TORQUE 4.1.2

Administrator Guide

September 2012

© 2012 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 ..... **ix**
Introduction ix
TORQUE Administrator Guide overview xi

Chapter 1: Overview **1**
1.1 TORQUE installation overview 1
1.1.1 TORQUE architecture 1
1.1.2 Installing TORQUE 2
1.1.3 Compute nodes 3
1.1.4 Enabling TORQUE as a service 5
1.2 Initializing/Configuring TORQUE on the server (pbs_server) 5
1.2.1 Specifying compute nodes 7
1.2.2 Configuring TORQUE on compute nodes 8
1.2.3 Configuring Ports 9
1.2.4 Configuring trqauthd for client commands 10
1.2.5 Finalizing configurations 12
1.3 Advanced configuration 12
1.3.1 Customizing the install 12
1.3.2 Server configuration 19
1.4 Manual setup of initial server configuration 23
1.5 Server node file configuration 24
1.5.1 Basic node specification 25
1.5.2 Specifying virtual processor count for a node 25
1.5.3 Specifying GPU count for a node 25
1.5.4 Specifying node features (node properties) 26
1.6 Testing server configuration 26
1.7 TORQUE on NUMA systems 28
1.7.1 TORQUE NUMA configuration 28
1.7.2 Building TORQUE with NUMA support 28
1.8 TORQUE Multi-MOM 32
1.8.1 Multi-MOM configuration 32
1.8.2 Stopping pbs_mom in Multi-MOM mode 34

Chapter 2: Submitting and managing jobs **35**
2.1 Job submission 35
2.1.1 Multiple job submission 36
2.1.2 Managing multi-node jobs 37
2.1.3 Requesting resources 38

2.1.4 Requesting generic resources 42
2.1.5 Requesting floating resources 43
2.1.6 Requesting other resources 43
2.1.7 Exported batch environment variables 43
2.1.8 Enabling trusted submit hosts 45
2.1.9 Example submit scripts 45
2.2 Monitoring jobs 46
2.3 Canceling jobs 46
2.4 Job preemption 46
2.5 Keeping completed jobs 47
2.6 Job checkpoint and restart 47
2.6.1 Introduction to BLCR 48
2.6.2 Configuration files and scripts 48
2.6.3 Starting a checkpointable job 55
2.6.4 Checkpointing a job 56
2.6.5 Restarting a job 56
2.6.6 Acceptance tests 57
2.7 Job exit status 57
2.8 Service jobs 61
2.8.1 Submitting service jobs 62
2.8.2 Submitting service jobs in MCM 62
2.8.3 Managing service jobs 62

Chapter 3: Managing nodes 63
3.1 Adding nodes 63
3.2 Node properties 64
3.3 Changing node state 65
3.4 Host security 65
3.5 Linux cpuset support 66
3.6 Scheduling cores 68
3.6.1 Geometry request configuration 68
3.6.2 Geometry request usage 69
3.6.3 Geometry request considerations 69
3.7 Scheduling GPUs 69
3.7.1 Using GPUs with NUMA 70
3.7.2 TORQUE NVIDIA GPGPUs 71

Chapter 4: Setting server policies 75
4.1 Queue configuration 75
4.1.1 Queue attributes 76
4.1.2 Example queue configuration 85
4.1.3 Setting a default queue 85
4.1.4 Mapping a queue to subset of resources 86
4.1.5 Creating a routing queue 86

4.2 Server high availability ... 88

Chapter 5: Integrating schedulers for TORQUE ... 93

Chapter 6: Configuring data management ... 95

6.1 SCP setup ... 95

6.1.1 Generating SSH key on source host ... 96

6.1.2 Copying public SSH key to each destination host ... 96

6.1.3 Configuring the SSH daemon on each destination host ... 96

6.1.4 Validating correct SSH configuration ... 97

6.1.5 Enabling bi-directional SCP access ... 97

6.1.6 Compiling TORQUE to support SPC ... 97

6.1.7 Troubleshooting ... 98

6.2 NFS and other networked filesystems ... 98

6.3 File stage-in/stage-out ... 99

Chapter 7: MPI (Message Passing Interface) support ... 101

7.1 MPICH ... 101

7.2 MPICH-VMI ... 102

7.3 Open MPI ... 103

Chapter 8: Resources ... 105

Chapter 9: Accounting records ... 109

Chapter 10: Job logging ... 111

10.1 Job log location and name ... 111

10.2 Enabling job logs ... 111

Chapter 11: Troubleshooting ... 113

11.1 Host resolution ... 113

11.2 Firewall configuration ... 114

11.3 TORQUE log files ... 114

11.4 Using "tracejob" to locate job failures ... 114

11.5 Using GDB to locate job failures ... 117

11.6 Other diagnostic options ... 117

11.7 Stuck jobs ... 118

11.8 Frequently asked questions (FAQ) ... 119

11.9 Compute node health check ... 124

11.9.1 Configuring MOMs to launch a health check ... 125

11.9.2 Creating the health check script ... 125

11.9.3 Adjusting node state based on the health check output ... 125

11.9.4 Example health check script ... 126

11.10 Debugging ... 126

Appendices **133**
A.1 Commands overview 135
momctl 136
pbs_mom 141
pbs_server 151
pbs_track 154
pbsdsh 156
pbsnodes 157
qalter 159
qchkpt 167
qdel 168
qgpumode 170
qgpureset 170
qhold 171
qmgr 173
qrerun 176
qrls 177
qrun 179
qsig 180
qstat 182
qsub 189
qterm 207
B.1 Server parameters 209
C.1 Node manager (MOM) configuration 225
C.1.1 Parameters 225
C.1.2 Node features and generic consumable resource specification 239
C.1.3 Command-line arguments 240
D.1 Diagnostics and error codes 243
E.1 Considerations before upgrading 249
F.1 Large cluster considerations 251
F.1.1 Scalability guidelines 251
F.1.2 End user command caching 252
F.1.3 Other considerations 254
G.1 Prologue and epilogue scripts 257
G.1.1 Script order of execution 258
G.1.2 Script environment 258
G.1.3 Per job prologue and epilogue scripts 260
G.1.4 Prologue and epilogue scripts time out 260
G.1.5 Prologue error processing 261
H.1 Running multiple TORQUE servers and MOMs on the same node 265
I.1 Security overview 267
J.1 Job submission filter ("qsub wrapper") 269
K.1 "torque.cfg" configuration file 271
L.1 TORQUE Quick Start Guide 275

M.1 BLCR acceptance tests 279
M.1.1 Test environment 279
M.1.2 Test 1 - Basic operation 280
M.1.3 Test 2 - Persistence of checkpoint images 282
M.1.4 Test 3 - Restart after checkpoint 283
M.1.5 Test 4 - Multiple checkpoint/restart 284
M.1.6 Test 5 - Periodic checkpoint 284
M.1.7 Test 6 - Restart from previous image 285

Welcome

Welcome to the **TORQUE Administrator Guide**, version **4.1.2**. This guide is intended as a reference for both users and system administrators.

Note: Advanced TORQUE Administration is a video tutorial of a session offered at Moab Con that offers further details on advanced TORQUE administration.

For more information about this guide, see these topics:

- **TORQUE Administrator Guide overview** on page xi
- **Introduction** on page ix

Introduction

This section contains some basic introduction information to help you get started using TORQUE. It contains these topics:

- **What is a Resource Manager?** on page ix
- **What are Batch Systems?** on page ix
- **Basic Job Flow** on page x

What is a Resource Manager?

While TORQUE has a built-in scheduler, pbs_sched, it is typically used solely as a *resource manager* with a scheduler making requests to it. Resources managers provide the low-level functionality to start, hold, cancel, and monitor jobs. Without these capabilities, a scheduler alone cannot control jobs.

What are Batch Systems?

While TORQUE is flexible enough to handle scheduling a conference room, it is primarily used in batch systems. Batch systems are a collection of computers and other resources (networks, storage systems, license servers, and so forth) that operate under the notion that the whole is greater than the sum of the parts. Some batch systems consist of just a handful of machines running single-processor jobs, minimally managed by the users themselves. Other systems have thousands and thousands of machines executing users' jobs simultaneously while tracking software licenses and access to hardware equipment and storage systems.

Pooling resources in a batch system typically reduces technical administration of resources while offering a uniform view to users. Once configured properly, batch systems abstract away many of the details

involved with running and managing jobs, allowing higher resource utilization. For example, users typically only need to specify the minimal constraints of a job and do not need to know the individual machine names of each host on which they are running. With this uniform abstracted view, batch systems can execute thousands and thousands of jobs simultaneously.

Batch systems are comprised of four different components: (1) Master Node, (2) Submit/Interactive Nodes, (3) Compute Nodes, and (4) Resources.

Component	Description
Master Node	A batch system will have a master node where pbs_server runs. Depending on the needs of the systems, a master node may be dedicated to this task, or it may fulfill the roles of other components as well.
Submit/Interactive Nodes	Submit or interactive nodes provide an entry point to the system for users to manage their workload. For these nodes, users are able to submit and track their jobs. Additionally, some sites have one or more nodes reserved for interactive use, such as testing and troubleshooting environment problems. These nodes have client commands (such as qsub and qhold).
Computer Nodes	Compute nodes are the workhorses of the system. Their role is to execute submitted jobs. On each compute node, pbs_mom runs to start, kill, and manage submitted jobs. It communicates with pbs_server on the master node. Depending on the needs of the systems, a compute node may double as the master node (or more).
Resources	Some systems are organized for the express purpose of managing a collection of resources beyond compute nodes. Resources can include high-speed networks, storage systems, license managers, and so forth. Availability of these resources is limited and needs to be managed intelligently to promote fairness and increased utilization.

Basic Job Flow

The life cycle of a job can be divided into four stages: (1) creation, (2) submission, (3) execution, and (4) finalization.

Stage	Description

Creation	Typically, a submit script is written to hold all of the parameters of a job. These parameters could include how long a job should run (**walltime**), what resources are necessary to run, and what to execute. The following is an example submit file:
	`#PBS -N localBlast`
	`#PBS -S /bin/sh`
	`#PBS -l nodes=1:ppn=2,walltime=240:00:00`
	`#PBS -M user@my.organization.com`
	`#PBS -m ea`
	`source ~/.bashrc`
	`cd $HOME/work/dir`
	`sh myBlast.sh -i -v`
	This submit script specifies the name of the job (localBlast), what environment to use (`/bin/sh`), that it needs both processors on a single node (**nodes=1:ppn=2**), that it will run for at most 10 days, and that TORQUE should email "user@my.organization.com" when the job exits or aborts. Additionally, the user specifies where and what to execute.

Submission	A job is submitted with the qsub command. Once submitted, the policies set by the administration and technical staff of the site dictate the priority of the job and therefore, when it will start executing.
Execution	Jobs often spend most of their lifecycle executing. While a job is running, its status can be queried with qstat.
Finalilzation	When a job completes, by default, the `stdout` and `stderr` files are copied to the directory where the job was submitted.

Related topics

- **TORQUE Administrator Guide overview** on page xi

TORQUE Administrator Guide overview

Chapter 1: Overview on page 1 provides the details for installation and initialization, advanced configuration options, and (optional) qmgr option necessary to get the system up and running. System testing is also covered.

Chapter 2: Submitting and managing jobs on page 35 covers different actions applicable to jobs. The first section details how to submit a job and request resources (nodes, software licenses, and so forth), and provides several examples. Other actions include monitoring, canceling, preemption, and keeping completed jobs.

Chapter 3: Managing nodes on page 63 covers administrator tasks relating to nodes, which includes the following: adding nodes, changing node properties, and identifying state. Also an explanation of how to configure restricted user access to nodes is covered in **Host security** on page 65.

Chapter 4: Setting server policies on page 75 details server-side configurations of queue and high availability.

Chapter 5: Integrating schedulers for TORQUE on page 93 offers information about using the native scheduler versus an advanced scheduler.

Chapter 6: Configuring data management on page 95 deals with issues of data management. For non-network file systems, **SCP setup** on page 95 details setting up SSH keys and nodes to automate transferring data. **NFS and other networked filesystems** on page 98 covers configuration for these file systems. This chapter also addresses the use of file staging using the the **stagein** and **stageout** directives of the qsub command.

Chapter 7: MPI (Message Passing Interface) support on page 101 offers details supporting MPI.

Chapter 8: Resources on page 105 covers configuration, utilization, and states of resources.

Chapter 9: Accounting records on page 109 explains how jobs are tracked by TORQUE for accounting purposes.

Chapter 10: Job logging on page 111 explains how to enable job logs that contain information for completed jobs.

Chapter 11: Troubleshooting on page 113 is a guide that offers help with general problems. It includes FAQ and instructions for how to set up and use compute node checks. It also explains how to debug TORQUE.

The appendices provide tables of commands, parameters, configuration options, error codes, the Quick Start Guide, and so forth.

- A.1 Commands overview on page 135
- B.1 Server parameters on page 209
- C.1 Node manager (MOM) configuration on page 225
- D.1 Diagnostics and error codes on page 243
- E.1 Considerations before upgrading on page 249
- F.1 Large cluster considerations on page 251
- G.1 Prologue and epilogue scripts on page 257
- H.1 Running multiple TORQUE servers and MOMs on the same node on page 265
- I.1 Security overview on page 267
- J.1 Job submission filter ("qsub wrapper") on page 269
- K.1 "torque.cfg" configuration file on page 271
- L.1 TORQUE Quick Start Guide on page 275
- M.1 BLCR acceptance tests on page 279

Related topics

- **Introduction** on page ix

Chapter 1: Overview

This section contains some basic information about TORQUE, including how to install and configure it on your system. For details, see these topics:

- **TORQUE installation overview** on page 1
- **Initializing/Configuring TORQUE on the server (pbs_server)** on page 5
- **Advanced configuration** on page 12
- **Manual setup of initial server configuration** on page 23
- **Server node file configuration** on page 24
- **Testing server configuration** on page 26
- **TORQUE on NUMA systems** on page 28
- **TORQUE Multi-MOM** on page 32

1.1 TORQUE installation overview

This section contains information about TORQUE architecture and explains how to install TORQUE. It also describes how to install tpackages on compute nodes and how to enable TORQUE as a service.

For details, see these topics:

- **TORQUE architecture** on page 1
- **Installing TORQUE** on page 2
- **Compute nodes** on page 3
- **Enabling TORQUE as a service** on page 5

Related topics

- **Troubleshooting** on page 113

1.1.1 TORQUE architecture

A TORQUE cluster consists of one head node and many compute nodes. The head node runs the pbs_server daemon and the compute nodes run the pbs_mom daemon. Client commands for submitting and managing jobs can be installed on any host (including hosts not running pbs_server or pbs_mom).

The head node also runs a scheduler daemon. The scheduler interacts with pbs_server to make local policy decisions for resource usage and allocate nodes to jobs. A simple FIFO scheduler, and code to construct more advanced schedulers, is provided in the TORQUE source distribution. Most TORQUE users choose to use a packaged, advanced scheduler such as Maui or Moab.

Users submit jobs to pbs_server using the qsub command. When pbs_server receives a new job, it informs the scheduler. When the scheduler finds nodes for the job, it sends instructions to run the job with the node list to pbs_server. Then, pbs_server sends the new job to the first node in the node list and instructs it to launch the job. This node is designated the execution host and is called *Mother Superior*. Other nodes in a job are called *sister moms*.

Related topics

- **TORQUE installation overview** on page 1
- **Installing TORQUE** on page 2

1.1.2 Installing TORQUE

Build the distribution on the machine that will act as the TORQUE server - the machine which monitors and controls all compute nodes by running the pbs_server daemon.

The built distribution package works only on compute nodes of a similar architecture. Nodes with different architecture must have the installation package built on them individually.

To install TORQUE

1. Download the TORQUE distribution file from http://www.adaptivecomputing.com/resources/downloads/torque/. You can also download source code using Subversion from the repository at `svn://clusterresources.com/torque/`. Use the following command to display all branches:

   ```
   svn list svn://clusterresources.com/torque/
   ```

   If you're checking source out from SVN, read the `README.building-40` file in the repository.

2. Extract the packaged file and navigate to the unpackaged directory.

   ```
   > tar -xzvf torque-4.1.2.tar.gz
   > cd torque-4.1.2/
   ```

3. Do the following to configure the package:

   a. Run `./configure`. TORQUE has many options you can specify to configure your installation. For more information, see **Customizing the install** on page 12.

   By default, `make install` installs all files in `/usr/local/bin`, `/usr/local/lib`, `/usr/local/sbin`, `/usr/local/include`, and `/usr/local/man` . You can specify an installation prefix other than `/usr/local` by using `--prefix` as an argument to `./configure`. Note that TORQUE cannot be installed into a directory path that contains a space.

```
> ./configure
```

You may need the `libssl-devel` and `libxml2-devel` packages to build 4.1.2. The package names may vary.

4. Run `make` and `make install`.

   TORQUE must be installed by a root user. If running `sudo` fails, switch to root with `su-`.

   ```
   > make
   > sudo make install
   ```

   OSX 10.4 users need to change **#define __TDARWIN** in `src/include/pbs_config.h` to **#define __TDARWIN_8**.

5. After installation, verify that you have the **PATH** environment variable configured to include `/usr/local/bin/` and `/usr/local/sbin/`.

   By default, `make install` creates a directory at `/var/spool/torque`. This directory is referred to as TORQUE_HOME. TORQUE_HOME has several sub-directories, including `server_priv/`, `server_logs/`, `mom_priv/`, `mom_logs/`, and other directories used in the configuration and running of TORQUE.

6. Verify you have environment variables configured so your system can find the shared libraries and binary files for TORQUE.

   To set the library path, add the directory where the TORQUE libraries are installed. For example, if your TORQUE libraries are installed in `/usr/local/lib` (if you changed the default library path at configure time, use that path instead.), execute the following:

   ```
   > echo '/usr/local/lib' > /etc/ld.so.conf.d/torque.conf
   > ldconfig
   ```

While Adaptive Computing distributes the RPM files as part of the build, it does not support those files. Not every Linux distribution uses RPM. Adaptive Computing provides a single solution using `make` and `make install` that works across all Linux distributions and most UNIX systems. We recognize the RPM format provides many advantages for deployment but it is up to the individual site to repackage the TORQUE installation to match their individual needs.

Related topics

- **TORQUE installation overview** on page 1
- **Compute nodes** on page 3

1.1.3 Compute nodes

Use the Cluster Resources tpackage system to create self-extracting tarballs which can be distributed and installed on compute nodes. The tpackages are customizable. See the `INSTALL` file for additional options

and features.

