attributes, often describing hardware or software features, associated with the node
GEVENT	GEVENT[*<EVENTNAME>*]=*<STRING>*	---	Generic event occurrence and context data
GMETRIC	GMETRIC[*<METRICNAME>*] =*<DOUBLE>*	---	Current value of generic metric, i.e., 'GMETRIC[temp]=103.5'.
IDLETIME	*<INTEGER>*	---	Number of seconds since last detected keyboard or mouse activity (often used with desktop harvesting)
MAXTASK	*<INTEGER>*	*<CPROC>*	Maximum number of tasks allowed on the node at any given time
NETADDR	*<STRING>*	---	The IP address of the machine
NODEINDEX	*<INTEGER>*	---	The node's index
OS	*<STRING>*	---	Operating system running on node
OSLIST	One or more comma delimited *<STRING>*'s with quotes if the string has spaces (i.e. "SAS7 AS3 Core Baseline Build v0.1.0","RedHat AS3-U5Development Build v0.2").	---	Operating systems accepted by node

NAME	FORMAT	DEFAULT	DESCRIPTION
OTHER	*<ATTR>=<VALUE>* [,*<ATTR>=<VALUE>*]...	---	Opaque node attributes assigned to node
PARTITION	*<STRING>*	DEFAULT	Partition to which node belongs
POWER	*<BOOLEAN>*		Whether the machine is on or off
PRIORITY	*<INTEGER>*	---	Node allocation priority
RACK	*<INTEGER>*	0	Rack location of the node
SLOT	*<INTEGER>*	0	Slot location of the node
STATE*	one of the following: `Idle`, `Running`, `Busy`, `Unknown`, `Drained`, `Draining`, or `Down`	Down	State of the node
UPDATETIME*	*<EPOCHTIME>*	0	Time node information was last updated
VARATTR	*<ATTR1>=<VAL1>* [+*<ATTR2>=<VAL2>*]...	---	Plus-delimited (+) list of *<ATTR>=<VAL>* pairs that jobs can request
VARIABLE	*<ATTR>=<VAL>*	---	Generic variables to be associated with node
VMOSLIST	*<STRING>*	---	Comma-delimited list (,) of supported virtual machine operating systems for this node
XRES	one or more comma delimited *<NAME>*:*<VALUE>* pairs (i.e. `MATLAB:6,COMPILER:100`)	---	Amount of external usage of a particular generic resource

* indicates required field

Node states have the following definitions:

State	Description
Busy	Node is running some jobs and will not accept additional jobs
Down	Resource Manager problems have been detected. Node is incapable of running jobs.
Draining	Node is responding but will not accept new jobs
Idle	Node is ready to run jobs but currently is not running any.
Running	Node is running some jobs and will accept additional jobs
Unknown	Node is capable of running jobs but the scheduler will need to determine if the node state is actually Idle, Running, or Busy.

W.1.2 Query Workload Data Format

NAME	FORMAT	DEFAULT	DESCRIPTION
ACCOUNT	*<STRING>*	---	AccountID associated with job
ARGS	*<STRING>*	---	job command-line arguments
COMMENT	*<STRING>*	0	job resource manager extension arguments including QoS, dependencies, reservation constraints, etc
COMPLETETIME*	*<EPOCHTIME>*	0	time job completed execution
DDISK	*<INTEGER>*	0	quantity of local disk space (in MB) which must be dedicated to each task of the job
DGRES	`name:value [,name:value]`	---	Dedicated generic resources per task.
DPROCS	*<INTEGER>*	1	number of processors dedicated per task
DSWAP	*<INTEGER>*	0	quantity of virtual memory (swap, in MB) which must be dedicated to each task of the job
ENDDATE	*<EPOCHTIME>*	[ANY]	time by which job must complete

NAME	FORMAT	DEFAULT	DESCRIPTION
ENV	*<STRING>*	---	job environment variables
ERROR	*<STRING>*	---	file to contain STDERR
EVENT	*<EVENT>*	---	event or exception experienced by job
EXEC	*<STRING>*	---	job executable command
EXITCODE	*<INTEGER>*	---	job exit code
FLAGS	*<STRING>*	---	job flags
GEOMETRY	*<STRING>*	---	String describing task geometry required by job
GNAME*	*<STRING>*	---	GroupID under which job will run
HOSTLIST	comma or colon delimited list of host names - suffix the host list with a carat (^) to mean superset; suffix with an asterisk (*) to mean subset; otherwise, the host list is interpreted as an exact set	`[ANY]`	list of required hosts on which job must run. (see TASKLIST) A subset means the specified host list is used first to select hosts for the job. If the job requires more hosts than are in the host list, they will be obtained from elsewhere if possible. If the job does not require all of the jobs in the host list, it will use only the ones it needs. A superset means the host list is the *only* source of hosts that should be considered for running the job. If the job can't find the necessary resources in the hosts in this list it should *not* run. No other hosts should be considered in allocating the job.
INPUT	*<STRING>*	---	file containing STDIN
IWD	*<STRING>*	---	job's initial working directory
NAME	*<STRING>*	---	User specified name of job
NODES	*<INTEGER>*	1	Number of nodes required by job (See Node Definition for more info)
OUTPUT	*<STRING>*	---	file to contain STDOUT