To create tpackages

1. Configure and make as normal, and then run `make packages`.

```
> make packages
Building ./torque-package-clients-linux-i686.sh ...
Building ./torque-package-mom-linux-i686.sh ...
Building ./torque-package-server-linux-i686.sh ...
Building ./torque-package-gui-linux-i686.sh ...
Building ./torque-package-devel-linux-i686.sh ...
Done.

The package files are self-extracting packages that can be copied and executed on
your production machines. Use --help for options.
```

2. Copy the desired packages to a shared location.

```
> cp torque-package-mom-linux-i686.sh /shared/storage/
> cp torque-package-clients-linux-i686.sh /shared/storage/
```

3. Install the tpackages on the compute nodes.

   Cluster Resources recommends that you use a remote shell, such as SSH, to install tpackages on remote systems. Set up shared SSH keys if you do not want to supply a password for each host.

   > The only required package for the compute node is mom-linux. Additional packages are recommended so you can use client commands and submit jobs from compute nodes.

   The following is an example of how to copy and install mom-linux in a distributed fashion.

```
> for i in node01 node02 node03 node04 ; do scp torque-package-mom-linux-i686.sh
${i}:/tmp/. ; done
> for i in node01 node02 node03 node04 ; do scp torque-package-clients-linux-i686.sh
${i}:/tmp/. ; done
> for i in node01 node02 node03 node04 ; do ssh ${i} /tmp/torque-package-mom-linux-
i686.sh --install ; done
> for i in node01 node02 node03 node04 ; do ssh ${i} /tmp/torque-package-clients-
linux-i686.sh --install ; done
```

Alternatively, you can use a tool like xCAT instead of dsh.

To use a tool like xCAT

1. Copy the tpackage to the nodes.

```
> prcp torque-package-linux-i686.sh noderange:/destinationdirectory/
```

2. Install the tpackage.

```
> psh noderange /tmp/torque-package-linux-i686.sh --install
```

Although optional, it is possible to use the TORQUE server as a compute node and install a pbs_mom with the pbs_server daemon.

Related topics

- **Installing TORQUE** on page 2
- **TORQUE installation overview** on page 1

1.1.4 Enabling TORQUE as a service

Enabling TORQUE as a service is optional. In order to run TORQUE as a service, you must enable running client commands (for instructions, see **Configuring trqauthd for client commands** on page 10.

The method for enabling TORQUE as a service is dependent on the Linux variant you are using. Startup scripts are provided in the `contrib/init.d/` directory of the source package.

- RedHat (as root)

```
> cp contrib/init.d/pbs_mom /etc/init.d/pbs_mom
> chkconfig --add pbs_mom
```

- SuSE (as root)

```
> cp contrib/init.d/suse.pbs_mom /etc/init.d/pbs_mom
> insserv -d pbs_mom
```

- Debian (as root)

```
> cp contrib/init.d/debian.pbs_mom /etc/init.d/pbs_mom
> update-rc.d pbs_mom defaults
```

You will need to customize these scripts to match your system.

These options can be added to the self-extracting packages. For more details, see the `INSTALL` file.

Related topics

- **TORQUE installation overview** on page 1

1.2 Initializing/Configuring TORQUE on the server (pbs_server)

The TORQUE server (pbs_server) contains all the information about a cluster. It knows about all of the MOM nodes in the cluster based on the information in the `$TORQUE_HOME/server_priv/nodes` file (See **Configuring TORQUE on compute nodes** on page 8). It also maintains the status of each MOM node through updates from the MOMs in the cluster (see **pbsnodes** on page 157). All jobs are submitted via qsub to the server, which maintains a master database of all jobs and their states.

Schedulers such as Moab Workload Manager receive job, queue, and node information from pbs_server and submit all jobs to be run to pbs_server.

The server configuration is maintained in a file named serverdb, located in $TORQUE_HOME/server_priv. The serverdb file contains all parameters pertaining to the operation of TORQUE plus all of the queues which are in the configuration. For pbs_server to run, serverdb must be initialized.

You can initialize serverdb in two different ways, but the recommended way is to use the ./torque.setup script:

- Execute ./torque.setup from the build directory (see **./torque.setup** on page 6).
- Use pbs_server -t create (see **Initializing/Configuring TORQUE on the server (pbs_server)** on page 5).

Restart pbs_server after initializing serverdb.

```
> qterm
> pbs_server
```

./torque.setup

The torque.setup script uses pbs_server -t create to initialize serverdb and then adds a user as a manager and operator of TORQUE and other commonly used attributes. The syntax is as follows:

/torque.setup *username*

```
> ./torque.setup ken
> qmgr -c 'p s'

#
# Create queues and set their attributes.
#
#
# Create and define queue batch
#
create queue batch set queue batch queue_type = Execution
set queue batch resources_default.nodes = 1
set queue batch resources_default.walltime = 01:00:00
set queue batch enabled = True
set queue batch started = True
#
# Set server attributes.
#
set server scheduling = True
set server acl_hosts = kmn
set server managers = ken@kmn
set server operators = ken@kmn
set server default_queue = batch
set server log_events = 511
set server mail_from = adm
set server scheduler_iteration = 600
set server node_check_rate = 150
set server tcp_timeout = 6
set server mom_job_sync = True
set server keep_completed = 300
```

pbs_server -t create

The -t create option instructs pbs_server to create the serverdb file and initialize it with a minimum configuration to run pbs_server.

```
> pbs_server -t create
```

To see the configuration and verify that TORQUE is configured correctly, use **qmgr**:

```
> qmgr -c 'p s'

#
# Set server attributes.
#
set server acl_hosts = kmn
set server log_events = 511
set server mail_from = adm
set server scheduler_iteration = 600
set server node_check_rate = 150
set server tcp_timeout = 6
```

A single queue named batch and a few needed server attributes are created.

This section contains these topics:

- **Specifying compute nodes** on page 7
- **Configuring TORQUE on compute nodes** on page 8
- **Finalizing configurations** on page 12

Related topics

- **Node manager (MOM) configuration** on page 225
- **Advanced configuration** on page 12

1.2.1 Specifying compute nodes

The environment variable *TORQUE_HOME* is where configuration files are stored. For TORQUE 2.1 and later, *TORQUE_HOME* is `/var/spool/torque/`. For earlier versions, *TORQUE_HOME* is `/usr/spool/PBS/`.

The pbs_server must recognize which systems on the network are its compute nodes. Specify each node on a line in the server's nodes file. This file is located at *TORQUE_HOME*`/server_priv/nodes`. In most cases, it is sufficient to specify just the names of the nodes on individual lines; however, various properties can be applied to each node.

Only a root user can access the `server_priv` directory.

Syntax of nodes file:

```
node-name[:ts] [np=] [gpus=] [properties]
```

- The **node-name** must match the hostname on the node itself, including whether it is fully qualified or shortened.
- The **[:ts]** option marks the node as timeshared. Timeshared nodes are listed by the server in the node status report, but the server does not allocate jobs to them.
- The **[np=]** option specifies the number of virtual processors for a given node. The value can be less than, equal to, or greater than the number of physical processors on any given node.
- The **[gpus=]** option specifies the number of GPUs for a given node. The value can be less than, equal to, or greater than the number of physical GPUs on any given node.

- The node processor count can be automatically detected by the TORQUE server if **auto_node_np** is set to TRUE. This can be set using this command:

  ```
  qmgr -c set server auto_node_np = True
  ```

  Setting **auto_node_np** to TRUE overwrites the value of np set in `TORQUE_HOME/server_priv/nodes`.

- The **[properties]** option allows you to specify arbitrary strings to identify the node. Property strings are alphanumeric characters only and must begin with an alphabetic character.
- Comment lines are allowed in the nodes file if the first non-white space character is the pound sign (#).

The following example shows a possible node file listing.

`TORQUE_HOME/server_priv/nodes`:

```
# Nodes 001 and 003-005 are cluster nodes
#
node001 np=2 cluster01 rackNumber22
#
# node002 will be replaced soon
node002:ts waitingToBeReplaced
# node002 will be replaced soon
#
node003 np=4 cluster01 rackNumber24
node004 cluster01 rackNumber25
node005 np=2 cluster01 rackNumber26 RAM16GB
node006
node007 np=2
node008:ts np=4
...
```

Related topics

- **Initializing/Configuring TORQUE on the server (pbs_server)** on page 5

1.2.2 Configuring TORQUE on compute nodes

If using TORQUE self extracting packages with default compute node configuration, no additional steps are required and you can skip this section.

If installing manually, or advanced compute node configuration is needed, edit the *TORQUE_HOME*`/mom_priv/config` file on each node. The recommended settings follow.

`TORQUE_HOME/mom_priv/config`:

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

This file is identical for all compute nodes and can be created on the head node and distributed in parallel to all systems.

Related topics

- **Initializing/Configuring TORQUE on the server (pbs_server)** on page 5

1.2.3 Configuring Ports

You can configure the various ports that TORQUE uses for communication. Most ports can be configured multiple ways. The ports you can configure are:

- pbs_server listening port
- pbs_mom listening port
- port pbs_server uses to communicate to the pbs_mom
- port pbs_mom uses to communicate to the pbs_server
- port client commands use to communicate to the pbs_server
- port trqauthd uses to communicate to the pbs_server

If you are running pbspro on the same system, be aware that it uses the same environment variables and `/etc/services` entries.

Configuring the pbs_server listening port

To configure the port the `pbs_server` listens on, follow any of these steps:

- Set an environment variable called `PBS_BATCH_SERVICE_PORT` to the port desired.
- Edit the `/etc/services` file and set `pbs port_num/tcp`.
- Start `pbs_server` with the `-p` option.

```
$ pbs_server -p port_num
```

- Edit the `$PBS_HOME/server_name` file and change `server_name` to `server_name:<port_num>`
- Start `pbs_server` with the `-H` option.

```
$ pbs_server -H server_name:port_num
```

Configuring the pbs_mom listening port

To configure the port the `pbs_mom` listens on, follow any of these steps:

- Set an environment variable called `PBS_MOM_SERVICE_PORT` to the port desired.
- Edit the `/etc/services` file and set `pbs_mom port_num/tcp`.
- Start `pbs_mom` with the `-M` option.

```
$ pbs_mom -M port_num
```

- Edit the `nodes` file entry for that list: add `mom_service_port=port_num`.

Configuring the port pbs_server uses to communicate with pbs_mom

To configure the port the `pbs_server` uses to communicate with `pbs_mom`, follow any of these steps:

- Set an environment variable called PBS_MOM_SERVICE_PORT to the port desired.
- Edit the /etc/services file and set pbs_mom port_num/tcp.
- Start pbs_mom with the -M option.

```
$ pbs_server -M port_num
```

Configuring the port pbs_mom uses to communicate with pbs_server

To configure the port the pbs_mom uses to communicate with pbs_server, follow any of these steps:

- Set an environment variable called PBS_BATCH_SERVICE_PORT to the port desired.
- Edit the /etc/services file and set pbs port_num/tcp.
- Start pbs_mom with the -S option.

```
$ pbs_mom -p port_num
```

- Edit the nodes file entry for that list: add mom_service_port=port_num.

Configuring the port client commands use to communicate with pbs_server

To configure the port client commands use to communicate with pbs_server, follow any of these steps:

- Edit the /etc/services file and set pbs port_num/tcp.
- Edit the $PBS_HOME/server_name file and change server_name to server_name:<*port_num*>

Configuring the port trqauthd uses to communicate with pbs_server

To configure the port trqauthd uses to communicate with pbs_server, follow any of these steps:

- Edit the $PBS_HOME/server_name file and change server_name to server_name:<*port_num*>

Related topics

- **Initializing/Configuring TORQUE on the server (pbs_server)** on page 5
- pbs_server
- pbs_mom
- trqauthd
- client commands

1.2.4 Configuring trqauthd for client commands

trqauthd is a daemon used by TORQUE client utilities to authorize user connections to pbs_server. Once started, it remains resident. TORQUE client utilities then communicate with trqauthd on port 15005 on the loopback interface. It is multi-threaded and can handle large volumes of simultaneous requests.

Running trqauthd

trqauthd must be run as root. It must also be running on any host where TORQUE client commands will execute.

By default, trqauthd is installed to `/usr/local/bin`.

trqauthd can be invoked directly from the command line or by the use of init.d scripts which are located in the `contrib/init.d` directory of the TORQUE source.

There are three `init.d` scripts for trqauthd in the `contrib/init.d` directory of the TORQUE source tree:

Script	Description
debian.trqauthd	Used for apt-based systems (debian, ubuntu are the most common variations of this)
suse.trqauthd	Used for suse-based systems
trqauthd	An example for other package managers (Redhat, Scientific, CentOS, and Fedora are some common examples)

You should edit these scripts to be sure they will work for your site.

Inside each of the scripts are the variables PBS_DAEMON and PBS_HOME. These two variables should be updated to match your TORQUE installation. PBS_DAEMON needs to point to the location of trqauthd. PBS_HOME needs to match your TORQUE installation.

Choose the script that matches your dist system and copy it to `/etc/init.d`. If needed, rename it to **trqauthd**.

To start the daemon

```
/etc/init.d/trqauthd start
```

To stop the daemon

```
/etc/init.d/trqauthd stop
```

OR

```
service trqauthd start/stop
```

Related topics

- **Initializing/Configuring TORQUE on the server (pbs_server)** on page 5

1.2.5 Finalizing configurations

After configuring the `serverdb` and the `server_priv/nodes` files, and after ensuring minimal MOM configuration, restart the `pbs_server` on the server node and the `pbs_mom` on the compute nodes.

Compute Nodes:

```
> pbs_mom
```

Server Node:

```
> qterm -t quick
> pbs_server
```

After waiting several seconds, the `pbsnodes -a` command should list all nodes in state `free`.

Related topics

- **Initializing/Configuring TORQUE on the server (pbs_server)** on page 5

1.3 Advanced configuration

This section contains information about how you can customize the installation and configure the server to ensure that the server and nodes are communicating correctly. For details, see these topics:

- **Customizing the install** on page 12
- **Server configuration** on page 19

Related topics

- **Server parameters** on page 209

1.3.1 Customizing the install

The TORQUE `configure` command has several options available. Listed below are some suggested options to use when running `./configure`.

- By default, TORQUE does not install the admin manuals. To enable this, use `--enable-docs`.
- By default, TORQUE does not enable **syslog** usage. To enable this, use `--enable-syslog`.

Table 1-1: Optional Features

Option	Description
--disable-clients	Directs TORQUEnot to build and install the TORQUE client utilities such as qsub, qstat, qdel, etc.

Option	Description
--disable-FEATURE	Do not include FEATURE (same as --enable-FEATURE=no).
--disable-libtool-lock	Avoid locking (might break parallel builds).
--disable-mom	Do not include the MOM daemon.
--disable-mom-check-spool	Don't check free space on spool directory and set an error.
--disable-posix-memlock	Disable the moms use of mlockall. Some versions of OSs seem to have buggy POSIX MEMLOCK.
--disable-priv-ports	Disable the use of privileged ports for authentication. Some versions of OSX have a buggy bind() and cannot bind to privileged ports.
--disable-qsub-keep-override	Do not allow the qsub -k flag to override -o -e.
--disable-server	Do not include server and scheduler.
--disable-shell-pipe	Give the job script file as standard input to the shell instead of passing its name via a pipe.
--disable-spool	If disabled, TORQUE will create output and error files directly in $HOME/.pbs_spool if it exists or in $HOME otherwise. By default, TORQUE will spool files in TORQUE_HOME/spool and copy them to the users home directory when the job completes.
--disable-xopen-networking	With HPUX and GCC, don't force usage of XOPEN and libxnet.
--enable-acct-x	Enable adding x attributes to accounting log.
--enable-array	Setting this under IRIX enables the SGI Origin 2000 parallel support. Normally autodetected from the /etc/config/array file.

Option	Description
--enable-auto-run	Allows jobs to run automatically as soon as they are queued if resources are available (available in TORQUE 2.3.1 and later).
--enable-blcr	Enable BLCR support.
--enable-cpa	Enable Cray's CPA support.
--enable-cpu-set	Enable Linux 2.6 kernel cpusets. It is recommended that you turn on this feature to prevent a job from expanding across more CPU cores than it is assigned.
--enable-debug	Prints debug information to the console for pbs_server and pbs_mom while they are running.. (This is different than **--with-debug** which will compile with debugging symbols.)
--enable-dependency-tracking	Do not reject slow dependency extractors.
--enable-drmaa	Build the DRMAA 1.0 library.
--enable-fast-install[=PKGS]	Optimize for fast installation [default=yes].
--enable-FEATURE[=ARG]	Include FEATURE [ARG=yes].
--enable-file-sync	Open files with sync on each write operation. This has a negative impact on TORQUE performance. This is disabled by default.
--enable-force-nodefile	Forces creation of nodefile regardless of job submission parameters. Not on by default.
--enable-gcc-warnings	Enable gcc strictness and warnings. If using gcc, default is to error on any warning.
--enable-geometry-requests	TORQUE is compiled to use procs_bitmap during job submission.

Option	Description
--enable-gui	Include the GUI-clients.
--enable-maintainer-mode	This is for the autoconf utility and tells autoconf to enable so called rebuild rules. See maintainer mode for more information.
--enable-maxdefault	Turn on the RESOURCEMAXDEFAULT flag. Versions of TORQUE earlier than 2.4.5 attempted to apply queue and server defaults to a job that didn't have defaults specified. If a setting still did not have a value after that, TORQUE applied the queue and server maximum values to a job (meaning, the maximum values for an applicable setting were applied to jobs that had no specified or default value). In TORQUE 2.4.5 and later, the queue and server maximum values are no longer used as a value for missing settings. To re-enable this behavior in TORQUE 2.4.5 and later, use `--enable-maxdefault`.
--enable-nochildsignal	Turn on the NO_SIGCHLD flag.
--enable-nodemask	Enable nodemask-based scheduling on the Origin 2000.
--enable-pemask	Enable pemask-based scheduling on the Cray T3e.
--enable-plock-daemons[=ARG]	Enable daemons to lock themselves into memory: logical-or of 1 for pbs_server, 2 for pbs_scheduler, 4 for pbs_mom (no argument means 7 for all three).
--enable-quickcommit	Turn on the QUICKCOMMIT flag.
--enable-shared[=PKGS]	Build shared libraries [default=yes].
--enable-shell-use-argv	Enable this to put the job script name on the command line that invokes the shell. Not on by default. Ignores --enable-shell-pipe setting.
--enable-sp2	Build PBS for an IBM SP2.
--enable-srfs	Enable support for SRFS on Cray.