NAME	FORMAT	DEFAULT	DESCRIPTION
PARTITIONMASK	one or more colon delimited *<STRING>*s	[ANY]	list of partitions in which job can run
PREF	colon delimited list of *<STRING>*s	---	List of preferred node features or variables. (See PREF for more information.)
PRIORITY	*<INTEGER>*	---	system priority (absolute or relative - use '+' and '–' to specify relative)
QOS	*<INTEGER>*	0	quality of service requested
QUEUETIME*	*<EPOCHTIME>*	0	time job was submitted to resource manager
RARCH	*<STRING>*	---	architecture required by job
RCLASS	list of bracket enclosed *<STRING>* :*<INTEGER>* pairs	---	list of *<CLASSNAME>*:*<COUNT>* pairs indicating type and number of class instances required per task. (i.e. [batch:1] or [batch:2][tape:1])
RDISK	*<INTEGER>*	0	local disk space (in MB) required to be configured on nodes allocated to the job
RDISKCMP	one of >=, >, ==, <, or <=	>=	local disk comparison (i.e. node must have > 2048 MB local disk)
REJCODE	*<INTEGER>*	0	reason job was rejected
REJCOUNT	*<INTEGER>*	0	number of times job was rejected
REJMESSAGE	*<STRING>*	---	text description of reason job was rejected
REQRSV	<STRING>	---	Name of reservation in which job must run
RESACCESS	<STRING>	---	List of reservations in which job can run
RFEATURES	colon delimited list <STRING>'s	---	List of features required on nodes

NAME	FORMAT	DEFAULT	DESCRIPTION
RMEM	<INTEGER>	0	real memory (RAM, in MB) required to be configured on nodes allocated to the job
RMEMCMP	one of '>=', '>', '==', '<', or '<='	>=	real memory comparison (i.e. node must have >= 512MB RAM)
ROPSYS	<STRING>	---	operating system required by job
RSOFTWARE	<RESTYPE>[{+|:} <COUNT>] [@<TIMEFRAME>]	---	software required by job
RSWAP	<INTEGER>	0	virtual memory (swap, in MB) required to be configured on nodes allocated to the job
RSWAPCMP	one of '>=', '>', '==', '<', or '<='	>=	virtual memory comparison (i.e. node must have ==4096 MB virtual memory)
SID	<STRING>	---	system id (global job system owner)
STARTDATE	<EPOCHTIME>	0	earliest time job should be allowed to start
STARTTIME*	<EPOCHTIME>	0	time job was started by the resource manager
STATE*	one of Idle, Running, Hold, Suspended, Completed, or Removed	Idle	State of job
SUSPENDTIME	<INTEGER>	0	Number of seconds job has been suspended
TASKLIST	one or more comma-delimited <STRING>'s	---	list of **allocated** tasks, or in other words, comma-delimited list of node ID's associated with each active task of job (i.e., cl01, cl02, cl01, cl02, cl03) The tasklist is initially selected by the scheduler at the time the StartJob command is issued. The resource manager is then responsible for starting the job on these nodes and maintaining this task distribution information throughout the life of the job. (see HOSTLIST)

NAME	FORMAT	DEFAULT	DESCRIPTION
TASKS*	<INTEGER>	1	Number of tasks required by job (See Task Definition for more info)
TASKPERNODE	<INTEGER>	0	exact number of tasks required per node
UNAME*	<STRING>	---	UserID under which job will run
UPDATETIME*	<EPOCHTIME>	0	Time job was last updated
WCLIMIT*	[[HH:]MM:]SS	864000	walltime required by job

* indicates required field

Job states have the following definitions:

State	Definition
Completed	Job has completed
Hold	Job is in the queue but is not allowed to run
Idle	Job is ready to run
Removed	Job has been canceled or otherwise terminated externally
Running	Job is currently executing
Suspended	job has started but execution has temporarily been suspended

Completed and canceled jobs should be maintained by the resource manager for a brief time, perhaps 1 to 5 minutes, before being purged. This provides the scheduler time to obtain all final job state information for scheduler statistics.

Related topics

- Managing Resources with SLURM
- Managing Resources Directly with the Native Interface

W.2 Managing Resources with SLURM

This section demonstrates how Moab uses the Moab RM language (formerly called WIKI) to communicate with SLURM. For SLURM configuration instructions, see the Moab-SLURM Integration Guide.