Option	Description
--enable-static [=PKGS]	Build static libraries [default=yes].
--enable-sys-log	Enable (default) the use of syslog for error reporting.
--enable-tcl-qstat	Setting this builds qstat with Tcl interpreter features. This is enabled if Tcl is enabled.
--enable-unix-sockets	Enable the use of Unix Domain sockets for authentication.

Table 1-2: Optional packages

Option	Description
--with-blcr=DIR	BLCR installation prefix (Available in versions 2.5.6 and 3.0.2 and later).
--with-blcr-include=DIR	Include path for libcr.h (Available in versions 2.5.6 and 3.0.2 and later).
--with-blcr-lib=DIR	Lib path for libcr (Available in versions 2.5.6 and 3.0.2 and later).
--with-blcr-bin=DIR	Bin path for BLCR utilities (Available in versions 2.5.6 and 3.0.2 and later).
--with-cpa-include=DIR	Include path for cpalib.h.
--with-cpa-lib=DIR	Lib path for libcpalib.
--with-debug	Compile with debugging symbols.
--with-default-server=HOSTNAME	Set the name of the computer that clients will access when no machine name is specified as part of the queue name. It defaults to the hostname of the machine on which PBS is being compiled.
--with-environ=PATH	Set the path containing the environment variables for the daemons. For SP2 and AIX systems, suggested setting is to /etc/environment. Defaults to the file "pbs_environment" in the server-home. Relative paths are interpreted within the context of the server-home.
--with-gnu-ld	Assume the C compiler uses GNU ld [default=no].

Option	Description
--with-mail-domain=MAILDOMAIN	Override the default domain for outgoing mail messages, i.e. "user@maildomain". The default maildomain is the hostname where the job was submitted from.
--with-modulefiles[=DIR]	Use module files in specified directory [/etc/modulefiles].
--with-momlogdir	Use this directory for MOM logs.
--with-momlogsuffix	Use this suffix for MOM logs.
--without-PACKAGE	Do not use PACKAGE (same as --with-PACKAGE=no).
--without-readline	Do not include readline support (default: included if found).
--with-PACKAGE[=ARG]	Use PACKAGE [ARG=yes].
--with-pam=DIR	Directory that holds the system PAM modules. Defaults to /lib(64)/security on Linux.
--with-pic	Try to use only PIC/non-PIC objects [default=use both].
--with-qstatrc-file=FILE	Set the name of the file that qstat will use if there is no ".qstatrc" file in the directory where it is being invoked. Relative path names will be evaluated relative to the server home directory (see above). If this option is not specified, the default name for this file will be set to "qstatrc" (no dot) in the server home directory.
--with-rcp	One of "scp", "rcp", "mom_rcp", or the fullpath of a remote file copy program. scp is the default if found, otherwise mom_rcp is used. Some rcp programs don't always exit with valid error codes in case of failure. mom_rcp is a copy of BSD rcp included with this source that has correct error codes, but it is also old, unmaintained, and doesn't have largefile support.
--with-sched=TYPE	Sets the scheduler type. If TYPE is "c", the scheduler will be written in C. If TYPE is "tcl" the server will use a Tcl based scheduler. If TYPE is "basl", TORQUE will use the rule based scheduler. If TYPE is "no", then no scheduling is done. "c" is the default.
--with-sched-code=PATH	Sets the name of the scheduler to use. This only applies to BASL schedulers and those written in the C language. For C schedulers this should be a directory name and for BASL schedulers a filename ending in ".basl". It will be interpreted relative to srctree/src/schedulers.SCHD_TYPE/samples. As an example, an appropriate BASL scheduler realtive path would be "nas.basl". The default scheduler code for "C" schedulers is "fifo".

Option	Description
--with-scp	In TORQUE 2.1 and later, SCP is the default remote copy protocol. See **--with-rcp** if a different protocol is desired.
--with-sendmail[=FILE]	Sendmail executable to use.
--with-server-home=DIR	Set the server home/spool directory for PBS use. Defaults to /var/spool/torque.
--with-server-name-file=FILE	Set the file that will contain the name of the default server for clients to use. If this is not an absolute pathname, it will be evaluated relative to the server home directory that either defaults to /usr/spool/PBS or is set using the --with-server-home option to configure. If this option is not specified, the default name for this file will be set to "server_name".
--with-tcl	Directory containing tcl configuration (tclConfig.sh).
--with-tclatrsep=CHAR	Set the Tcl attribute separator character this will default to "." if unspecified.
--with-tclinclude	Directory containing the public Tcl header files.
--with-tclx	Directory containing tclx configuration (tclxConfig.sh).
--with-tk	Directory containing tk configuration (tkConfig.sh).
--with-tkinclude	Directory containing the public Tk header files.
--with-tkx	Directory containing tkx configuration (tkxConfig.sh).
--with-tmpdir=DIR	Set the tmp directory that pbs_mom will use. Defaults to "/tmp". This is a Cray-specific feature.
--with-xauth=PATH	Specify path to xauth program.

HAVE_WORDEXP

`Wordxp()` performs a shell-like expansion, including environment variables. By default, `HAVE_WORDEXP` is set to **1** in `src/pbs_config.h`. If set to 1, will limit the characters that can be used in a job name to those allowed for a file in the current environment, such as BASH. If set to 0, any valid character for the file system can be used.

If a user would like to disable this feature by setting `HAVE_WORDEXP` to 0 in `src/include/pbs_config.h`, it is important to note that the error and the output file names will not expand environment variables, including `$PBS_JOBID`. The other important consideration is that characters that BASH dislikes, such as **()**, will not be allowed in the output and error file names for jobs by default.

Related topics

- **Advanced configuration** on page 12
- **Server configuration** on page 19

1.3.2 Server configuration

See these topics for details:

- **Server configuration overview** on page 19
- **Name service configuration** on page 19
- **Configuring job submission hosts** on page 19
- **Configuring TORQUE on a multi-homed server** on page 21
- **Architecture specific notes** on page 21
- **Specifying non-root administrators** on page 21
- **Setting up email** on page 21
- **Using MUNGE authentication** on page 22
- **Setting up the MOM hierarchy** on page 22

Server configuration overview

There are several steps to ensure that the server and the nodes are completely aware of each other and able to communicate directly. Some of this configuration takes place within TORQUE directly using the qmgr command. Other configuration settings are managed using the pbs_server `nodes` file, DNS files such as `/etc/hosts` and the `/etc/hosts.equiv` file.

Name service configuration

Each node, as well as the server, must be able to resolve the name of every node with which it will interact. This can be accomplished using `/etc/hosts`, DNS, NIS, or other mechanisms. In the case of `/etc/hosts`, the file can be shared across systems in most cases.

A simple method of checking proper name service configuration is to verify that the server and the nodes can "ping" each other.

Configuring job submission hosts

Using RCmd authentication

When jobs can be submitted from several different hosts, these hosts should be trusted via the R* commands (such as rsh and rcp). This can be enabled by adding the hosts to the /etc/hosts.equiv file of the machine executing the pbs_server daemon or using other R* command authorization methods. The exact specification can vary from OS to OS (see the man page for **ruserok** to find out how your OS validates remote users). In most cases, configuring this file is as simple as adding a line to your `/etc/hosts.equiv` file, as in the following:

`/etc/hosts.equiv`:

```
#[+ | -] [hostname] [username]
mynode.myorganization.com
.....
```

Either of the hostname or username fields may be replaced with a wildcard symbol (+). The (+) may be used as a stand-alone wildcard but not connected to a username or hostname, e.g., `+node01` or `+user01`. However, a (-) may be used in that manner to specifically exclude a user.

Following the Linux man page instructions for hosts.equiv may result in a failure. You cannot precede the user or hostname with a (+). To clarify, `node1 +user1` will not work and **user1** will not be able to submit jobs.

For example, the following lines will not work or will not have the desired effect:

```
+node02 user1
node02 +user1
```

These lines will work:

```
node03 +
+ jsmith
node04 -tjones
```

The most restrictive rules must precede more permissive rules. For example, to restrict user tsmith but allow all others, follow this format:

```
node01 -tsmith
node01 +
```

Please note that when a hostname is specified, it must be the fully qualified domain name (FQDN) of the host. Job submission can be further secured using the server or queue **acl_hosts** and **acl_host_enabled** parameters (for details, see **Queue attributes** on page 76).

Using the "submit_hosts" service parameter

Trusted submit host access may be directly specified without using RCmd authentication by setting the server submit_hosts parameter via **qmgr** as in the following example:

```
> qmgr -c 'set server submit_hosts = host1'
> qmgr -c 'set server submit_hosts += host2'
> qmgr -c 'set server submit_hosts += host3'
```

Use of **submit_hosts** is potentially subject to DNS spoofing and should not be used outside of controlled and trusted environments.

Allowing job submission from compute hosts

If preferred, all compute nodes can be enabled as job submit hosts without setting `.rhosts` or `hosts.equiv` by setting the allow_node_submit parameter to **true**.

Configuring TORQUE on a multi-homed server

If the pbs_server daemon is to be run on a multi-homed host (a host possessing multiple network interfaces), the interface to be used can be explicitly set using the SERVERHOST parameter.

Architecture specific notes

With some versions of Mac OS/X, it is required to add the line `$restricted *.<DOMAIN>` to the pbs_mom configuration file. This is required to work around some socket bind bugs in the OS.

Specifying non-root administrators

By default, only root is allowed to start, configure and manage the pbs_server daemon. Additional trusted users can be authorized using the parameters **managers** and **operators**. To configure these parameters use the **qmgr** command, as in the following example:

```
> qmgr
Qmgr: set server managers += josh@*.fsc.com
Qmgr: set server operators += josh@*.fsc.com
```

All manager and operator specifications must include a user name and either a fully qualified domain name or a host expression.

To enable all users to be trusted as both operators and administrators, place the **+** (plus) character on its own line in the server_priv/acl_svr/operators and server_priv/acl_svr/managers files.

Setting up email

Moab relies on emails from TORQUE about job events. To set up email, do the following:

To set up email

1. Use the `--with-sendmail` configure option at configure time. TORQUE needs to know where the email application is. If this option is not used, TORQUE tries to find the sendmail executable. If it isn't found, TORQUE cannot send emails.

   ```
   > ./configure --with-sendmail=<path_to_executable>
   ```

2. Set mail_domain in your server settings. If your domain is clusterresources.com, execute:

   ```
   > qmgr -c 'set server mail_domain=clusterresources.com'
   ```

3. (Optional) You can override the default mail_body_fmt and mail_subject_fmt values via **qmgr**:

   ```
   > qmgr -c 'set server mail_body_fmt=Job: %i \n Name: %j \n On host: %h \n \n %m \n \n
   %d'
   > qmgr -c 'set server mail_subject_fmt=Job %i - %r'
   ```

By default, users receive e-mails on job aborts. Each user can select which kind of e-mails to receive by using the qsub -m option when submitting the job. If you want to dictate when each user should receive e-mails, use a submit filter (for details, see **Job submission filter ("qsub wrapper")** on page 269).

Using MUNGE authentication

MUNGE is an authentication service that creates and validates user credentials. It was developed by Lawrence Livermore National Laboratoy (LLNL) to be highly scalable so it can be used in large environments such as HPC clusters. To learn more about MUNGE and how to install it, see http://code.google.com/p/munge/.

Configuring TORQUE to use MUNGE is a compile time operation. When you are building TORQUE, use `–enable-munge-auth` as a command line option with `./configure`.

```
> ./configure –enable-munge-auth
```

You can use only one authorization method at a time. If `–enable-munge-auth` is configured, the privileged port ruserok method is disabled.

TORQUE does not link any part of the MUNGE library into its executables. It calls the MUNGE and UNMUNGE utilities which are part of the MUNGE daemon. The MUNGE daemon must be running on the server and all submission hosts. The TORQUE client utilities call MUNGE and then deliver the encrypted credential to pbs_server where the credential is then unmunged and the server verifies the user and host against the authorized users configured in serverdb.

Authorized users are added to serverdb using qmgr and the **authorized_users** parameter. The syntax for **authorized_users** is `authorized_users=`*<user>*`@`*<host>*. To add an authorized user to the server you can use the following qmgr command:

```
> qmgr -c 'set server authorized_users=user1@hosta
> qmgr -c 'set server authorized_users+=user2@hosta
```

The previous example adds user1 and user2 from hosta to the list of authorized users on the server. Users can be removed from the list of authorized users by using the **-=** syntax as follows:

```
> qmgr -c 'set server authorized_users-=user1@hosta
```

Users must be added with the <user>@<host> syntax. The user and the host portion can use the '*' wildcard to allow multiple names to be accepted with a single entry. A range of user or host names can be specified using a [a-b] syntax where *a* is the beginning of the range and *b* is the end.

```
> qmgr -c 'set server authorized_users=user[1-10]@hosta
```

This allows user1 through user10 on hosta to run client commands on the server.

Setting up the MOM hierarchy

The MOM hierarchy allows you to override the compute nodes' default behavior of reporting status updates directly to the pbs_server. Instead, you configure compute nodes so that each node sends its status update information to another compute node. The compute nodes pass the information up a tree or hierarchy until eventually the information reaches a node that will pass the information directly to pbs_server. This can significantly reduce traffic and time required to keep the cluster status up to date.

The name of the file that contains the configuration information is named mom_hierarchy. By default, it is located in the `/var/spool/torque/server_priv` directory. The file uses syntax similar to XML:

```
<path>
  <level> comma-separated node list </level>
```

```
  <level> comma-separated node list </level>
  ...
</path>
...
```

The **<path></path>** tag pair identifies a group of compute nodes. The **<level></level>** tag pair contains a comma-separated list of compute node names. Multiple paths can be defined with multiple levels within each path.

Within a **<path>** tag pair the levels define the hierarchy. All nodes in the top level communicate directly with the server. All nodes in lower levels communicate to the first available node in the level directly above it. If the first node in the upper level goes down, the nodes in the subordinate level will then communicate to the next node in the upper level. If no nodes are available in an upper level then the node will communicate directly to the server.

If an upper level node has fallen out and then becomes available, the lower level nodes will eventually find that the node is available and start sending their updates to that node.

If you want to specify MOMs on a different port than the default, you must list the node in the form: *hostname:mom_manager_port*.

For example:

```
<path>
  <level>hostname:mom_manager_port</level>
  ...
</path>
...
```

Related topics

- **Advanced configuration** on page 12

1.4 Manual setup of initial server configuration

On a new installation of TORQUE, the server database must be initialized using the command `pbs_server -t create`. This command creates a file in *$TORQUEHOME*`/server_priv` named `serverdb` which contains the server configuration information.

The following output from qmgr shows the base configuration created by the command `pbs_server -t create`:

```
qmgr -c 'p s'
#
Set server attributes.
#
set server acl_hosts = kmn
set server log_events = 511
set server mail_from = adm
set server scheduler_iteration = 600
set server node_check_rate = 150
set server tcp_timeout = 6
```

This is a bare minimum configuration and it is not very useful. By using qmgr, the server configuration can be modified to set up TORQUE to do useful work. The following qmgr commands will create a queue and enable the server to accept and run jobs.

```
pbs_server -t create
qmgr -c "set server scheduling=true"
qmgr -c "create queue batch queue_type=execution"
qmgr -c "set queue batch started=true"
qmgr -c "set queue batch enabled=true"
qmgr -c "set queue batch resources_default.nodes=1"
qmgr -c "set queue batch resources_default.walltime=3600"
qmgr -c "set server default_queue=batch"
```

These commands must be executed by root.

In this example, the configuration database is initialized and the scheduling interface is activated (using '`scheduling=true`'). This option allows the scheduler to receive job and node events which allow it to be more responsive. The next command creates a queue and specifies the queue type. Within PBS, the queue must be declared an '`execution` queue in order for it to run jobs. Additional configuration (i.e., setting the queue to `started` and `enabled`) allows the queue to *accept* job submissions, and *launch* queued jobs.

The next two lines are optional, setting default `node` and `walltime` attributes for a submitted job. These defaults will be picked up by a job if values are not explicitly set by the submitting user. The final line, `default_queue=batch`, is also a convenience line and indicates that a job should be placed in the `batch` queue unless explicitly assigned to another queue.

Additional information on configuration can be found in the admin manual and in the **qmgr** main page.

Related topics

- **TORQUE installation overview** on page 1

1.5 Server node file configuration

This section contains information about configuring server node files. It explains how to specify node virtual processor counts and GPU counts, as well as how to specify node features or properties. For details, see these topics:

- **Basic node specification** on page 25
- **Specifying virtual processor count for a node** on page 25
- **Specifying GPU count for a node** on page 25
- **Specifying node features (node properties)** on page 26

Related topics

- **TORQUE installation overview** on page 1
- **Server parameters** on page 209
- Moab node feature overview

1.5.1 Basic node specification

For the pbs_server to communicate with each of the moms, it needs to know which machines to contact. Each node that is to be a part of the batch system must be specified on a line in the server `nodes` file. This file is located at `TORQUE_HOME/server_priv/nodes`. In most cases, it is sufficient to specify just the node name on a line as in the following example:

`server_priv/nodes`:

```
node001
node002
node003
node004
```

Related topics

- **Server node file configuration** on page 24

1.5.2 Specifying virtual processor count for a node

By default each node has one virtual processor. Increase the number using the **np** attribute in the nodes file. The value of np can be equal to the number of physical cores on the node or it can be set to a value which represents available "execution slots" for the node. The value used is determined by the administrator based on hardware, system, and site criteria.

The following example shows how to set the np value in the nodes file. In this example, we are assuming that node001 and node002 has four physical cores. The administrator wants the value of np for node001 to reflect that it has four cores. However, node002 will be set up to handle multiple virtual processors without regard to the number of physical cores on the system.

`server_priv/nodes`:

```
node001 np=2
node002 np=12
...
```

Related topics

- **Server node file configuration** on page 24

1.5.3 Specifying GPU count for a node

Administrators can manually set the number of GPUs on a node or if they are using NVIDIA GPUs and drivers, they can have them detected automatically. For more information about how to set up TORQUE

with NVIDIA GPUS, see TORQUE NVIDIA GPGPUS.

To manually set the number of GPUs on a node, use the gpus attribute in the nodes file. The value of GPUs is determined by the administrator based on hardware, system, and site criteria.

The following example shows how to set the GPU value in the nodes file. In the example, we assume node01 and node002 each have two physical GPUs. The administrator wants the value of node001 to reflect the physical GPUs available on that system and adds gpus=2 to the nodes file entry for node001. However, node002 will be set up to handle multiple virtual GPUs without regard to the number of physical GPUs on the system.

server_priv/nodes:

```
node001 gpus=1
node002 gpus=4
...
```

Related topics

- **Server node file configuration** on page 24

1.5.4 Specifying node features (node properties)

Node features can be specified by placing one or more white space-delimited strings on the line for the associated host as in the following example:

server_priv/nodes:

```
node001 np=2 fast ia64
node002 np=4 bigmem fast ia64 smp
...
```

These features can be used by users to request specific nodes when submitting jobs. For example:

```
qsub -l nodes=1:bigmem+1:fast job.sh
```

This job submission will look for a node with the bigmem feature (node002) and a node with the fast feature (either node001 or node002).

Related topics

- **Server node file configuration** on page 24

1.6 Testing server configuration

If you have initialized TORQUE using the torque.setup script or started TORQUE using pbs_server -t create and pbs_server is still running, terminate the server by calling qterm. Next, start pbs_server again without the -t create arguments. Follow the script below to verify your server configuration. The output for the examples below are based on the nodes file example in Specifying node features and Server configuration.

```
# verify all queues are properly configured
> qstat -q

server:kmn

Quere   Memory   CPU Time   Walltime   Node   Run  Que  Lm  State
-----   ------   --------   --------   ----   ---  ---  --  -----
batch   --        --        --         --     0    0    --  ER
                                             ---  ---
                                             0    0

# view additional server configuration
> qmgr -c 'p s'
#
# Create queues and set their attributes
#
#
# Create and define queue batch
#
create queue batch
set queue batch queue_type = Execution
set queue batch resources_default.nodes = 1
set queue batch resources_default.walltime = 01:00:00
set queue batch enabled = True
set queue batch started = True
#
# Set server attributes.
#
set server scheduling = True
set server acl_hosts = kmn
set server managers = user1@kmn
set server operators = user1@kmn
set server default_queue = batch
set server log_events = 511
set server mail_from = adm
set server scheduler_iteration = 600
set server node_check_rate = 150
set server tcp_timeout = 300
set server job_stat_rate = 45
set server poll_jobs = True
set server mom_job_sync = True
set server keep_completed = 300
set server next_job_number = 0

# verify all nodes are correctly reporting
> pbsnodes -a
node001
  state=free
  np=2
  properties=bigmem,fast,ia64,smp
  ntype=cluster
  status=rectime=1328810402,varattr=,jobs=,state=free,netload=6814326158,gres=,
loadave=0.21,ncpus=6,physmem=8193724kb,
availmem=13922548kb,totmem=16581304kb,idletime=3,nusers=3,nsessions=18,sessions=1876
1120 1912 1926 1937 1951 2019 2057 28399 2126 2140 2323 5419 17948 19356 27726 22254
29569,uname=Linux kmn 2.6.38-11-generic #48-Ubuntu SMP Fri Jul 29 19:02:55 UTC 2011 x86_
64,opsys=linux
  mom_service_port = 15002
  mom_manager_port = 15003
  gpus = 0
# submit a basic job - DO NOT RUN AS ROOT
> su - testuser
> echo "sleep 30" | qsub

# verify jobs display
> qstat

Job id   Name   User      Time Use  S  Queue
```

```
------   -----  ----      --------  -- -----
0.kmn    STDIN  knielson         0  Q  batch
```

At this point, the job should be in the **Q** state and will not run because a scheduler is not running yet. TORQUE can use its native scheduler by running pbs_sched or an advanced scheduler (such as Moab Workload Manager). See Integrating schedulers for details on setting up an advanced scheduler.

Related topics

- **TORQUE installation overview** on page 1

1.7 TORQUE on NUMA systems

Starting in TORQUE version 3.0, TORQUE can be configured to take full advantage of Non-Uniform Memory Architecture (NUMA) systems. The following instructions are a result of development on SGI Altix and UV hardware.

For details, see these topics:

- **TORQUE NUMA configuration** on page 28
- **Building TORQUE with NUMA support** on page 28

1.7.1 TORQUE NUMA configuration

There are three steps to configure TORQUE to take advantage of NUMA architectures:

1. Configure TORQUE with `--enable-numa-support`.
2. Create the `mom_priv/mom.layout` file.
3. Configure `server_priv/nodes`.

Related topics

- **TORQUE on NUMA systems** on page 28

1.7.2 Building TORQUE with NUMA support

To turn on NUMA support for TORQUE the `--enable-numa-support` option must be used during the configure portion of the installation. In addition to any other configuration options, add the `--enable-numa-support` option as indicated in the following example:

```
$ ./configure --enable-numa-support
```

Don't use MOM hierarchy with NUMA.

When TORQUE is enabled to run with NUMA support, there is only a single instance of pbs_mom (MOM) that is run on the system. However, TORQUE will report that there are multiple nodes running in the

cluster. While pbs_mom and pbs_server both know there is only one instance of pbs_mom, they manage the cluster as if there were multiple separate MOM nodes.

The mom.layout file is a virtual mapping between the system hardware configuration and how the administrator wants TORQUE to view the system. Each line in mom.layout equates to a node in the cluster and is referred to as a NUMA node.

Automatically Creating mom.layout (Recommended)

A perl script named mom_gencfg is provided in the contrib/ directory that generates the mom.layout file for you. The script can be customized by setting a few variables in it. To automatically create the mom.layout file, follow these instructions (these instructions are also included in the script):

1. Verify hwloc version 1.1 or higher is installed - see contrib/hwloc_install.sh.
2. Install Sys::Hwloc from CPAN.
3. Verify $PBS_HOME is set to the proper value.
4. Update the variables in the 'Config Definitions' section of the script. Especially update firstNodeId and nodesPerBoard if desired. The firstNodeId variable should be set above 0 if you have a root cpuset that you wish to exclude and the nodesPerBoard variable is the number of NUMA nodes per board. Each node is defined in /sys/devices/system/node, in a subdirectory node<node index>
5. Back up your current file in case a variable is set incorrectly or neglected.
6. Run the script.
7.
```
$ ./mom_gencfg
```

Manually Creating mom.layout

To properly set up the mom.layout file, it is important to know how the hardware is configured. Use the topology command line utility and inspect the contents of /sys/devices/system/node. The hwloc library can also be used to create a custom discovery tool.

Typing topology on the command line of a NUMA system produces something similar to the following:

```
Partition number: 0
6 Blades
72 CPUs
378.43 Gb Memory Total

Blade         ID       asic  NASID         Memory
-------------------------------------------------
    0 r001i01b00  UVHub 1.0      0    67089152 kB
    1 r001i01b01  UVHub 1.0      2    67092480 kB
    2 r001i01b02  UVHub 1.0      4    67092480 kB
    3 r001i01b03  UVHub 1.0      6    67092480 kB
    4 r001i01b04  UVHub 1.0      8    67092480 kB
    5 r001i01b05  UVHub 1.0     10    67092480 kB

CPU      Blade PhysID CoreID APIC-ID Family Model Speed L1(KiB) L2(KiB) L3(KiB)
------------------------------------------------------------------------------
  0 r001i01b00     00     00       0      6    46  2666 32d/32i     256   18432
  1 r001i01b00     00     02       4      6    46  2666 32d/32i     256   18432
  2 r001i01b00     00     03       6      6    46  2666 32d/32i     256   18432
  3 r001i01b00     00     08      16      6    46  2666 32d/32i     256   18432
  4 r001i01b00     00     09      18      6    46  2666 32d/32i     256   18432
  5 r001i01b00     00     11      22      6    46  2666 32d/32i     256   18432
  6 r001i01b00     01     00      32      6    46  2666 32d/32i     256   18432
  7 r001i01b00     01     02      36      6    46  2666 32d/32i     256   18432
  8 r001i01b00     01     03      38      6    46  2666 32d/32i     256   18432
  9 r001i01b00     01     08      48      6    46  2666 32d/32i     256   18432
 10 r001i01b00     01     09      50      6    46  2666 32d/32i     256   18432
 11 r001i01b00     01     11      54      6    46  2666 32d/32i     256   18432
 12 r001i01b01     02     00      64      6    46  2666 32d/32i     256   18432
 13 r001i01b01     02     02      68      6    46  2666 32d/32i     256   18432
 14 r001i01b01     02     03      70      6    46  2666 32d/32i     256   18432
```

From this partial output, note that this system has 72 CPUs on 6 blades. Each blade has 12 CPUs grouped into clusters of 6 CPUs. If the entire content of this command were printed you would see each Blade ID and the CPU ID assigned to each blade.

The topology command shows how the CPUs are distributed, but you likely also need to know where memory is located relative to CPUs, so go to `/sys/devices/system/node`. If you list the node directory you will see something similar to the following:

```
# ls -al
total 0
drwxr-xr-x 14 root root    0 Dec 3 12:14 .
drwxr-xr-x 14 root root    0 Dec 3 12:13 ..
-r--r--r--  1 root root 4096 Dec 3 14:58 has_cpu
-r--r--r--  1 root root 4096 Dec 3 14:58 has_normal_memory
drwxr-xr-x  2 root root 0 Dec 3 12:14 node0
drwxr-xr-x  2 root root 0 Dec 3 12:14 node1
drwxr-xr-x  2 root root 0 Dec 3 12:14 node10
drwxr-xr-x  2 root root 0 Dec 3 12:14 node11
drwxr-xr-x  2 root root 0 Dec 3 12:14 node2
drwxr-xr-x  2 root root 0 Dec 3 12:14 node3
drwxr-xr-x  2 root root 0 Dec 3 12:14 node4
drwxr-xr-x  2 root root 0 Dec 3 12:14 node5
drwxr-xr-x  2 root root 0 Dec 3 12:14 node6
drwxr-xr-x  2 root root 0 Dec 3 12:14 node7
drwxr-xr-x  2 root root 0 Dec 3 12:14 node8
drwxr-xr-x  2 root root 0 Dec 3 12:14 node9
-r--r--r--  1 root root 4096 Dec 3 14:58 online
-r--r--r--  1 root root 4096 Dec 3 14:58 possible
```

The directory entries node0, node1,...node11 represent groups of memory and CPUs local to each other. These groups are a node board, a grouping of resources that are close together. In most cases, a node board is made up of memory and processor cores. Each bank of memory is called a memory node by the operating system, and there are certain CPUs that can access that memory very rapidly. Note under the

directory for node board node0 that there is an entry called `cpulist`. This contains the CPU IDs of all CPUs local to the memory in node board 0.

Now create the `mom.layout` file. The content of cpulist 0-5 are local to the memory of node board 0, and the memory and cpus for that node are specified in the layout file by saying nodes=0. The cpulist for node board 1 shows 6-11 and memory node index 1. To specify this, simply write nodes=1. Repeat this for all twelve node boards and create the following `mom.layout` file for the 72 CPU system.

```
nodes=0
nodes=1
nodes=2
nodes=3
nodes=4
nodes=5
nodes=6
nodes=7
nodes=8
nodes=9
nodes=10
nodes=11
```

Each line in the mom.layout file is reported as a node to pbs_server by the pbs_mom daemon.

The `mom.layout` file does not need to match the hardware layout exactly. It is possible to combine node boards and create larger NUMA nodes. The following example shows how to do this:

```
nodes=0-1
```

The memory nodes can be combined the same as CPUs. The memory nodes combined must be contiguous. You cannot combine mem 0 and 2.

Configuring server_priv/nodes

The pbs_server requires awareness of how the MOM is reporting nodes since there is only one MOM daemon and multiple MOM nodes. So, configure the `server_priv/nodes` file with the **num_node_boards** and **numa_board_str** attributes. The attribute num_node_boards tells pbs_server how many numa nodes are reported by the MOM. Following is an example of how to configure the nodes file with num_node_boards:

```
numa-10 np=72 num_node_boards=12
```

This line in the nodes file tells pbs_server there is a host named numa-10 and that it has 72 processors and 12 nodes. The pbs_server divides the value of np (72) by the value for num_node_boards (12) and determines there are 6 CPUs per NUMA node.

In this example, the NUMA system is uniform in its configuration of CPUs per node board, but a system does not need to be configured with the same number of CPUs per node board. For systems with non-uniform CPU distributions, use the attribute **numa_board_str** to let pbs_server know where CPUs are located in the cluster.

The following is an example of how to configure the `server_priv/nodes` file for non-uniformly distributed CPUs:

```
Numa-11 numa_board_str=6,8,12
```

In this configuration, pbs_server knows it has three MOM nodes and the nodes have 6, 8, and 12 CPUs respectively. Note that the attribute np is not used. The np attribute is ignored because the number of CPUs per node is expressly given.

Enforcement of memory resource limits

TORQUE can better enforce memory limits with the use of the utility **memacctd**. The memacctd utility is provided by SGI on SuSe Linux Enterprise Edition (SLES). It is a daemon that caches memory footprints when it is queried. When configured to use the memory monitor, TORQUE queries memacctd. It is up to the user to make sure memacctd is installed. See the SGI memacctd man page for more information.

To configure TORQUE to use memacctd for memory enforcement

1. Start **memacctd** as instructed by SGI.
2. Reconfigure TORQUE with `--enable-numa-mem-monitor`. This will link in the necessary library when TORQUE is recompiled.
3. Recompile and reinstall TORQUE.
4. Restart all MOM nodes.
5. (Optional) Alter the **qsub** filter to include a default memory limit for all jobs that are not submitted with memory limit.

Related topics

- **TORQUE NUMA configuration** on page 28
- **TORQUE on NUMA systems** on page 28

1.8 TORQUE Multi-MOM

Starting in TORQUE version 3.0 users can run multiple MOMs on a single node. The initial reason to develop a multiple MOM capability was for testing purposes. A small cluster can be made to look larger since each MOM instance is treated as a separate node.

When running multiple MOMs on a node each MOM must have its own service and manager ports assigned. The default ports used by the MOM are 15002 and 15003. With the multi-mom alternate ports can be used without the need to change the default ports for pbs_server even when running a single instance of the MOM.

For details, see these topics:

- **Multi-MOM configuration** on page 32
- **Stopping pbs_mom in Multi-MOM mode** on page 34

1.8.1 Multi-MOM configuration

There are three steps to setting up multi-MOM capability:

1. **Configure server_priv/nodes** on page 33
2. **/ect/hosts file** on page 33
3. **Sarting pbs_mom with multi-MOM options** on page 33

Configure server_priv/nodes

The attributes **mom_service_port** and **mom_manager_port** were added to the nodes file syntax to accommodate multiple MOMs on a single node. By default pbs_mom opens ports 15002 and 15003 for the service and management ports respectively. For multiple MOMs to run on the same IP address they need to have their own port values so they can be distinguished from each other. pbs_server learns about the port addresses of the different MOMs from entries in the `server_priv/nodes` file. The following is an example of a nodes file configured for multiple MOMs:

```
hosta   np=2
hosta-1 np=2 mom_service_port=30001 mom_manager_port=30002
hosta-2 np=2 mom_service_port=31001 mom_manager_port=31002
hosta-3 np=2 mom_service_port=32001 mom_manager_port=32002
```

Note that all entries have a unique host name and that all port values are also unique. The entry `hosta` does not have a mom_service_port or mom_manager_port given. If unspecified, then the MOM defaults to ports 15002 and 15003.

/ect/hosts file

Host names in the `server_priv/nodes` file must be resolvable. Creating an alias for each host enables the server to find the IP address for each MOM; the server uses the port values from the `server_priv/nodes` file to contact the correct MOM. An example `/etc/hosts` entry for the previous `server_priv/nodes` example might look like the following:

```
192.65.73.10 hosta hosta-1 hosta-2 hosta-3
```

Even though the host name and all the aliases resolve to the same IP address, each MOM instance can still be distinguished from the others because of the unique port value assigned in the `server_priv/nodes` file.

Sarting pbs_mom with multi-MOM options

To start multiple instances of pbs_mom on the same node, use the following syntax (see **pbs_mom** on page 141 for details):

```
pbs_mom -m -M <port value of mom_service_port> -R <port value of mom_manager_port> -A
<name of MOM alias>
```

Continuing based on the earlier example, if you want to create four MOMs on `hosta`, type the following at the command line:

```
# pbs_mom -m -M 30001 -R 30002 -A hosta-1
# pbs_mom -m -M 31001 -R 31002 -A hosta-2
# pbs_mom -m -M 32001 -R 32002 -A hosta-3
# pbs_mom
```

Notice that the last call to pbs_mom uses no arguments. By default pbs_mom opens on ports 15002 and 15003. No arguments are necessary because there are no conflicts.

Related topics

- **TORQUE Multi-MOM** on page 32
- **Stopping pbs_mom in Multi-MOM mode** on page 34

1.8.2 Stopping pbs_mom in Multi-MOM mode

Terminate pbs_mom by using the `momctl -s` command (for details, see **momctl**). For any MOM using the default manager port 15003, the `momctl -s` command stops the MOM. However, to terminate MOMs with a manager port value not equal to 15003, you must use the following syntax:

```
momctl -s -p <port value of mom_manager_port>
```

The **-p** option sends the terminating signal to the MOM manager port and the MOM is terminated.

Related topics

- **TORQUE Multi-MOM** on page 32
- **Multi-MOM configuration** on page 32

Chapter 2: Submitting and managing jobs

This section contains information about how you can submit and manage jobs with TORQUE. For details, see the following topics:

- **Job submission** on page 35
- **Monitoring jobs** on page 46
- **Canceling jobs** on page 46
- **Job preemption** on page 46
- **Keeping completed jobs** on page 47
- **Job checkpoint and restart** on page 47
- **Job exit status** on page 57
- **Service jobs** on page 61

2.1 Job submission

Job submission is accomplished using the **qsub** command, which takes a number of command line arguments and integrates such into the specified PBS command file. The PBS command file may be specified as a filename on the qsub command line or may be entered via STDIN.

- The PBS command file does not need to be executable.
- The PBS command file may be *piped* into qsub (i.e., `cat pbs.cmd | qsub`).
- In the case of parallel jobs, the PBS command file is staged to, and executed on, the first allocated compute node only. (Use **pbsdsh** to run actions on multiple nodes.)
- The command script is executed from the user's home directory in all cases. (The script may determine the submission directory by using the $PBS_O_WORKDIR environment variable)
- The command script will be executed using the default set of user environment variables unless the **-V** or **-v** flags are specified to include aspects of the job submission environment.

By default, job submission is allowed only on the TORQUE server host (host on which **pbs_server** is running). Enablement of job submission from other hosts is documented in **Server configuration** on page 19.

Versions of TORQUE earlier than 2.4.5 attempted to apply queue and server defaults to a job that didn't have defaults specified. If a setting still did not have a value after that, TORQUE applied the queue and server maximum values to a job (meaning, the maximum values for an applicable setting were applied to jobs that had no specified or default value).

In TORQUE 2.4.5 and later, the queue and server maximum values are no longer used as a value for missing settings.