- W.2.1 Commands
  - W.2.1.1 Resource Query
    - W.2.1.1.1 Query Resources Request Format
    - W.2.1.1.2 Query Resources Response Format
  - W.2.1.2 Workload Query
    - W.2.1.2.1 Query Workload Request Format
    - W.2.1.2.2 Query Workload Response Format
    - W.2.1.2.3 Query Workload Example
  - W.2.1.3 Start Job
  - W.2.1.4 Cancel Job
  - W.2.1.5 Suspend Job
  - W.2.1.6 Resume Job
  - W.2.1.7 Requeue Job
  - W.2.1.8 Signal Job
  - W.2.1.9 Modify Job
  - W.2.1.10 JobAddTask
  - W.2.1.11 JobRemoveTask
- W.2.2 Rejection Codes

W.2.1 Commands

All commands are requested via a socket interface, one command per socket connection. All fields and values are specified in ASCII text.

Supported Commands are:

- Query Resources
- Query Workload
- Start Job
- Cancel Job
- Suspend Job

- Resume Job
- Requeue Job
- JOBADDTASK
- JOBRELEASETASK

W.2.1.1 Moab RM Language Query Resources

W.2.1.1.1 Moab RM Language Query Resources Request Format

`CMD=GETNODES ARG={<UPDATETIME>:<NODEID>[:<NODEID>]... | <UPDATETIME>:ALL}`

Only nodes updated more recently than *<UPDATETIME>* will be returned where *<UPDATETIME>* is specified as the epoch time of interest. Setting *<UPDATETIME>* to `0` will return information for all nodes. Specify a colon delimited list of *NODEID*s if specific nodes are desired or use the keyword `ALL` to receive information for all nodes.

W.2.1.1.2 Moab RM Language Resources Response Format

The query resources response format is one or more line of the following format (separated with a new line):

`<NODEID><ATTR>=<VALUE>[;<ATTR>=<VALUE>]...`

<ATTR> is one of the names in the table below and the format of *<VALUE>* is dependent on *<ATTR>*.

Example 25-6: Moab RM language resource query and response

Request:

```
CMD=GETNODES ARG=0:node001:node002:node003
```

Response:

```
node001 UPDATETIME=963004212;STATE=Busy;OS=AIX43;ARCH=RS6000...
node002 UPDATETIME=963004213;STATE=Busy;OS=AIX43;ARCH=RS6000...
...
```

W.2.1.2 Moab RM Language Query Workload

W.2.1.2.1 Moab RM Language Query Workload Request Format

`CMD=GETJOBS ARG={<UPDATETIME>:<JOBID>[:<JOBID>]... | <UPDATETIME>:ALL }`

Only jobs updated more recently than *<UPDATETIME>* will be returned where *<UPDATETIME>* is specified as the epoch time of interest. Setting *<UPDATETIME>* to `0` will return information for all jobs. Specify a colon delimited list of JOBID's if information for specific jobs is desired or use the keyword `ALL` to receive information about all jobs.

W.2.1.2.2 Moab RM Language Query Workload Response Format

`SC=<STATUSCODE> ARG=<JOBCOUNT>#<JOBID>:<FIELD>=<VALUE>;[<FIELD>=<VALUE>;]...`
`[#<JOBID>:<FIELD>=<VALUE>;[<FIELD>=<VALUE>;]...]...`

or

SC=<*STATUSCODE*> RESPONSE=<*RESPONSE*>

FIELD is either the text name listed below or A<*FIELDNUM*>
(i.e. UPDATETIME or A2)

STATUSCODE values:

- 0 SUCCESS
- -1 INTERNAL ERROR

RESPONSE is a statuscode sensitive message describing error or state details.

W.2.1.2.3 Moab RM Language Query Workload Example

Request:

```
CMD=GETJOBS ARG=0:ALL
```

Response:

```
ARG=2#nebo3001.0:UPDATETIME=9780000320;STATE=Idle;WCLIMIT=3600;...
```

W.2.1.3 StartJob

The StartJob command may only be applied to jobs in the Idle state. It causes the job to begin running using the resources listed in the NodeID list.

send CMD=STARTJOB ARG=<*JOBID*> TASKLIST=<*NODEID*>[:<*NODEID*>]...

receive SC=<*STATUSCODE*> RESPONSE=<*RESPONSE*>

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message possibly further describing an error or state

Example 25-7: Job start

```
# Start job nebo.1 on nodes cluster001 and cluster002

# send
CMD=STARTJOB ARG=nebo.1 TASKLIST=cluster001:cluster002
# receive
SC=0;RESPONSE=job nebo.1 started with 2 tasks
```

W.2.1.4 CancelJob

The CancelJob command, if applied to an active job, will terminate its execution. If applied to an idle or active job, the CancelJob command will change the job's state to Canceled.

send CMD=CANCELJOB ARG=<*JOBID*> TYPE=<*CANCELTYPE*>

<*CANCELTYPE*> is one of the following:

ADMIN (command initiated by scheduler administrator)
WALLCLOCK (command initiated by scheduler because job exceeded its specified wallclock limit)

receive SC=<*STATUSCODE*> RESPONSE=<*RESPONSE*>

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message further describing an error or state

Example 25-8: Job cancel

```
# Cancel job nebo.2

# send
CMD=CANCELJOB ARG=nebo.2 TYPE=ADMIN'
# receive
SC=0 RESPONSE=job nebo.2 canceled
```

W.2.1.5 SuspendJob

The SuspendJob command can only be issued against a job in the state `Running`. This command suspends job execution and results in the job changing to the `Suspended` state.

send `CMD=SUSPENDJOB ARG=<JOBID>`

receive `SC=<STATUSCODE> RESPONSE=<RESPONSE>`

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message possibly further describing an error or state

Example 25-9: Job suspend

```
# Suspend job nebo.3

# send
CMD=SUSPENDJOB ARG=nebo.3
# receive
SC=0 RESPONSE=job nebo.3 suspended
```

W.2.1.6 ResumeJob

The ResumeJob command can only be issued against a job in the state `Suspended`. This command resumes a suspended job returning it to the `Running` state.

send `CMD=RESUMEJOB ARG=<JOBID>`

receive `SC=<STATUSCODE> RESPONSE=<RESPONSE>`

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message further describing an error or state

Example 25-10: Job resume

```
# Resume job nebo.3

# send
CMD=RESUMEJOB ARG=nebo.3
# receive
SC=0 RESPONSE=job nebo.3 resumed
```

W.2.1.7 RequeueJob

The RequeueJob command can only be issued against an active job in the state `Starting` or `Running`. This command requeues the job, stopping execution and returning the job to an idle state in the queue. The requeued job will be eligible for execution the next time resources are available.

send `CMD=REQUEUEJOB ARG=<JOBID>`

receive `SC=<STATUSCODE> RESPONSE=<RESPONSE>`

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message further describing an error or state

Example 25-11: job requeue

```
# Requeue job nebo.3

# send
CMD=REQUEUEJOB ARG=nebo.3
# receive
SC=0 RESPONSE=job nebo.3 requeued
```

W.2.1.8 SignalJob

The SignalJob command can only be issued against an active job in the state `Starting` or `Running`. This command signals the job, sending the specified signal to the master process. The signaled job will be remain in the same state it was before the signal was issued.

send `CMD=SIGNALJOB ARG=<JOBID> ACTION=signal VALUE=<SIGNAL>`

receive `SC=<STATUSCODE> RESPONSE=<RESPONSE>`

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message further describing an error or state

Example 25-12: Job signal

```
# Signal job nebo.3

# send
CMD=SIGNALJOB ARG=nebo.3 ACTION=signal VALUE=13
# receive
SC=0 RESPONSE=job nebo.3 signalled
```

W.2.1.9 ModifyJob

The ModifyJob command can be issued against any active or queued job. This command modifies specified attributes of the job.

send `CMD=MODIFYJOB ARG=<JOBID> [BANK=name] [NODES=num] [PARTITION=name] [TIMELIMIT=minutes]`

receive `SC=<STATUSCODE> RESPONSE=<RESPONSE>`

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message further describing an error or state

Example 25-13: Job modify

```
# Signal job nebo.3

# send
CMD=MODIFYJOB ARG=nebo.3 TIMELIMIT=9600
# receive
SC=0 RESPONSE=job nebo.3 modified
```

W.2.1.10 JobAddTask

The JobAddTask command allocates additional tasks to an active job.

send CMD=JOBADDTASK ARG=*<JOBID>* *<NODEID>* [*<NODEID>*]...

receive SC=*<STATUSCODE>* RESPONSE=*<RESPONSE>*

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message possibly further describing an error or state

Example 25-14: Job addtask

```
# Add 3 default tasks to job nebo30023.0 using resources located on nodes cluster002,
cluster016, and cluster112.

# send
CMD=JOBADDTASK ARG=nebo30023.0 DEFAULT cluster002 cluster016 cluster112
# receive
SC=0 RESPONSE=3 tasks added
```

W.2.1.11 JobRemoveTask

The JobRemoveTask command removes tasks from an active job.

send CMD=JOBREMOVETASK ARG=*<JOBID>* *<TASKID>* [*<TASKID>*]...