This section contains these topics:

- **Multiple job submission** on page 36
- **Requesting resources** on page 38
- **Requesting generic resources** on page 42
- **Requesting floating resources** on page 43
- **Requesting other resources** on page 43
- **Exported batch environment variables** on page 43
- **Enabling trusted submit hosts** on page 45
- **Example submit scripts** on page 45

Related topics

- Maui Documentation
- http://www.lunarc.lu.se
- http://www.clusters.umaine.edu/wiki/index.php/Example_Submission_Scripts
- **Job submission filter ("qsub wrapper")** on page 269 – Allow local checking and modification of submitted job

2.1.1 Multiple job submission

Sometimes users will want to submit large numbers of jobs based on the same job script. Rather than using a script to repeatedly call qsub, a feature known as job arrays now exists to allow the creation of multiple jobs with one qsub command. Additionally, this feature includes a new job naming convention that allows users to reference the entire set of jobs as a unit, or to reference one particular job from the set.

Job arrays are submitted through the **-t** option to qsub, or by using `#PBS -t` in your batch script. This option takes a comma-separated list consisting of either a single job ID number, or a pair of numbers separated by a dash. Each of these jobs created will use the same script and will be running in a nearly identical environment.

```
> qstat -t 0-4 job_script
1098[].hostname

> qstat
1098[0].hostname ...
1098[1].hostname ...
1098[2].hostname ...
1098[3].hostname ...
1098[4].hostname ...
```

Versions of TORQUE earlier than 2.3 had different semantics for the **-t** argument. In these versions, -t took a single integer number—a count of the number of jobs to be created.

Each 1098[x] job has an environment variable called PBS_ARRAYID, which is set to the value of the array index of the job, so 1098[0].hostname would have PBS_ARRAYID set to 0. This allows you to create job arrays where each job in the array performs slightly different actions based on the value of this variable, such as performing the same tasks on different input files. One other difference in the environment between jobs in the same array is the value of the PBS_JOBNAME variable.

```
# These two examples are equivalent in TORQUE 2.2
> qsub -t 0-99
> qsub -t 100

# You can also pass comma delimited lists of ids and ranges:
> qsub -t 0,10,20,30,40
> qsub -t 0-50,60,70,80
```

Running qstat displays a job summary, which provides an overview of the array's state. To see each job in the array, run qstat **-t**.

The **qalter**, **qdel**, **qhold**, and **qrls** commands can operate on arrays—either the entire array or a range of that array. Additionally, any job in the array may be accessed normally by using that job's ID, just as you would with any other job. For example, running the following command would run only the specified job:

```
qrun 1098[0].hostname
```

Slot Limit

The slot limit is a way for administrators to limit the number of jobs from a job array that can be eligible for scheduling at the same time. When a slot limit is used, TORQUE puts a hold on all jobs in the array that exceed the slot limit. When an eligible job in the array completes, TORQUE removes the hold flag from the next job in the array. Slot limits can be declared globally with the max_slot_limit parameter, or on a per-job basis with qsub -t.

Related topics

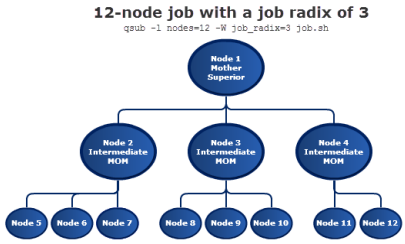
- **Job submission** on page 35

2.1.2 Managing multi-node jobs

By default, when a multi-node job runs, the Mother Superior manages the job across all the sister nodes by communicating with each of them and updating pbs_server. Each of the sister nodes sends its updates and stdout and stderr directly to the Mother Superior. When you run an extremely large job using hundreds or thousands of nodes, you may want to reduce the amount of network traffic sent from the sisters to the Mother Superior by specifying a job radix. Job radix sets a maximum number of nodes with which the Mother Superior and resulting intermediate MOMs communicate and is specified using the **-W** on page 199 option for qsub.

For example, if you submit a smaller, 12-node job and specify job_radix=3, Mother Superior and each resulting intermediate MOM is only allowed to receive communication from 3 subordinate nodes.

Image 2-1: Job radix example



The Mother Superior picks three sister nodes with which to communicate the job information. Each of those nodes (intermediate MOMs) receives a list of all sister nodes that will be subordinate to it. They each contact up to three nodes and pass the job information on to those nodes. This pattern continues until the bottom level is reached. All communication is now passed across this new hierarchy. The stdout and stderr data is aggregated and sent up the tree until it reaches the Mother Superior, where it is saved and copied to the `.o` and `.e` files.

Job radix is meant for extremely large jobs only. It is a tunable parameter and should be adjusted according to local conditions in order to produce the best results.

2.1.3 Requesting resources

Various resources can be requested at the time of job submission. A job can request a particular node, a particular node attribute, or even a number of nodes with particular attributes. Either native TORQUE resources, or external scheduler resource extensions may be specified. The native TORQUE resources are listed in the following table:

Resource	Format	Description
arch	string	Specifies the administrator defined system architecture required. This defaults to whatever the **PBS_MACH** string is set to in "local.mk".
cput	seconds, or [[HH:]MM;]SS	Maximum amount of CPU time used by all processes in the job.
epilogue	string	Specifies a user owned epilogue script which will be run before the system epilogue and epilogue.user scripts at the completion of a job. The syntax is `epilogue=<file>`. The file can be designated with an absolute or relative path. For more information, see **Prologue and epilogue scripts** on page 257.
file	**size**	The amount of total disk requested for the job. (Ignored on Unicos.)

Resource	Format	Description
host	string	Name of the host on which the job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent.
mem	size	Maximum amount of physical memory used by the job. (Ignored on Darwin, Digital Unix, Free BSD, HPUX 11, IRIX, NetBSD, and SunOS. Also ignored on Linux if number of nodes is not 1. Not implemented on AIX and HPUX 10.)
nice	integer	Number between -20 (highest priority) and 19 (lowest priority). Adjust the process execution priority.
nodes	{<node_count> | <hostname>} [:ppn=<ppn>] [:gpus=<gpu>] [:<property> [:<property>]...] [+ ...]	Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more node_specs joined with the + (plus) character: node_spec [+node_spec...]. Each node_spec is a number of nodes required of the type declared in the node_spec and a name of one or more properties desired for the nodes. The number, the name, and each property in the node_spec are separated by a : (colon). If no number is specified, one (1) is assumed. The name of a node is its hostname. The properties of nodes are: • **ppn=#** - Specify the number of virtual processors per node requested for this job. The number of virtual processors available on a node by default is 1, but it can be configured in the $TORQUE_HOME/server_priv/nodes file using the np attribute (see **Server node file configuration** on page 24). The virtual processor can relate to a physical core on the node or it can be interpreted as an "execution slot" such as on sites that set the node np value greater than the number of physical cores (or hyper-thread contexts). The ppn value is a characteristic of the hardware, system, and site, and its value is to be determined by the administrator. • **gpus=#** - Specify the number of GPUs per node requested for this job. The number of GPUs available on a node can be configured in the $TORQUE_HOME/server_priv/nodes file using the gpu attribute (see **Server node file configuration** on page 24). The GPU value is a characteristic of the hardware, system, and site, and its value is to be determined by the administrator. • **property** - A string assigned by the system administrator specifying a node's features. Check with your administrator as to the node names and properties available to you. See **qsub -l nodes** on page 41 for examples. By default, the node resource is mapped to a virtual node (that is, directly to a processor, not a full physical compute node). This behavior can be changed within Maui or Moab by setting the JOBNODEMATCHPOLICY parameter. See "Appendix F: Parameters" of the Moab Workload Manager Administrator Guide for more information.

Resource	Format	Description
opsys	string	Specifies the administrator defined operating system as defined in the MOM configuration file.
other	string	Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. This does not work for msub using Moab and Maui.
pcput	seconds, or [[HH:]MM:]SS	Maximum amount of CPU time used by any single process in the job.
pmem	**size**	Maximum amount of physical memory used by any single process of the job. (Ignored on Fujitsu. Not implemented on Digital Unix and HPUX.)
procs	procs=<integer>	*(Applicable in version 2.5.0 and later.)* The number of processors to be allocated to a job. The processors can come from one or more qualified node(s). Only one procs declaration may be used per submitted **qsub** command. `> qsub -l nodes=3 -l procs=2`
procs_bitmap	string	A string made up of 1's and 0's in reverse order of the processor cores requested. A **procs_bitmap=1110** means the job requests a node that has four available cores, but the job runs exclusively on cores two, three, and four. With this bitmap, core one is not used. For more information, see **Scheduling cores** on page 68.
prologue	string	Specifies a user owned prologue script which will be run after the system prologue and prologue.user scripts at the beginning of a job. The syntax is `prologue=<file>`. The file can be designated with an absolute or relative path. For more information, see **Prologue and epilogue scripts** on page 257.
pvmem	**size**	Maximum amount of virtual memory used by any single process in the job. (Ignored on Unicos.)
size	integer	For TORQUE, this resource has no meaning. It is passed on to the scheduler for interpretation. In the Moab scheduler, the size resource is intended for use in Cray installations only.

Resource	Format	Description
software	string	Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. (See "Scheduler License Manager" in the Moab Workload Manager Administrator Guide for more information.)
vmem	**size**	Maximum amount of virtual memory used by all concurrent processes in the job. (Ignored on Unicos.)
walltime	seconds, or [[HH:]MM:]SS	Maximum amount of real time during which the job can be in the running state.

size

The size format specifies the maximum amount in terms of bytes or words. It is expressed in the form *integer[suffix]*. The suffix is a multiplier defined in the following table ("b" means bytes [the default] and "w" means words). The size of a word is calculated on the execution server as its word size.

Suffix		Multiplier
b	w	1
kb	kw	1024
mb	mw	1,048,576
gb	gw	1,073,741,824
tb	tw	1,099,511,627,776

Example 2-1: qsub -l nodes

Usage	Description
`> qsub -l nodes=12`	Request 12 nodes of any type
`> qsub -l nodes=2:server+14`	Request 2 "server" nodes and 14 other nodes (a total of 16) - this specifies two node_specs, "2:server" and "14"

Usage	Description
`> qsub -l nodes=server:hippi-+10:noserver+3:bigmem:hippi`	Request (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not servers, and (c) 3 nodes that have a large amount of memory and have hippi
`> qsub -l nodes=b2005+b1803+b1813`	Request 3 specific nodes by hostname
`> qsub -l nodes=4:ppn=2`	Request 2 processors on each of four nodes
`> qsub -l nodes=1:ppn=4`	Request 4 processors on one node
`> qsub -l nodes=2:blue:ppn=2+red:ppn=3+b1014`	Request 2 processors on each of two blue nodes, three processors on one red node, and the compute node "b1014"

Example 2-2:

This job requests a node with 200MB of available memory:

```
> qsub -l mem=200mb /home/user/script.sh
```

Example 2-3:

This job will wait until node01 is free with 200MB of available memory:

```
> qsub -l nodes=node01,mem=200mb /home/user/script.sh
```

Related topics

- **Job submission** on page 35

2.1.4 Requesting generic resources

When **generic** resources have been assigned to nodes using the server's nodes file, these resources can be requested at the time of job submission using the *other* field. (See "Consumable Generic Resources" in the Moab Workload Manager Administrator Guide for details on configuration within Moab).

Example 2-1: Generic

This job will run on any node that has the generic resource **matlab**.

```
> qsub -l other=matlab /home/user/script.sh
```

This can also be requested at the time of job submission using the `-W x=GRES:matlab` flag.

Related topics

- **Requesting resources** on page 38
- **Job submission** on page 35

2.1.5 Requesting floating resources

When **floating** resources have been set up inside Moab, they can be requested in the same way as **generic** resources. Moab will automatically understand that these resources are floating and will schedule the job accordingly. (See "Floating Generic Resources" in the Moab Workload Manager Administrator Guide for details on configuration within Moab.)

Example 2-1: Floating

This job will run on any node when there are enough floating resources available.

```
> qsub -l other=matlab /home/user/script.sh
```

This can also be requested at the time of job submission using the `-W x=GRES:matlab` flag.

Related topics

- **Requesting resources** on page 38
- **Job submission** on page 35

2.1.6 Requesting other resources

Many other resources can be requested at the time of job submission using the Moab Workload Manager. See "Resource Manager Extensions" in the Moab Workload Manager Administrator Guide for a list of these supported requests and correct syntax.

Related topics

- **Requesting resources** on page 38
- **Job submission** on page 35

2.1.7 Exported batch environment variables

When a batch job is started, a number of variables are introduced into the job's environment that can be used by the batch script in making decisions, creating output files, and so forth. These variables are listed in the following table:

Variable	Description
PBS_JOBNAME	User specified jobname

Variable	Description
PBS_ARRAYID	Zero-based value of job array index for this job (in version 2.2.0 and later)
PBS_O_WORKDIR	Job's submission directory
PBS_ENVIRONMENT	N/A
PBS_TASKNUM	Number of tasks requested
PBS_O_HOME	Home directory of submitting user
PBS_MOMPORT	Active port for MOM daemon
PBS_O_LOGNAME	Name of submitting user
PBS_O_LANG	Language variable for job
PBS_JOBCOOKIE	Job cookie
PBS_NODENUM	Node offset number
PBS_O_SHELL	Script shell
PBS_O_JOBID	Unique pbs job id
PBS_O_HOST	Host on which job script is currently running
PBS_QUEUE	Job queue
PBS_NODEFILE	File containing line delimited list on nodes allocated to the job
PBS_O_PATH	Path variable used to locate executables within job script

Related topics

- **Requesting resources** on page 38
- **Job submission** on page 35

2.1.8 Enabling trusted submit hosts

By default, only the node running the pbs_server daemon is allowed to submit jobs. Additional nodes can be trusted as submit hosts by taking any of the following steps:

- Set the **allow_node_submit** server parameter (see **Allowing job submission from compute hosts** on page 20).

  Allows any host trusted as a compute host to also be trusted as a submit host.

- Set the **submit_hosts** server parameter (comma-delimited) (see **Using the "submit_hosts" service parameter** on page 20).

  Allows specified hosts to be trusted as a submit host.

- Use **.rhosts** to enable `ruserok()` based authentication (see **Using RCmd authentication** on page 19).

See **Configuring job submission hosts** on page 19 for more information.

If **allow_node_submit** is set, the **allow_proxy_user**parameter must also be set to allow user proxying when submitting/running jobs.

Related topics

- **Job submission** on page 35

2.1.9 Example submit scripts

The following is an example job test script:

```
#!/bin/sh
#
#This is an example script example.sh
#
#These commands set up the Grid Environment for your job:
#PBS -N ExampleJob
#PBS -l nodes=1,walltime=00:01:00
#PBS -q np_workq
#PBS -M YOURUNIQNAME@umich.edu
#PBS -m abe

#print the time and date
date

#wait 10 seconds
sleep 10

#print the time and date again
date
```

Related topics

- **Job submission** on page 35

2.2 Monitoring jobs

TORQUE allows users and administrators to monitor submitted jobs with the **qstat** command. If the command is run by a non-administrative user, it will output just that user's jobs. For example:

```
> qstat
Job id           Name             User             Time Use S Queue
---------------- ---------------- ---------------- -------- - -----
4807             scatter          user01           12:56:34 R batch
...
```

Related topics

- **Submitting and managing jobs** on page 35

2.3 Canceling jobs

TORQUE allows users and administrators to cancel submitted jobs with the **qdel** command. The job will be sent TERM and KILL signals killing the running processes. When the top-level job script exits, the job will exit. The only parameter is the ID of the job to be canceled.

If a job is canceled by an operator or manager, an email notification will be sent to the user. Operators and managers may add a comment to this email with the `-m` option.

```
$ qstat
Job id           Name             User             Time Use S Queue
---------------- ---------------- ---------------- -------- - -----
4807             scatter          user01           12:56:34 R batch
...
$ qdel -m "hey! Stop abusing the NFS servers" 4807
$
```

Related topics

- **Submitting and managing jobs** on page 35

2.4 Job preemption

TORQUE supports job preemption by allowing authorized users to suspend and resume jobs. This is supported using one of two methods. If the node supports OS-level preemption, TORQUE will recognize that during the configure process and enable it. Otherwise, the MOM may be configured to launch a custom *checkpoint script* in order to support preempting a job. Using a custom checkpoint script requires that the job understand how to resume itself from a checkpoint after the preemption occurs.

Configuring a checkpoint script on a MOM

To configure the MOM to support a checkpoint script, the `$checkpoint_script` parameter must be set in the MOM's configuration file found in `TORQUE_HOME/mom_priv/config`. The checkpoint script should have execute permissions set. A typical configuration file might look as follows:

mom_priv/config:

```
$pbsserver          node06
$logevent           255
$restricted         *.mycluster.org
$checkpoint_script  /opt/moab/tools/mom-checkpoint.sh
```

The second thing that must be done to enable the checkpoint script is to change the value of MOM_CHECKPOINT to **1** in /src/include/pbs_config.h. (In some instances, MOM_CHECKPOINT may already be defined as 1.) The new line should be as follows:

/src/include/pbs_config.h:

```
#define MOM_CHECKPOINT 1
```

Related topics

- **Submitting and managing jobs** on page 35

2.5 Keeping completed jobs

TORQUE provides the ability to report on the status of completed jobs for a configurable duration after the job has completed. This can be enabled by setting the **keep_completed** attribute on the job execution queue. This should be set to the number of seconds that jobs should be held in the queue. Completed jobs will be reported in the **C** state and the exit status is seen in the exit_status job attribute.

By maintaining status information about completed (or canceled, failed, etc.) jobs, administrators can better track failures and improve system performance. This allows TORQUE to better communicate with Moab Workload Manager and track the status of jobs. This gives Moab the ability to track specific failures and to schedule the workload around possible hazards. (See NODEFAILURERESERVETIME in "Appendix F: Parameters" of the Moab Workload Manager Administrator Guide for more information.)

Related topics

- **Submitting and managing jobs** on page 35

2.6 Job checkpoint and restart

While TORQUE has had a job checkpoint and restart capability for many years, this was tied to machine specific features. Now TORQUE supports BLCR—an architecture independent package that provides for process checkpoint and restart.

The support for BLCR is only for serial jobs, not for any MPI type jobs.

This section contains these topics:

- **Introduction to BLCR** on page 48
- **Configuration files and scripts** on page 48

- **Starting a checkpointable job** on page 55
- **Checkpointing a job** on page 56
- **Restarting a job** on page 56
- **Acceptance tests** on page 57

Related topics

- **Submitting and managing jobs** on page 35

2.6.1 Introduction to BLCR

BLCR is a kernel level package. It must be downloaded and installed from BLCR.

After building and making the package, it must be installed into the kernel with commands as follows. These can be installed into the file `/etc/modules` but all of the testing was done with explicit invocations of **modprobe**.