receive SC=*<STATUSCODE>* RESPONSE=*<RESPONSE>*

STATUSCODE >= 0 indicates SUCCESS
STATUSCODE < 0 indicates FAILURE
RESPONSE is a text message further describing an error or state

Example 25-15: Job removetask

```
# Free resources allocated to tasks 14, 15, and 16 of job nebo30023.0

# send
CMD=JOBREMOVETASK ARG=nebo30023.0 14 15 16
# receive
SC=0 RESPONSE=3 tasks removed
```

W.2.2 Rejection Codes

- 0xx - success - no error
  - 00x - success
    - 000 - success
  - 01x - usage/help reply
    - 010 - usage/help reply
  - 02x - status reply
    - 020 - general status reply
- 1xx - warning
  - 10x - general warning
    - 100 - general warning
  - 11x - no content
    - 110 - general wire protocol or network warning
    - 112 - redirect
    - 114 - protocol warning
  - 12x - no matching results
    - 120 - general message format warning
    - 122 - incomplete specification (best guess action/response applied)
  - 13x - security warning
    - 130 - general security warning
    - 132 - insecure request
    - 134 - insufficient privileges (response was censored/action reduced in scope)
  - 14x - content or action warning
    - 140 - general content/action warning
    - 142 - no content (server has processed the request but there is no data to be returned)
    - 144 - no action (no object to act upon)
    - 146 - partial content
    - 148 - partial action
  - 15x - component defined
  - 18x - application defined

- 2xx - wire protocol/network failure
    - 20x - protocol failure
        - 200 - general protocol/network failure
    - 21x - network failure
        - 210 - general network failure
        - 212 - cannot resolve host
        - 214 - cannot resolve port
        - 216 - cannot create socket
        - 218 - cannot bind socket
    - 22x - connection failure
        - 220 - general connection failure
        - 222 - cannot connect to service
        - 224 - cannot send data
        - 226 - cannot receive data
    - 23x - connection rejected
        - 230 - general connection failure
        - 232 - connection timed-out
        - 234 - connection rejected - too busy
        - 236 - connection rejected - message too big
    - 24x - malformed framing
        - 240 - general framing failure
        - 242 - malformed framing protocol
        - 244 - invalid message size
        - 246 - unexpected end of file
    - 25x - component defined
    - 28x - application defined
- 3xx - messaging format error
    - 30x - general messaging format error
        - 300 - general messaging format error
    - 31x - malformed XML document
        - 310 - general malformed XML error

    - 32x - XML schema validation error
        - 320 - general XML schema validation
    - 33x - general syntax error in request
        - 330 - general syntax error in response
        - 332 - object incorrectly specified
        - 334 - action incorrectly specified
        - 336 - option/parameter incorrectly specified
    - 34x - general syntax error in response
        - 340 - general response syntax error
        - 342 - object incorrectly specified
        - 344 - action incorrectly specified
        - 346 - option/parameter incorrectly specified
    - 35x - synchronization failure
        - 350 - general synchronization failure
        - 352 - request identifier is not unique
        - 354 - request id values do not match
        - 356 - request id count does not match
- 4xx - security error occurred
    - 40x - authentication failure - client signature
        - 400 - general client signature failure
        - 402 - invalid authentication type
        - 404 - cannot generate security token key - inadequate information
        - 406 - cannot canonicalize request
        - 408 - cannot sign request
    - 41x - negotiation failure
        - 410 - general negotiation failure
        - 412 - negotiation request malformed
        - 414 - negotiation request not understood
        - 416 - negotiation request not supported

    - 42x - authentication failure
        - 420 - general authentication failure
        - 422 - client signature failure
        - 424 - server authentication failure
        - 426 - server signature failure
        - 428 - client authentication failure
    - 43x - encryption failure
        - 430 - general encryption failure
        - 432 - client encryption failure
        - 434 - server decryption failure
        - 436 - server encryption failure
        - 438 - client decryption failure
    - 44x - authorization failure
        - 440 - general authorization failure
        - 442 - client authorization failure
        - 444 - server authorization failure
    - 45x - component defined failure
    - 48x - application defined failure
- 5xx - event management request failure
    - 50x - reserved
        - 500 - reserved
- 6xx - reserved for future use
    - 60x - reserved
        - 600 - reserved
- 7xx - server side error occurred
    - 70x - server side error
        - 700 - general server side error
    - 71x - server does not support requested function
        - 710 - server does not support requested function
    - 72x - internal server error
        - 720 - general internal server error

- 73x - resource unavailable
  - 730 - general resource unavailable error
  - 732 - software resource unavailable error
  - 734 - hardware resource unavailable error
- 74x - request violates policy
  - 740 - general policy violation
- 75x - component-defined failure
- 78x - application-defined failure