Installing BLCR into the kernel:

```
# /sbin/insmod /usr/local/lib/blcr/2.6.12-1.234/blcr_imports.ko
# /sbin/insmod /usr/local/lib/blcr/2.6.12-1.234/blcr_vmadump.ko
# /sbin/insmod /usr/local/lib/blcr/2.6.12-1.234/blcr.ko
```

The BLCR system provides four command line utilities:

- cr_checkpoint
- cr_info
- cr_restart
- cr_run

For more information about BLCR, see the BLCR Administrator's Guide.

Related topics

- **Job checkpoint and restart** on page 47

2.6.2 Configuration files and scripts

Configuring and Building TORQUE for BLCR:

```
> ./configure --enable-unixsockets=no --enable-blcr
> make
> sudo make install
```

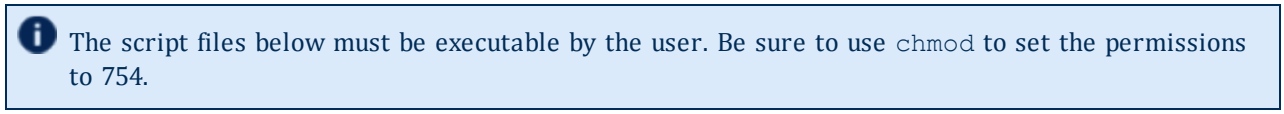
Depending on where BLCR is installed you may also need to use the following configure options to specify BLCR paths:

Option	Description
--with-blcr-include=*DIR*	include path for libcr.h
--with-blcr-lib=*DIR*	lib path for libcr
--with-blcr-bin=*DIR*	bin path for BLCR utilities

The pbs_mom configuration file located in `/var/spool/torque/mom_priv` must be modified to identify the script names associated with invoking the BLCR commands. The following variables should be used in the configuration file when using BLCR checkpointing.

Variable	Description
$checkpoint_interval	How often periodic job checkpoints will be taken (minutes)
$checkpoint_script	The name of the script file to execute to perform a job checkpoint
$restart_script	The name of the script file to execute to perform a job restart
$checkpoint_run_exe	The name of an executable program to be run when starting a checkpointable job (for BLCR, cr_run)

The following example shows the contents of the configuration file used for testing the BLCR feature in TORQUE.


The script files below must be executable by the user. Be sure to use `chmod` to set the permissions to 754.

Example 2-1: Script file permissions

```
# chmod 754 blcr*
# ls -l
total 20
-rwxr-xr-- 1 root root 2112 2008-03-11 13:14 blcr_checkpoint_script
-rwxr-xr-- 1 root root 1987 2008-03-11 13:14 blcr_restart_script
-rw-r--r-- 1 root root 215 2008-03-11 13:13 config
drwxr-x--x 2 root root 4096 2008-03-11 13:21 jobs
-rw-r--r-- 1 root root 7 2008-03-11 13:15 mom.lock
```

Example 2-2: mom_priv/config

```
$checkpoint_script /var/spool/torque/mom_priv/blcr_checkpoint_script
$restart_script /var/spool/torque/mom_priv/blcr_restart_script
$checkpoint_run_exe /usr/local/bin/cr_run
$pbsserver makua.cridomain
$loglevel 7
```

Example 2-3: `mom_priv/blcr_checkpoint_script`

```
#! /usr/bin/perl
###############################################################################
#
# Usage: checkpoint_script
#
# This script is invoked by pbs_mom to checkpoint a job.
#
###############################################################################
use strict;
use Sys::Syslog;

# Log levels:
# 0 = none -- no logging
# 1 = fail -- log only failures
# 2 = info -- log invocations
# 3 = debug -- log all subcommands
my $logLevel = 3;

logPrint(2, "Invoked: $0 " . join(' ', @ARGV) . "\n");

my ($sessionId, $jobId, $userId, $signalNum, $checkpointDir, $checkpointName);
my $usage =
  "Usage: $0        \n";

# Note that depth is not used in this script but could control a limit to the number of
checkpoint
# image files that are preserved on the disk.
#
# Note also that a request was made to identify whether this script was invoked by the
job's
# owner or by a system administrator. While this information is known to pbs_server, it
# is not propagated to pbs_mom and thus it is not possible to pass this to the script.
# Therefore, a workaround is to invoke qmgr and attempt to set a trivial variable.
# This will fail if the invoker is not a manager.

if (@ARGV == 7)
{
    ($sessionId, $jobId, $userId, $checkpointDir, $checkpointName, $signalNum $depth) =
       @ARGV;
}
else { logDie(1, $usage); }

# Change to the checkpoint directory where we want the checkpoint to be created
chdir $checkpointDir
  or logDie(1, "Unable to cd to checkpoint dir ($checkpointDir): $!\n")
  if $logLevel;

my $cmd = "cr_checkpoint";
$cmd .= " --signal $signalNum" if $signalNum;
$cmd .= " --tree $sessionId";
$cmd .= " --file $checkpointName";
my $output = `$cmd 2>&1`;
my $rc = $? >> 8;
logDie(1, "Subcommand ($cmd) failed with rc=$rc:\n$output")
  if $rc && $logLevel >= 1;
logPrint(3, "Subcommand ($cmd) yielded rc=$rc:\n$output")
  if $logLevel >= 3;
exit 0;

###############################################################################
# logPrint($message)
# Write a message (to syslog) and die
###############################################################################
sub logPrint
{
    my ($level, $message) = @_;
    my @severity = ('none', 'warning', 'info', 'debug');
```

```
    return if $level > $logLevel;

    openlog('checkpoint_script', '', 'user');
    syslog($severity[$level], $message);
    closelog();
}

##############################################################################
# logDie($message)
# Write a message (to syslog) and die
##############################################################################
sub logDie
{
    my ($level, $message) = @_;
    logPrint($level, $message);
    die($message);
}
```

Example 2-4: `mom_priv/blcr_restart_script`

```
#! /usr/bin/perl
###############################################################################
#
# Usage: restart_script
#
# This script is invoked by pbs_mom to restart a job.
#
###############################################################################
use strict;
use Sys::Syslog;

# Log levels:
# 0 = none -- no logging
# 1 = fail -- log only failures
# 2 = info -- log invocations
# 3 = debug -- log all subcommands
my $logLevel = 3;

logPrint(2, "Invoked: $0 " . join(' ', @ARGV) . "\n");

my ($sessionId, $jobId, $userId, $checkpointDir, $restartName);
my $usage =
  "Usage: $0 \n";
if (@ARGV == 5)
{
    ($sessionId, $jobId, $userId, $checkpointDir, $restartName) =
       @ARGV;
}
else { logDie(1, $usage); }

# Change to the checkpoint directory where we want the checkpoint to be created
chdir $checkpointDir
  or logDie(1, "Unable to cd to checkpoint dir ($checkpointDir): $!\n")
  if $logLevel;

my $cmd = "cr_restart";
$cmd .= " $restartName";
my $output = `$cmd 2>&1`;
my $rc = $? >> 8;
logDie(1, "Subcommand ($cmd) failed with rc=$rc:\n$output")
  if $rc && $logLevel >= 1;
logPrint(3, "Subcommand ($cmd) yielded rc=$rc:\n$output")
  if $logLevel >= 3;
exit 0;

###############################################################################
# logPrint($message)
# Write a message (to syslog) and die
###############################################################################
sub logPrint
{
    my ($level, $message) = @_;
    my @severity = ('none', 'warning', 'info', 'debug');

    return if $level > $logLevel;
    openlog('restart_script', '', 'user');
    syslog($severity[$level], $message);
    closelog();
}

###############################################################################
# logDie($message)
# Write a message (to syslog) and die
###############################################################################
sub logDie
{
    my ($level, $message) = @_;
```

```
    logPrint($level, $message);
    die($message);
}
```

Related topics

- **Job checkpoint and restart** on page 47

2.6.3 Starting a checkpointable job

Not every job is checkpointable. A job for which checkpointing is desirable must be started with the `-c` command line option. This option takes a comma-separated list of arguments that are used to control checkpointing behavior. The list of valid options available in the 2.4 version of TORQUE is show below.

Option	Description
none	No checkpointing (not highly useful, but included for completeness).
enabled	Specify that checkpointing is allowed, but must be explicitly invoked by either the `qhold` or `qchkpt` commands.
shutdown	Specify that checkpointing is to be done on a job at pbs_mom shutdown.
periodic	Specify that periodic checkpointing is enabled. The default interval is 10 minutes and can be changed by the `$checkpoint_interval` option in the MOM configuration file, or by specifying an interval when the job is submitted.
interval=minutes	Specify the checkpoint interval in minutes.
depth=number	Specify a number (depth) of checkpoint images to be kept in the checkpoint directory.
dir=path	Specify a checkpoint directory (default is `/var/spool/torque/checkpoint`).

Example 2-1: Sample test program

```
#include "stdio.h"
int main( int argc, char *argv[] )
{
int i;
          for (i=0; i<100; i++)
          {
                  printf("i = %d\n", i);
                  fflush(stdout);
                  sleep(1);
          }
}
```

Example 2-2: Instructions for building test program

```
> gcc -o test test.c
```

Example 2-3: Sample test script

```
#!/bin/bash ./test
```

Example 2-4: Starting the test job

```
> qstat
> qsub -c enabled,periodic,shutdown,interval=1 test.sh
77.jakaa.cridomain
> qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
77.jakaa                  test.sh          jsmith                 0 Q batch
>
```

If you have no scheduler running, you might need to start the job with **qrun**.

As this program runs, it writes its output to a file in `/var/spool/torque/spool`. This file can be observed with the command `tail -f`.

Related topics

- **Job checkpoint and restart** on page 47

2.6.4 Checkpointing a job

Jobs are checkpointed by issuing a **qhold** command. This causes an image file representing the state of the process to be written to disk. The directory by default is `/var/spool/torque/checkpoint`.

This default can be altered at the queue level with the `qmgr` command. For example, the command `qmgr -c set queue batch checkpoint_dir=/tmp` would change the checkpoint directory to `/tmp` for the queue 'batch'.

The default directory can also be altered at job submission time with the `-c dir=/tmp` command line option.

The name of the checkpoint directory and the name of the checkpoint image file become attributes of the job and can be observed with the command `qstat -f`. Notice in the output the names **checkpoint_dir** and **checkpoint_name**. The variable checkpoint_name is set when the image file is created and will not exist if no checkpoint has been taken.

A job can also be checkpointed without stopping or holding the job with the command **qchkpt**.

Related topics

- **Job checkpoint and restart** on page 47

2.6.5 Restarting a job

Restarting a job in the Held state

The **qrls** command is used to restart the hibernated job. If you were using the `tail -f` command to watch the output file, you will see the test program start counting again.

It is possible to use the **qalter** command to change the name of the checkpoint file associated with a job. This could be useful if there were several job checkpoints and it restarting the job from an older image was specified.

Restarting a job in the Completed state

In this case, the job must be moved to the Queued state with the **qrerun** command. Then the job must go to the Run state either by action of the scheduler or if there is no scheduler, through using the **qrun** command.

Related topics

- **Job checkpoint and restart** on page 47

2.6.6 Acceptance tests

A number of tests were made to verify the functioning of the BLCR implementation. See **BLCR acceptance tests** on page 279 for a description of the testing.

Related topics

- **Job checkpoint and restart** on page 47

2.7 Job exit status

Once a job under TORQUE has completed, the `exit_status` attribute will contain the result code returned by the job script. This attribute can be seen by submitting a `qstat -f` command to show the entire set of information associated with a job. The `exit_status` field is found near the bottom of the set of output lines.

Example 2-1: qstat -f (job failure)

```
Job Id: 179.host
    Job_Name = STDIN
    Job_Owner = user@host
    job_state = C
    queue = batchq server = host
    Checkpoint = u ctime = Fri Aug 29 14:55:55 2008
    Error_Path = host:/opt/moab/STDIN.e179
    exec_host = node1/0
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Fri Aug 29 14:55:55 2008
    Output_Path = host:/opt/moab/STDIN.o179
    Priority = 0
    qtime = Fri Aug 29 14:55:55 2008
    Rerunable = True Resource_List.ncpus = 2
    Resource_List.nodect = 1
    Resource_List.nodes = node1
    Variable_List = PBS_O_HOME=/home/user,PBS_O_LOGNAME=user,
 PBS_O_PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:,PBS_O_
SHELL=/bin/bash,PBS_O_HOST=host,
 PBS_O_WORKDIR=/opt/moab,PBS_O_QUEUE=batchq
    sched_hint = Post job file processing error; job 179.host on host node1/0Ba
 d UID for job execution REJHOST=pala.cridomain MSG=cannot find user 'user' in password
file
    etime = Fri Aug 29 14:55:55 2008
    exit_status = -1
```

This code can be useful in diagnosing problems with jobs that may have unexpectedly terminated.

If TORQUE was unable to start the job, this field will contain a negative number produced by the pbs_mom. Otherwise, if the job script was successfully started, the value in this field will be the return value of the script.

Example 2-2: TORQUE supplied exit codes

Name	Value	Description
JOB_EXEC_OK	0	Job execution successful
JOB_EXEC_FAIL1	-1	Job execution failed, before files, no retry
JOB_EXEC_FAIL2	-2	Job execution failed, after files, no retry
JOB_EXEC_RETRY	-3	Job execution failed, do retry
JOB_EXEC_INITABT	-4	Job aborted on MOM initialization
JOB_EXEC_INITRST	-5	Job aborted on MOM init, chkpt, no migrate
JOB_EXEC_INITRMG	-6	Job aborted on MOM init, chkpt, ok migrate

Name	Value	Description
JOB_EXEC_BADRESRT	-7	Job restart failed
JOB_EXEC_CMDFAIL	-8	Exec() of user command failed
JOB_EXEC_STDOUTFAIL	-9	
JOB_EXEC_OVERLIMIT	-10	

Example 2-3: Exit code from C program

```
$ cat error.c

#include
#include


int
main(int argc, char *argv)
{
   exit(256+11);
}


$ gcc -o error error.c


$ echo ./error | qsub
180.xxx.yyy


$ qstat -f
Job Id: 180.xxx.yyy
    Job_Name = STDIN
    Job_Owner = test.xxx.yyy
    resources_used.cput = 00:00:00
    resources_used.mem = 0kb
    resources_used.vmem = 0kb
    resources_used.walltime = 00:00:00
    job_state = C
    queue = batch
    server = xxx.yyy
    Checkpoint = u
    ctime = Wed Apr 30 11:29:37 2008
    Error_Path = xxx.yyy:/home/test/STDIN.e180
    exec_host = node01/0
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Wed Apr 30 11:29:37 2008
    Output_Path = xxx.yyy:/home/test/STDIN.o180
    Priority = 0
    qtime = Wed Apr 30 11:29:37 2008
    Rerunable = True
    Resource_List.neednodes = 1
    Resource_List.nodect = 1
    Resource_List.nodes = 1
    Resource_List.walltime = 01:00:00
    session_id = 14107
    substate = 59
    Variable_List = PBS_O_HOME=/home/test,PBS_O_LANG=en_US.UTF-8,
        PBS_O_LOGNAME=test,
        PBS_O_PATH=/usr/local/perltests/bin:/home/test/bin:/usr/local/s
        bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games,
        PBS_O_SHELL=/bin/bash,PBS_SERVER=xxx.yyy,
        PBS_O_HOST=xxx.yyy,PBS_O_WORKDIR=/home/test,
        PBS_O_QUEUE=batch
    euser = test
    egroup = test
    hashname = 180.xxx.yyy
    queue_rank = 8
    queue_type = E
    comment = Job started on Wed Apr 30 at 11:29
    etime = Wed Apr 30 11:29:37 2008
    exit_status = 11
    start_time = Wed Apr 30 11:29:37 2008
    start_count = 1
```

Notice that the C routine **exit** passes only the low order byte of its argument. In this case, 256+11 is really 267 but the resulting exit code is only 11 as seen in the output.

Related topics

- **Job checkpoint and restart** on page 47
- **Submitting and managing jobs** on page 35

2.8 Service jobs

TORQUE service jobs are a special kind of job that is treated differently by TORQUE than normal batch jobs. TORQUE service jobs are **not** related to Moab's dynamic service jobs. A TORQUE service job cannot dynamically grow and shrink in size over time.

Jobs are marked as service jobs at the time they are submitted to Moab or TORQUE. Just like a normal job, a script file is specified with the job. In a batch job, the contents of the script file are taken by TORQUE and executed on the compute nodes. For a service job, however, the script file is assumed to respond to certain command-line arguments. Instead of just executing the script, TORQUE will use these command-line arguments to start, stop, and check on the status of the job. Listed below are the three command-line arguments that must be supported by any script submitted as part of a TORQUE service job:

- `start`: The script should take this argument and launch its service/workload. The script should remain executing/running until the service stops.
- `stop`: The script should take this argument and stop the service/workload that was earlier started.
- `status`: The script should take this argument and return, via standard out, either "running" if the service/workload is running as expected or "stopped" if the service is not running.

This feature was created with long-running services in mind. The command-line arguments should be familiar to users who interact with Unix services, as each of the service scripts found in `/etc/init.d/` also accept and respond to the arguments as explained above.

For example, if a user wants to start the Apache 2 server on a compute node, they can use a TORQUE service job and specify a script which will start, stop, and check on the status of the "httpd" daemon--possibly by using the already present `/etc/init.d/httpd` script.

If you wish to submit service jobs only through TORQUE, no special version of Moab is required. If you wish to submit service jobs using Moab's msub, then Moab 5.4 is required.

For details, see these topics:

- **Submitting service jobs** on page 62
- **Submitting service jobs in MCM** on page 62
- **Managing service jobs** on page 62

2.8.1 Submitting service jobs

There is a new option to **qsub**, "**-s**" which can take either a 'y' or 'n' (yes or no, respectively). When "-s y" is present, then the job is marked as a service job.

```
qsub -l walltime=100:00:00,nodes=1 -s y service_job.py
```

The example above submits a job to TORQUE with a walltime of 100 hours, one node, and it is marked as a service job. The script "service_job.py" will be used to start, stop, and check the status of the service/workload started on the compute nodes.

Moab, as of version 5.4, is able to accept the "-s y" option when msub is used for submission. Moab will then pass this information to TORQUE when the job is migrated.

Related topics

- **Service jobs** on page 61

2.8.2 Submitting service jobs in MCM

Submitting a service job in MCM requires the latest Adaptive Computing Suite snapshot of MCM. It also requires MCM to be started with the `"--future=2"` option.