- 8xx - client side error occurred
  - 80x - general client side error
    - 800 - general client side error
  - 81x - request not supported
    - 810 - request not supported
  - 82x - application specific failure
    - 820 - general application specific failure

- 9xx - miscellaneous
  - 90x - general miscellaneous error
    - 900 - general miscellaneous error
  - 91x - general insufficient resources error
    - 910 - general insufficient resources error
  - 99x - general unknown error
    - 999 - unknown error

Related topics

- Moab Resource Manager Language Data Format
- Managing Resources Directly with the Native Interface

W.3 Moab RM Language Socket Protocol Description

Moab RM language is formerly known as WIKI. The Moab scheduler uses a simple protocol for socket connections to the user client and the resource manager as described below:

`<SIZE><CHAR>CK=<CKSUM><WS>TS=<TIMESTAMP><WS>AUTH=<AUTH><WS>DT=<DATA>`

Attribute	Description
<SIZE>	8 character decimal ASCII representation of the size of the packet following '*<SIZE><CHAR>*' Leading zeroes must be used to pad this value to 8 characters if necessary.
<CHAR>	A single ASCII character
<CKSUM>	A 16 character hexadecimal ASCII DES-based checksum calculated using the algorithm below* and *<SEED>* selected and kept secret by the site admins. The checksum is performed on the line from *TS=* to the end of the message including *<DATA>*.
<WS>	A series of white space characters consisting of either tabs and/or space characters.
<TIMESTAMP>	ASCII representation of epoch time
<AUTH>	Identifier of user requesting service (i.e., `USERNAME`)
<DT>	Data to be sent

An example header follows:

```
00001057 CK=cdf6d7a7ad45026f TS=922401962 AUTH=sched DT=<DATA>
```

where *<DATA>* is replaced by actual message data.

W.3.1 Checksum Algorithm ('C' version)

```
#define MAX_CKSUM_ITERATION 4

int GetChecksum(
 char *Buf,
 int BufSize,
 char *Checksum,
 char *CSKey) /* Note: pass in secret key */
 {
 unsigned int crc;
 unsigned int lword;
 unsigned int irword;
 int index;
 unsigned int Seed;
 Seed = (unsigned int)strtoul(CSKey,NULL,0);
 crc = 0;
 for (index = 0;index < BufSize;index++)
 {
 crc = (unsigned int)DoCRC((unsigned short)crc,Buf[index]);
 }
 lword = crc;
 irword = Seed;
 PSDES(&lword,&irword);
 sprintf(Checksum,"%08x%08x",
 lword,
```

```
 irword);
 return(SUCCESS);
 }


unsigned short DoCRC(
 unsigned short crc,
 unsigned char onech)
 {
 int index;
 unsigned int ans;
 ans = (crc ^ onech << 8);
 for (index = 0;index < 8;index++)
 {
 if (ans & 0x8000)
 ans = (ans <<= 1) ^ 4129;
 else
 ans <<= 1;
 }
 return((unsigned short)ans);
 }


int PSDES(
 unsigned int *lword,
 unsigned int *irword)
 {
 int index;
 unsigned int ia;
 unsigned int ib;
 unsigned int iswap;
 unsigned int itmph;
 unsigned int itmpl;
 static unsigned int c1[MAX_CKSUM_ITERATION] = {
 0xcba4e531, 0x537158eb, 0x145cdc3c, 0x0d3fdeb2 };
 static unsigned int c2[MAX_CKSUM_ITERATION] = {
 0x12be4590, 0xab54ce58, 0x6954c7a6, 0x15a2ca46 };
 itmph = 0;
 itmpl = 0;
 for (index = 0;index < MAX_CKSUM_ITERATION;index++)
 {
 iswap = *irword;
 ia = iswap ^ c1[index];
 itmpl = ia & 0xffff;
 itmph = ia >> 16;
 ib = (itmpl * itmpl) + ~(itmph*itmph);
 ia = (ib >> 16) | ((ib & 0xffff) << 16);
 *irword = (*lword) ^ ((ia ^ c2[index]) + (itmpl * itmph));
 *lword = iswap;
 }
 return(SUCCESS);
 }
```

W.3.2 Header Creation (PERL code)

(taken from PNNL's QBank client code)

```
##############################################################################
#
# subroutine wiki($COMMAND)
```

```
#
# Sends command to Moab server and returns the parsed result and status
#
###############################################################################
sub wiki
{
 my($COMMAND,$REQUEST,$result);
 my($sockaddr,$hostname);
 my($name,$aliases,$proto,$port,$type,$len,$thisaddr);
 my($thisport,$thatport,$response,$result);
 $COMMAND = shift;
 #
 # Establish socket connection
 #
 $sockaddr = 'S n a4 x8';
 chop ($hostname = `hostname`);
 ($name,$aliases,$proto)=getprotobyname('tcp');
 ($name,$aliases,$type,$len,$thisaddr)=gethostbyname($hostname);
 ($name,$aliases,$type,$len,$thataddr)=gethostbyname($BANKHOST);
 $thisport=pack($sockaddr, &AF_INET,0,$thisaddr);
 $thatport=pack($sockaddr, &AF_INET,$BANKPORT,$thataddr);
 socket(S, &PF_INET,&SOCK_STREAM,$proto) || die "cannot create socket\n";
 bind(S,$thisport) || die "cannot bind socket\n";
 connect(S,$thatport) || die "cannot connect socket\n";
 select(S); $| = 1; # Turn on autoflushing
 select(stdout); $| = 1; # Select STDOUT as default output
 #
 # Build and send command
 #
 $REQUEST="COMMAND=$COMMAND AUTH=$AUTH";
 chomp($CHECKSUM = `$QSUM "$REQUEST"`);
 $REQUEST .= " CHECKSUM=$CHECKSUM";
 my $command=pack "a8 a1 A*",sprintf("%08d",length($REQUEST))," ",$REQUEST;
 print S "$command"; # Send Command to server
 @REPLY=();
 while (<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;
}
```

W.3.3 Header Processing (PERL code)

```
sysread(NS,$length,8); # Read length string
sysread(NS,$delimiter,1); # Read delimiter byte
$DEBUG && print STDERR "length=[$length]\tdelimiter=[$delimiter]\n";
while($length) {
 $DEBUG && print STDERR "Awaiting $length bytes -- ".`date`;
 $length-=sysread(NS,$request,$length); # Read request
 sleep 1;
 }
%REQUEST=();
chomp($request);
foreach (@REQUEST=&shellwords($request)) # Parse arguments into array
 {
 ($key,$value)=split(/=/,$_);
 $REQUEST{$key}=$value unless defined $REQUEST{$key};
 }
$request =~ s/\s+CHECKSUM=.*//; # Strip off the checksum
print STDERR "REQUEST=$request\n";
```

```
chomp($checksum=`$QSUM "$request"`);
$me=$REQUEST{AUTH};
$command=$REQUEST{COMMAND};
if (!grep($command eq $_,@VALIDCMDS))
 { $REPLY = "STATUSCODE=0 RESULT=$command is not a valid command\n";}
elsif ($checksum ne $REQUEST{CHECKSUM})
 { $REPLY = "STATUSCODE=0 RESULT=Invalid Checksum\n";}
else
 { $REPLY = do $command(@REQUEST); }
$len=sprintf("%08d",length($REPLY)-1);
$delim=' ';
$DEBUG && print STDERR "REPLY=${len}${delim}$REPLY\n";
$buf="$len"."$delim"."$REPLY";
syswrite(NS,$buf,length($buf));
close NS;
```

SCHEDCFG flags

Flag	Description
AGGREGATENODEFEATURES	*AGGREGATENODEFEATURES* causes Moab to aggregate features reported by the different RMs. For example, if you have two RMs reporting different features for the same node, Moab will add both features together (instead of one being overwritten by the other). In order to set features manually, you can use mnodectl -m features (for details, see **mnodectl** on page 149).
ALLOWINFINITEJOBS	*ALLOWINFINITEJOBS* allows infinite wallclock times to be accepted. Previously, jobs with infinite job times were allowed by default.
ALLOWMULTICOMPUTE	*ALLOWMULTICOMPUTE* tells Moab how to resolve conflicting information from different resource managers. If *ALLOWMULTICOMPUTE* is specified, Moab will use the STATE and OS information from the resource manager that reports the node as online.
DISABLEPERJOBNODESETS	Disables a job's ability to override the system specified node set. See 13.3 Resource Manager Extensions for more information.
DISABLEPARTIALNODERESERVATIONS	Blocks partial node reservations.

Flag	Description
ENABLESLURMMEMPERCPU	By default Moab calls sbatch or srun with a `--mem=` request in a SLURM environment. When you set *ENABLESLURMMEMPERCPU*, Moab instead calls `--mem-per-cpu=`. This is to allow sites with policies that require the other parameter to use `--mem-per-cpu`.
ENFORCERESERVEDNODES	Without this flag Moab tries to optimize the reservation for a job before it starts, meaning a job may start on nodes that weren't part of its reservation. With this flag Moab tries to start jobs only on the nodes that were reserved.
ENFORCESAMENODESET	The same node set is not enforced across job requirements by default, rather each requirement is scheduled separately and the node sets are determined on a per-req basis. To have Moab enforce the same node set across all job requirements set this flag.
FASTGROUPLOOKUP	Moab will use the system call `getgrouplist` to gather group information. This can significantly improve performance on some LDAP systems.

Flag	Description
FASTRSVSTARTUP	Speeds up start time if there are existing reservations. *FASTRSVSTARTUP* is incompatible with partial node reservations. On very large systems, if there is a reservation in the checkpoint file on all the nodes, it would take a really long time for Moab to start up. For every node in the reservation, Moab checks every other node. With this flag, Moab just uses the nodelist that was checkpointed to create the reservation. It speeds up the startup process because it doesn't have to check every node. Where Moab would take 8 - 10 minutes to start up with an 18,000 node reservation without the flag, Moab can start up in 2-3 minutes with the flag. With the flag you will see one difference in checknode. A reservation that uses all the procs on a node initially shows that all the procs are blocked. Without the flag, and as jobs fill on the node, the blocked resources will be configured - dedicated (ex. 5/6). With the flag, the blocked resources will always be what the reservation is blocking and won't change when jobs fill on the node. Without flag: Reservations: brian.1x1 User -00:12:52 -> INFINITY (INFINITY) Blocked Resources@-00:00:02 Procs: 5/6 (83.33%) Mem: 0/5000 (0.00%) Blocked Resources@00:04:58 Procs: 6/6 (100.00%) Mem: 0/5000 (0.00%) m.2x1 Job:Running -00:00:02 -> 00:04:58 (00:05:00) Jobs: m.2 With flag: Reservations: brian.1x1 User -00:00:15 -> INFINITY (INFINITY) Blocked Resources@-00:00:02 Procs: 6/6 (100.00%) Mem: 0/5000 (0.00%) Blocked Resources@00:04:58 Procs: 6/6 (100.00%) Mem: 0/5000 (0.00%) m.1x1 Job:Running -00:00:02 -> 00:04:58 (00:05:00) Jobs: m.1

Flag	Description
	When you set the FASTRVSSTARTUP flag, Moab will also set the DISABLEPARTIALNODERESERVATIONS flag.
FILELOCKHA	This is a High Availability feature. *FILELOCKHA* prevents scheduling conflicts between multiple Moab servers.
FREECOMPLETEDJOBSUBMITSTRING	Moab frees the job submit string for completed jobs, decreasing the amount of memory needed during operation. This is useful in environments with large job scripts that can create a large memory footprint.
IGNOREPIDFILELOCK	Moab will not fail if it cannot get a lock on the `.moab.pid` file. This is useful when Moab is running on a shared file system where file locking can be unpredictable.
JOBSUSERSVWALLTIME	Allows jobs submitted without a walltime request or default walltime 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.
NOCLASSUPDATE	While running against TORQUE, Moab will not update classes when it refreshes each iteration. Moab loads the classes at startup, but does not refresh them until the next time it is restarted.

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:
	`mshow -a -w mintasks=1@procs:4+mem:4096`
	It is changed to this:
	`mshow -a -w mintasks=4@procs:1+,mem:1024,tpn=4`

Flag	Description
NOVMDESTROYDEPENDENCIES	The destroy job a in cloud workflow does not have any dependencies, allowing it to run whenever you cancel the service. For more information about destroy jobs, see **DESTROYTEMPLATE** on page 745 and **Creating a cloud workflow** on page 740.
OPTIMIZEDBACKFILL	On large systems that utilize system-wide reservations, backfill can take a considerable amount of time. This flag speeds up backfill scheduling by using an alternative BETA backfill algorithm. This flag will be the default in future versions of Moab.
OUTOFBANDVMRSV	Causes Moab to put reservations on untracked VMs (VMs that are created outside of Moab) so that the untracked VMs are visible when you use checkjob, mshow, checknode, etc. commands. These reservations are updated each new iteration, so if the VM is migrated or modified outside of Moab, the reservation will accordingly adjust automatically. The reservations are not checkpointed. The system re-creates them when you restart Moab (new reservation IDs, etc.). If a VM tracking job appears for the VM, then the reservation will be removed. This scheduler flag is highly recommended for Cloud users. For more information, see **About workload-driven cloud services** on page 733.
SHOWREQUESTEDPROCS	Shows requested processors regardless of NodeAccessPolicy in showq. When *SINGLEJOB* NODEACCESSPOLICY is used and the job requests one processor, showq displays the job with one processor.
SHOWUSERJOBSONLY	Causes Moab, when a non-admin user runs showq, to return only that user's jobs. If an administrator runs showq when this flag is set,Moab returns the jobs of all users; no restrictions are placed on administrators.
STRICTSPOOLDIRPERMISSIONS	Enforces at least a 511 permission on the Moab spool directory.

Flag	Description
UNMIGRATEONDEFER	Forces Moab to unmigrate a job if it enters a deferred state.