Once MCM is started, open the **Create Workload** window and verify **Show Advanced Options** is checked. Notice that there is a **Service** checkbox that can be selected in the **Flags/Options** area. Use this to specify the job is a service job.

Related topics

- **Service jobs** on page 61

2.8.3 Managing service jobs

Managing a service job is done much like any other job; only a few differences exist.

Examining the job with **qstat** `-f` will reveal that the job has the `service = True` attribute. Non-service jobs will not make any mention of the "service" attribute.

Canceling a service job is done with **qdel**, mjobctl `-c`, or through any of the GUI's as with any other job. TORQUE, however, cancels the job by calling the service script with the "stop" argument instead of killing it directly. This behavior also occurs if the job runs over its wallclock and TORQUE/Moab is configured to cancel the job.

If a service job completes when the script exits after calling it with "start," or if TORQUE invokes the script with "status" and does not get back "running," it will **not** be terminated by using the "stop" argument.

Related topics

- **Service jobs** on page 61

Chapter 3: Managing nodes

This section contains information about adding and configuring compute nodes. It explains how to work with host security for systems that require dedicated access to compute nodes. It also contains information about scheduling specific cores on a node at job submission.

For details, see these topics:

- **Adding nodes** on page 63
- **Node properties** on page 64
- **Changing node state** on page 65
- **Host security** on page 65
- **Linux cpuset support** on page 66
- **Scheduling cores** on page 68
- **Scheduling GPUs** on page 69

3.1 Adding nodes

TORQUE can add and remove nodes either dynamically with **qmgr** or by manually editing the `TORQUE_HOME/server_priv/nodes` file (see **Initializing/Configuring TORQUE on the server (pbs_server)** on page 5).

Run-time node changes

TORQUE can dynamically add nodes with the `qmgr` command. For example, the following command will add node **node003**:

```
> qmgr -c "create node node003"
```

The above command appends the `$TORQUE_HOME/server_priv/nodes` file with:

```
node003
```

Nodes can also be removed with a similar command:

```
> qmgr -c "delete node node003"
```

Typically, an administrator will want to change the state of a node instead of remove it (for details, see **Changing node state** on page 65).

It is highly recommended that you restart the pbs_server after you make any node changes, or just edit the nodes file manually and restart it.

Related topics

- **Managing nodes** on page 63

3.2 Node properties

TORQUE can associate properties with nodes to aid in identifying groups of nodes. It's typical for a site to conglomerate a heterogeneous sets of resources. To identify the different sets, properties can be given to each node in a set. For example, a group of nodes that has a higher speed network connection could have the property "ib". TORQUE can set, update, or remove properties either dynamically with **qmgr** or by manually editing the `nodes` file.

Run-time node changes

TORQUE can dynamically change the properties of a node with the `qmgr` command. For example, note the following to give **node001** the properties of "bigmem" and "dualcore":

```
> qmgr -c "set node node001 properties = bigmem"
> qmgr -c "set node node001 properties += dualcore"
```

To relinquish a stated property, use the "-=" operator.

Manual node changes

The properties of each node are enumerated in `TORQUE_HOME/server_priv/nodes`. The feature(s) must be in a space delimited list after the node name. For example, to give **node001** the properties of "bigmem" and "dualcore" and **node002** the properties of "bigmem" and "matlab," edit the nodes file to contain the following:

`server_priv/nodes`:

```
node001 bigmem dualcore
node002 np=4 bigmem matlab
```

For changes to the nodes file to be activated, pbs_server must be restarted.

For a full description of this file, please see the *PBS Administrator Guide*.

Related topics

- **Job submission** on page 35
- **Managing nodes** on page 63

3.3 Changing node state

A common task is to prevent jobs from running on a particular node by marking it **offline** with `pbsnodes -o nodename`. Once a node has been marked offline, the scheduler will no longer consider it available for new jobs. Simply use `pbsnodes -c nodename` when the node is returned to service.

Also useful is `pbsnodes -l`, which lists all nodes with an interesting state, such as down, unknown, or offline. This provides a quick glance at nodes that might be having a problem. (See **pbsnodes** for details.)

Related topics

- **Managing nodes** on page 63

3.4 Host security

Enabling PAM with TORQUE

TORQUE is able to take advantage of the authentication services provided through Pluggable Authentication Modules (PAM) to help administrators manage access to compute nodes by users. The PAM module available in TORQUE is located in the PAM security directory. This module, when used in conjunction with other PAM modules, restricts access to the compute node unless the user has a job currently running on the node.

To enable TORQUE PAM configure TORQUE using the `--with-pam` option. Using `--with-pam` is sufficient but if your PAM security modules are not in the default `/lib/security` or `/lib64/security` directory, you can specify the location using `--with-pam=`*<DIR>* where *<DIR>* is the directory where you want the modules to be installed. When TORQUE is installed the files `pam_pbssimpleauth.la` and `pam_pbssimpleauth.so` appear in `/lib/security`, `/lib64/security`, or the directory designated on the configuration line.

PAM is very flexible and policies vary greatly from one site to another. The following example restricts users trying to access a node using SSH. Administrators need to assess their own installations and decide how to apply the TORQUE PAM restrictions.

After installing TORQUE with PAM enabled, add the following two lines to `/etc/pam.c/sshd`:

```
account required pam_pbssimpleauth.so
account required pam_access.so
```

In `/etc/security/access.conf` make sure all users who access the compute node are added to the configuration. This is an example which allows the users root, george, allen, and michael access.

```
-:ALL EXCEPT root george allen michael torque:ALL
```

With this configuration, if user george has a job currently running on the compute node, george can use ssh to login to the node. If there are currently no jobs running, george is disconnected when attempting to login.

TORQUE PAM is good at keeping users out who do not have jobs running on a compute node. However, it does not have the ability to force a user to log out once they are in. To accomplish this use epilogue or prologue scripts to force users off the system.

Legacy TORQUE PAM configuration

There is an alternative PAM configuration for TORQUE that has been available since 2006. It can be found in the `contrib/pam_authuser` directory of the source tree. Adaptive Computing does not currently support this method but the instructions are given here for those who are currently using it and for those who wish to use it.

For systems requiring dedicated access to compute nodes (for example, users with sensitive data), TORQUE prologue and epilogue scripts provide a vehicle to leverage the authentication provided by linux-PAM modules. (See **Prologue and epilogue scripts** on page 257 for more information.)

To allow only users with running jobs (and root) to access compute nodes

1. Untar `contrib/pam_authuser.tar.gz` (found in the src tar ball).
2. Compile `pam_authuser.c` with `make` and `make install` on every compute node.
3. Edit `/etc/system-auth` as described in `README.pam_authuser`, again on every compute node.
4. Either make a tarball of the epilogue* and prologue* scripts (to preserve the symbolic link) and untar it in the `mom_priv` directory, or just copy epilogue* and prologue* to `mom_priv/`.

The prologue* scripts are Perl scripts that add the user of the job to `/etc/authuser`. The epilogue* scripts then remove the first occurrence of the user from `/etc/authuser`. File locking is employed in all scripts to eliminate the chance of race conditions. There is also some commented code in the epilogue* scripts, which, if uncommented, kills all processes owned by the user (using pkill), provided that the user doesn't have another valid job on the same node.

prologue and **epilogue** scripts were added to the `pam_authuser` tarball in version 2.1 of TORQUE.

Related topics

- **Managing nodes** on page 63

3.5 Linux cpuset support

- **Cpuset overview** on page 67
- **Cpuset support** on page 67
- **Cpuset configuration** on page 67
- **Cpuset advantages / disadvantages** on page 67

Cpuset overview

Linux kernel 2.6 Cpusets are logical, hierarchical groupings of CPUs and units of memory. Once created, individual processes can be placed within a cpuset. The processes will only be allowed to run/access the specified CPUs and memory. Cpusets are managed in a virtual file system mounted at `/dev/cpuset`. New cpusets are created by simply making new directories. Cpusets gain CPUs and memory units by simply writing the unit number to files within the cpuset.

Cpuset support

All nodes using cpusets must have the `hwloc` library (version 1.1 or higher) installed.

When started, pbs_mom will create an initial top-level cpuset at `/dev/cpuset/torque`. This cpuset contains all CPUs and memory of the host machine. If this "torqueset" already exists, it will be left unchanged to allow the administrator to override the default behavior. All subsequent cpusets are created within the torqueset.

When a job is started, the jobset is created at `/dev/cpuset/torque/$jobid` and populated with the CPUs listed in the `exec_host job` attribute. Also created are individual tasksets for each CPU within the jobset. This happens before prologue, which allows it to be easily modified, and it happens on all nodes.

The top-level batch script process is executed in the jobset. Tasks launched through the TM interface (pbsdsh and PW's mpiexec) will be executed within the appropriate taskset.

On job exit, all tasksets and the jobset are deleted.

Cpuset configuration

To configure cpuset

1. As root, mount the virtual filesystem for cpusets:

   ```
   mount -t cpuset none /dev/cpuset
   ```

   Do this for each MOM that is to use cpusets.

2. Because cpuset usage is a build-time option in TORQUE, you must add `-enable-cpuset` to your configure options:

   ```
   ./configure --enable-cpuset
   ```

3. Use this configuration for the MOMs across your system.

Cpuset advantages / disadvantages

Presently, any job can request a single CPU and proceed to use everything available in the machine. This is occasionally done to circumvent policy, but most often is simply an error on the part of the user. Cpuset support will easily constrain the processes to not interfere with other jobs.

Jobs on larger NUMA systems may see a performance boost if jobs can be intelligently assigned to specific CPUs. Jobs may perform better if striped across physical processors, or contained within the fewest number of memory controllers.

TM tasks are constrained to a single core, thus a multi-threaded process could seriously suffer.

Related topics

- **Managing nodes** on page 63
- **Geometry request configuration** on page 68

3.6 Scheduling cores

In TORQUE 2.4 and later, you can request specific cores on a node at job submission by using geometry requests. To use this feature, specify the **procs_bitmap** resource request of `qsub -l` (see **qsub**) at job submission.

For details about scheduling cores, see these topics:

- **Geometry request configuration** on page 68
- **Geometry request usage** on page 69
- **Geometry request considerations** on page 69

3.6.1 Geometry request configuration

A Linux kernel of 2.6 or later is required to use geometry requests, because this feature uses Linux cpusets in its implementation. In order to use this feature, the cpuset directory has to be mounted. For more information on how to mount the cpuset directory, see **Linux cpuset support** on page 66. If the operating environment is suitable for geometry requests, configure TORQUE with the `--enable-geometry-requests` option.

```
> ./configure --prefix=/home/john/torque --enable-geometry-requests
```

TORQUEis configured to install to `/home/john/torque` and to enable the geometry requests feature.

The geometry request feature uses a subset of the cpusets feature. When you configure TORQUE using `--enable-cpuset` and `--enable-geometry-requests` at the same time, and use `-l procs_bitmap=X`, the job will get the requested cpuset. Otherwise, the job is treated as if only `--enable-cpuset` was configured.

Related topics

- **Scheduling cores** on page 68

3.6.2 Geometry request usage

Once enabled, users can submit jobs with a geometry request by using the `procs_bitmap=<string>` resource request. **procs_bitmap** requires a numerical string made up of 1's and 0's. A 0 in the bitmap means the job can not run on the core that matches the 0's index in the bitmap. The index is in reverse order of the number of cores available. If a job is submitted with `procs_bitmap=1011`, then the job requests a node with four free cores, and uses only cores one, two, and four.

> The geometry request feature requires a node that has all cores free. A job with a geometry request cannot run on a node that has cores that are busy, even if the node has more than enough cores available to run the job.

```
qsub -l procs_bitmap=0011 ossl.sh
```

The job **ossl.sh** is submitted with a geometry request of **0011**.

In the above example, the submitted job can run only on a node that has four cores. When a suitable node is found, the job runs exclusively on cores one and two.

Related topics

- **Scheduling cores** on page 68

3.6.3 Geometry request considerations

As previously stated, jobs with geometry requests require a node with all of its cores available. After the job starts running on the requested cores, the node cannot run other jobs, even if the node has enough free cores to meet the requirements of the other jobs. Once the geometry requesting job is done, the node is available to other jobs again.

Related topics

- **Scheduling cores** on page 68

3.7 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 (see **Server node file configuration** on page 24). 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).

For more information, see these topics:

- **Using GPUs with NUMA** on page 70
- **TORQUE NVIDIA GPGPUs** on page 71

3.7.1 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** on page 69
- **TORQUE NVIDIA GPGPUs** on page 71

3.7.2 TORQUE NVIDIA GPGPUs

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 a NVIDAI 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)

For example, you would configure the a PBS_SERVER that does not have GPUs, but will be managing compute nodes with NVIDIA GPUs in this way:

Server

```
./configure --with-debug --with-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.)

For more information, see **Compute nodes** on page 3.

When updating, it is good practice to stop the pbs_server and make a backup of the TORQUE home directory. You will also want to backup the output of qmgr -c 'p s'. The update will only overwrite the binaries.

For further details, see these topics:

- **TORQUE configuration** on page 72
- **GPU modes for NVIDIA 260.x driver** on page 73
- **GPU Modes for NVIDIA 270.x driver** on page 73
- **gpu_status** on page 73
- **New NVIDIA GPU support** on page 73

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 69
- **Using GPUs with NUMA** on page 70

Chapter 4: Setting server policies

This section explains how to set up and configure your queue. It lists the queue attributes and describes how to set up a routing queue. This section also explains how to set up TORQUE to run in high availability mode. For details, see these topics:

- **Queue configuration** on page 75
- **Server high availability** on page 88

4.1 Queue configuration

Under TORQUE, queue configuration is accomplished using the **Server high availability** command. With this tool, the first step is to create the queue. This is accomplished using the `create` subcommand of qmgr as in the following example:

```
> qmgr -c "create queue batch queue_type=execution"
```

Once created, the queue must be configured to be operational. At a minimum, this includes setting the options **started** and **enabled**. Further configuration is possible using any combination of the attributes listed in what follows.

For Boolean attributes, **T**, **t**, **1**, **Y**, and **y** are all synonymous with "TRUE," and **F**, **f**, **0**, **N**, and **n** all mean "FALSE."

For **queue_type**, **E** and **R** are synonymous with "Execution" and "Routing" (respectively).

See these topics for more details:

- **Queue attributes** on page 76
- **Example queue configuration** on page 85
- **Setting a default queue** on page 85
- **Mapping a queue to subset of resources** on page 86
- **Creating a routing queue** on page 86

Related topics

- **Server parameters** on page 209
- **qalter** on page 159 - command which can move jobs from one queue to another

4.1.1 Queue attributes

This section lists the following queue attributes:

- **acl_groups** on page 77
- **acl_group_enable** on page 77
- **acl_group_sloppy** on page 77
- **acl_hosts** on page 78
- **acl_host_enable** on page 78
- **acl_logic_or** on page 78
- **acl_users** on page 78
- **acl_user_enable** on page 79
- **disallowed_types** on page 79
- **enabled** on page 79
- **keep_completed** on page 80
- **kill_delay** on page 80
- **max_queuable** on page 80
- **max_running** on page 81
- **max_user_queuable** on page 81
- **max_user_run** on page 81
- **priority** on page 81
- **queue_type** on page 82
- **resources_available** on page 82
- **resources_default** on page 82
- **resources_max** on page 83
- **resources_min** on page 83
- **route_destinations** on page 83
- **started** on page 84

This section also lists some queue resource limits (see **Assigning queue resource limits** on page 84).

For Boolean attributes, **T**, **t**, **1**, **Y**, and **y** are all synonymous with "TRUE," and **F**, **f**, **0**, **N**, and **n** all mean "FALSE."

acl_groups	
Format	<GROUP>[@<HOST>][+<USER>[@<HOST>]]...
Default	---
Description	Specifies the list of groups which may submit jobs to the queue. If acl_group_enable is set to true, only users with a primary group listed in acl_groups may utilize the queue. If the PBSACLUSEGROUPLIST variable is set in the pbs_server environment, acl_groups checks against all groups of which the job user is a member.

Example	`> qmgr -c "set queue batch acl_groups=staff"`
	`> qmgr -c "set queue batch acl_groups+=ops@h2"`
	`> qmgr -c "set queue batch acl_groups+=staff@h3"`
	Used in conjunction with **acl_group_enable**.

acl_group_enable	
Format	<BOOLEAN>
Default	FALSE
Description	If **TRUE**, constrains TORQUE to only allow jobs submitted from groups specified by the **acl_groups** parameter.
Example	`qmgr -c "set queue batch acl_group_enable=true"`

acl_group_sloppy	
Format	<BOOLEAN>
Default	FALSE
Description	If **TRUE**, **acl_groups** will be checked against all groups of which the job users is a member.
Example	---

acl_hosts	
Format	<HOST>[+<HOST>]...
Default	---
Description	Specifies the list of hosts that may submit jobs to the queue.
Example	`qmgr -c "set queue batch acl_hosts=h1+h2+h3"` Used in conjunction with acl_host_enable.

acl_host_enable	
Format	<BOOLEAN>
Default	FALSE
Description	If **TRUE**, constrains TORQUE to only allow jobs submitted from hosts specified by the acl_hosts parameter.
Example	`qmgr -c "set queue batch acl_logic_or=true"`

acl_logic_or	
Format	<BOOLEAN>
Default	FALSE
Description	If **TRUE**, user and group acls are logically OR'd together, meaning that either acl may be met to allow access. If FALSE or unset, then both acls are AND'd, meaning that both acls must be satisfied.
Example	`qmgr -c "set queue batch acl_logic_or=true"`

acl_users	
Format	<USER>[@<HOST>][+<USER>[@<HOST>]]...
Default	---

acl_users

Description	Specifies the list of users who may submit jobs to the queue. If **acl_user_enable** is set to **TRUE**, only users listed in acl_users may use the queue.

Example	`> qmgr -c "set queue batch acl_users=john"`
	`> qmgr -c "set queue batch acl_users+=steve@h2"`
	`> qmgr -c "set queue batch acl_users+=stevek@h3"`
	Used in conjunction with **acl_user_enable**.

acl_user_enable

Format	<BOOLEAN>
Default	FALSE
Description	If **TRUE**, constrains TORQUE to only allow jobs submitted from users specified by the **acl_users** parameter.
Example	`qmgr -c "set queue batch acl_user_enable=true"`

disallowed_types

Format	<type>[+<type>]...
Default	---
Description	Specifies classes of jobs that are not allowed to be submitted to this queue. Valid types are interactive, batch, rerunable, nonrerunable, fault_tolerant (as of version 2.4.0 and later), fault_intolerant (as of version 2.4.0 and later), and job_array (as of version 2.4.1 and later).
Example	`qmgr -c "set queue batch disallowed_types = interactive"` `qmgr -c "set queue batch disallowed_types += job_array"`

enabled

Format	<BOOLEAN>
Default	FALSE

enabled	
Description	Specifies whether the queue accepts new job submissions.
Example	`qmgr -c "set queue batch enabled=true"`

keep_completed	
Format	<INTEGER>
Default	0
Description	Specifies the number of seconds jobs should be held in the Completed state after exiting.
Example	`qmgr -c "set queue batch keep_completed=120"`

kill_delay	
Format	<INTEGER>
Default	2
Description	Specifies the number of seconds between sending a SIGTERM and a SIGKILL to a job in a specific queue that you want to cancel.
Example	`qmgr -c "set queue batch kill_delay=30"`

max_queuable	
Format	<INTEGER>
Default	unlimited
Description	Specifies the maximum number of jobs allowed in the queue at any given time (includes idle, running, and blocked jobs).
Example	`qmgr -c "set queue batch max_queuable=20"`

max_running	
Format	<INTEGER>
Default	unlimited
Description	Specifies the maximum number of jobs in the queue allowed to run at any given time.
Example	`qmgr -c "set queue batch max_running=20"`

max_user_queuable	
Format	<INTEGER>
Default	unlimited
Description	Specifies the maximum number of jobs, per user, allowed in the queue at any given time (includes idle, running, and blocked jobs). Version 2.1.3 and greater.
Example	`qmgr -c "set queue batch max_user_queuable=20"`

max_user_run	
Format	<INTEGER>
Default	unlimited
Description	Specifies the maximum number of jobs, per user, in the queue allowed to run at any given time.
Example	`qmgr -c "set queue batch max_user_run=10"`

priority	
Format	<INTEGER>
Default	0
Description	Specifies the priority value associated with the queue.

priority	
Example	`qmgr -c "set queue batch priority=20"`

queue_type	
Format	One of **e**, **execution**, **r**, or **route** (see **Creating a routing queue** on page 86)
Default	---
Description	Specifies the queue type. ⚠ This value must be explicitly set for all queues.
Example	`qmgr -c "set queue batch queue_type=execution"`

resources_available	
Format	<STRING>
Default	---
Description	Specifies to cumulative resources available to all jobs running in the queue.
Example	`qmgr -c "set queue batch resources_available.nodect=20"` ℹ You must restart pbs_server for changes to take effect. Also, resources_available is constrained by the smallest of queue.resources_available and server.resources_available.

resources_default	
Format	<STRING>
Default	---
Description	Specifies default resource requirements for jobs submitted to the queue.
Example	`qmgr -c "set queue batch resources_default.walltime=3600"`

resources_max	
Format	<STRING>
Default	---
Description	Specifies the maximum resource limits for jobs submitted to the queue.
Example	`qmgr -c "set queue batch resources_max.nodect=16"`

resources_min	
Format	<STRING>
Default	---
Description	Specifies the minimum resource limits for jobs submitted to the queue.
Example	`qmgr -c "set queue batch resources_min.nodect=2"`

route_destinations	
Format	<queue>[@<host>]
Default	---
Description	Specifies the potential destination queues for jobs submitted to the associated routing queue. This attribute is only valid for routing queues (see **Creating a routing queue** on page 86).

Example	`> qmgr -c "set queue route route_destinations=fast"`
	`> qmgr -c "set queue route route_destinations+=slow"`
	`> qmgr -c "set queue route route_destinations+=medium@hostname"`
	To set multiple queue specifications, use multiple commands:
	`> qmgr -c 's s route_destinations=batch'`
	`> qmgr -c 's s route_destinations+=long'`
	`> qmgr -c 's s route_destinations+=short'`

started	
Format	<BOOLEAN>
Default	FALSE
Description	Specifies whether jobs in the queue are allowed to execute.
Example	`qmgr -c "set queue batch started=true"`

Assigning queue resource limits

Administrators can use resources limits to help direct what kind of jobs go to different queues. There are four queue attributes where resource limits can be set: **resources_available**, **resources_default**, **resources_max**, and **resources_min**. The list of supported resources that can be limited with these attributes are **arch**, **mem**, **ncpus**, **nodect**, **nodes**, **procct**, **pvmemvmem**, and **walltime**.

Resource	Format	Description
arch	string	Specifies the administrator defined system architecture required.
mem	**size**	Amount of physical memory used by the job. (Ignored on Darwin, Digital Unix, Free BSD, HPUX 11, IRIX, NetBSD, and SunOS. Also ignored on Linux if number of nodes is not 1. Not implemented on AIX and HPUX 10.)
nodect	integer	Sets the number of nodes available. By default, TORQUE will set the number of nodes available to the number of nodes listed in the `$TORQUE_HOME/server_priv/nodes` file. nodect can be set to be greater than or less than that number. Generally, it is used to set the node count higher than the number of physical nodes in the cluster.
nodes	integer	Specifies the number of nodes.
procct	integer	Sets limits on the total number of execution slots (procs) allocated to a job. The number of procs is calculated by summing the products of all node and ppn entries for a job. For example `qsub -l nodes=2:ppn=2+3:ppn=4 job.sh` would yield a procct of 16. 2*2 (2:ppn=2) + 3*4 (3:ppn=4).
pvmem	**size**	Amount of virtual memory used by any single process in a job.
vmem	**size**	Amount of virtual memory used by all concurrent processes in the job.
walltime	seconds, or [[HH:] MM:]SS	Amount of real time during which a job can be in a running state.

size

The size format specifies the maximum amount in terms of bytes or words. It is expressed in the form *integer[suffix]*. The suffix is a multiplier defined in the following table ("b" means bytes [the default] and "w" means words). The size of a word is calculated on the execution server as its word size.

Suffix		Multiplier
b	w	1
kb	kw	1024
mb	mw	1,048,576
gb	gw	1,073,741,824
tb	tw	1,099,511,627,776

Related topics

- **Queue configuration** on page 75
- **Example queue configuration** on page 85

4.1.2 Example queue configuration

The following series of **qmgr** commands will create and configure a queue named batch:

```
qmgr -c "create queue batch queue_type=execution"
qmgr -c "set queue batch started=true"
qmgr -c "set queue batch enabled=true"
qmgr -c "set queue batch resources_default.nodes=1"
qmgr -c "set queue batch resources_default.walltime=3600"
```

This queue will accept new jobs and, if not explicitly specified in the job, will assign a nodecount of 1 and a walltime of 1 hour to each job.

Related topics

- **Queue configuration** on page 75

4.1.3 Setting a default queue

By default, a job must explicitly specify which queue it is to run in. To change this behavior, the server parameter **default_queue** may be specified as in the following example:

```
qmgr -c "set server default_queue=batch"
```

Related topics

- **Queue configuration** on page 75

4.1.4 Mapping a queue to subset of resources

TORQUE does not currently provide a simple mechanism for mapping queues to nodes. However, schedulers such as Moab and Maui can provide this functionality.

The simplest method is using `default_resources.neednodes` on an execution queue, setting it to a particular node attribute. Maui/Moab will use this information to ensure that jobs in that queue will be assigned nodes with that attribute. For example, suppose we have some nodes bought with money from the chemistry department, and some nodes paid by the biology department.

```
$TORQUE_HOME/server_priv/nodes:
node01 np=2 chem
node02 np=2 chem
node03 np=2 bio
node04 np=2 bio
qmgr:
set queue chem resources_default.neednodes=chem
set queue bio resources_default.neednodes=bio
```

This example does not preclude other queues from accessing those nodes. One solution is to use some other generic attribute with all other nodes and queues.

More advanced configurations can be made with standing reservations and QoSes.

Related topics

- **Queue configuration** on page 75

4.1.5 Creating a routing queue

A routing queue will steer a job to a destination queue based on job attributes and queue constraints. It is set up by creating a queue of **queue_type** "Route" with a **route_destinations** attribute set, as in the following example.

```
qmgr

# routing queue
create queue route
set queue route queue_type = Route
set queue route route_destinations = reg_64
set queue route route_destinations += reg_32
set queue route route_destinations += reg
set queue route enabled = True
set queue route started = True

# queue for jobs using 1-15 nodes
create queue reg
set queue reg queue_type = Execution
set queue reg resources_min.ncpus = 1
set queue reg resources_min.nodect = 1
set queue reg resources_default.ncpus = 1
set queue reg resources_default.nodes = 1
set queue reg enabled = True
set queue reg started = True

# queue for jobs using 16-31 nodes
create queue reg_32
set queue reg_32 queue_type = Execution
set queue reg_32 resources_min.ncpus = 31
set queue reg_32 resources_min.nodes = 16
set queue reg_32 resources_default.walltime = 12:00:00
set queue reg_32 enabled = True
set queue reg_32 started = True

# queue for jobs using 32+ nodes
create queue reg_64
set queue reg_64 queue_type = Execution
set queue reg_64 resources_min.ncpus = 63
set queue reg_64 resources_min.nodes = 32
set queue reg_64 resources_default.walltime = 06:00:00
set queue reg_64 enabled = True
set queue reg_64 started = True

# have all jobs go through the routing queue
set server default_queue = batch
set server resources_default.ncpus = 1
set server resources_default.walltime = 24:00:00
  ...
```

In this example, the compute nodes are dual processors and default walltimes are set according to the number of processors/nodes of a job. Jobs with 32 nodes (63 processors) or more will be given a default walltime of 6 hours. Also, jobs with 16-31 nodes (31-62 processors) will be given a default walltime of 12 hours. All other jobs will have the server default walltime of 24 hours.

The ordering of the route_destinations is important. In a routing queue, a job is assigned to the first possible destination queue based on the **resources_max**, **resources_min**, **acl_users**, and **acl_groups** attributes. In the preceding example, the attributes of a single processor job would first be checked against the reg_64 queue, then the reg_32 queue, and finally the reg queue.

Adding the following settings to the earlier configuration elucidates the queue resource requirements:

```
qmgr

set queue reg resources_max.ncpus = 30
set queue reg resources_max.nodect = 15
set queue reg_16 resources_max.ncpus = 62
set queue reg_16 resources_max.ncpus = 31
```

The time of enforcement of server and queue defaults is important in this example. TORQUE applies server and queue defaults differently in job centric and queue centric modes. For job centric mode, TORQUE waits to apply the server and queue defaults until the job is assigned to its final execution queue. For queue centric mode, it enforces server defaults before it is placed in the routing queue. In either mode, queue defaults override the server defaults. TORQUE defaults to job centric mode. To set queue centric mode, set `queue_centric_limits`, as in what follows:

```
qmgr

set server queue_centric_limits = true
```

An artifact of job centric mode is that if a job does not have an attribute set, the server and routing queue defaults are not applied when queue resource limits are checked. Consequently, a job that requests 32 nodes (not ncpus=32) will not be checked against a min_resource.ncpus limit. Also, for the preceding example, a job without any attributes set will be placed in the reg_64 queue, since the server ncpus default will be applied after the job is assigned to an execution queue.

Routine queue defaults are not applied to job attributes in versions 2.1.0 and before.

If the error message "`qsub: Job rejected by all possible destinations`" is reported when submitting a job, it may be necessary to add queue location information, (i.e., in the routing queue's **route_destinations** attribute, change "batch" to "batch@localhost").

Related topics

- **Queue configuration** on page 75
- **Queue attributes** on page 76

4.2 Server high availability

You can now run TORQUE in a redundant or high availability mode. This means that there can be multiple instances of the server running and waiting to take over processing in the event that the currently running server fails.

The high availability feature is available in the 2.3 and later versions of TORQUE. TORQUE 2.4 includes several enhancements to high availability (see **Enhanced high availability** on page 89).

For more details, see these sections:

- **Redundant server host machines** on page 89
- **Enhanced high availability** on page 89
- **Enhanced high availability with Moab** on page 90
- **How commands select the correct server host** on page 90
- **Job names** on page 91
- **Persistence of the pbs_server process** on page 91

- **High availability of the NFS server** on page 91
- **Example setup of high availability** on page 91

Redundant server host machines

High availability enables TORQUE to continue running even if pbs_server is brought down. This is done by running multiple copies of pbs_server which have their `torque/server_priv` directory mounted on a shared file system. The `torque/server_name` must include the host names of all nodes that run pbs_server. All MOM nodes also must include the host names of all nodes running pbs_server in their `torque/server_name` file. The syntax of the `torque/server_name` is a comma delimited list of host names.

For example:

```
host1,host2,host3
```

When configuring high availability, do not use $pbsserver to specify the host names. You must use the *$TORQUEHOMEDIR*`/server_name` file.

All instances of pbs_server need to be started with the `--ha` command line option that allows the servers to run at the same time. Only the first server to start will complete the full startup. The second server to start will block very early in the startup when it tries to lock the file `torque/server_priv/server.lock`. When the second server cannot obtain the lock, it will spin in a loop and wait for the lock to clear. The sleep time between checks of the lock file is one second.

Notice that not only can the servers run on independent server hardware, there can also be multiple instances of the pbs_server running on the same machine. This was not possible before as the second one to start would always write an error and quit when it could not obtain the lock.

Because the file `server_priv/serverdb` is created in a way which is not compatible between hardware architectures, the machines that are running pbs_server in high-availability mode must be of similar architecture. For example, a 32-bit machine is unable to read the `server_priv/serverdb` file of a 64-bit machine. Therefore, when choosing hardware, verify all servers are of the same architecture.

Enhanced high availability

The default high availability configuration of TORQUE 2.4 is backward compatible with version 2.3, but an enhanced high availability option is available with version 2.4. The enhanced version in 2.4 fixes some shortcomings in the default configuration and is more robust. The lock file mechanism used to trigger a fail-over in TORQUE 2.3 works correctly only if the primary pbs_server is taken down gracefully, and releases the lock on the file being used as the semaphore. If the server crashes, the lock stays in place and the backup server will not start unless the lock is manually removed by the administrator. With 2.4 enhanced high availability the reliance on the file system is bypassed with a much more reliable mechanism.

In order to use enhanced high availability with TORQUE 2.4, TORQUE must be configured using the `--enable-high-availability` option (in addition to all other configuration options you specify).

```
> ./configure --prefix=/usr/var/torque --enable-high-availability
```

This configuration option is not necessary in TORQUE 4.0 because high availability is enhanced high availability in TORQUE 4.0.

In the above example, TORQUE installs to the `/usr/var/torque` directory and is configured to use the high availability features.

Once TORQUE has been compiled and installed, it is launched the same way as with TORQUE 2.3; start each instance of pbs_server with the `--ha` option.

In addition to the new fail-over mechanism, three server options have been added to help manage enhanced high availability in TORQUE 2.4. The server parameters are **lock_file**, **lock_file_update_time**, and **lock_file_check_time**.

The lock_file option allows the administrator to change the location of the lock file. The default location is `torque/server_priv`. If the lock_file option is used, the new location must be on the shared partition so all servers have access.

The lock_file_update_time and lock_file_check_time parameters are used by the servers to determine if the primary server is active. The primary pbs_server will update the lock file based on the lock_file_update_time (default value of 3 seconds). All backup pbs_servers will check the lock file as indicated by the lock_file_check_time parameter (default value of 9 seconds). The lock_file_update_time must be less than the lock_file_check_time. When a failure occurs, the backup pbs_server takes up to the lock_file_check_time value to take over.

```
> qmgr -c "set server lock_file_check_time=5"
```

In the above example, after the primary pbs_server goes down, the backup pbs_server takes up to 5 seconds to take over. It takes additional time for all MOMs to switch over to the new pbs_server.

The clock on the primary and redundant servers must be synchronized in order for high availability to work. Use a utility such as NTP to ensure your servers have a synchronized time.

Enhanced high availability with Moab

When TORQUE is run with an external scheduler such as Moab, and the pbs_server is not running on the same host as Moab, pbs_server needs to know where to find the scheduler. To do this, use the following syntax (the port is required and the default is 15004):

```
> pbs_server --ha -l <moabhost:port>
```

If Moab is running in HA mode, add a `-l` option for each redundant server.

```
> pbs_server --ha -l <moabhost1:port> -l <moabhost2:port>
```

The root user of each Moab host must be added to the **operators** and **managers** lists of the server. This enables Moab to execute root level operations in TORQUE.

How commands select the correct server host

The various commands that send messages to pbs_server usually have an option of specifying the server name on the command line, or if none is specified will use the default server name. The default server name comes either from the environment variable **PBS_DEFAULT** or from the file `torque/server_name`.

When a command is executed and no explicit server is mentioned, an attempt is made to connect to the first server name in the list of hosts from PBS_DEFAULT or `torque/server_name`. If this fails, the next server name is tried. If all servers in the list are unreachable, an error is returned and the command fails.

Note that there is a period of time after the failure of the current server during which the new server is starting up where it is unable to process commands. The new server must read the existing configuration and job information from the disk, so the length of time that commands cannot be received varies. Commands issued during this period of time might fail due to timeouts expiring.

Job names

One aspect of this enhancement is in the construction of job names. Job names normally contain the name of the host machine where pbs_server is running. When job names are constructed, only the first name from the server specification list is used in building the job name.

Persistence of the pbs_server process

The system administrator must ensure that pbs_server continues to run on the server nodes. This could be as simple as a **cron** job that counts the number of pbs_server's in the process table and starts some more if needed.

High availability of the NFS server

One consideration of this implementation is that it depends on NFS file system also being redundant. NFS can be set up as a redundant service. See the following.

- Setting Up A Highly Available NFS Server
- Making NFS Work On Your Network
- Sourceforge Linux NFS FAQ
- NFS v4 main site

There are also other ways to set up a shared file system. See the following:

- Red Hat Global File System
- Data sharing with a GFS storage cluster

Example setup of high availability

1. The machines running pbs_server must have access to a shared `server_priv/` directory (usually an NFS share on a MoM).
2. All MoMs must have the same content in their `server_name` file. This can be done manually or via an NFS share. The `server_name` file contains a comma-delimited list of the hosts that run pbs_server.

```
# List of all servers running pbs_server
server1,server2
```

3. The machines running pbs_server must be listed in **acl_hosts**.

```
> qmgr -c "set server acl_hosts += server1"
> qmgr -c "set server acl_hosts += server2"
```

4. Start pbs_server with the --ha option.

```
[root@server1]$ pbs_server --ha

[root@server2]$ pbs_server --ha
```

Related topics

- **Setting server policies** on page 75
- **Queue configuration** on page 75

Chapter 5: Integrating schedulers for TORQUE

Selecting the cluster scheduler is an important decision and significantly affects cluster utilization, responsiveness, availability, and intelligence. The default TORQUE scheduler, pbs_sched, is very basic and will provide poor utilization of your cluster's resources. Other options, such as Maui Scheduler or Moab Workload Manager, are highly recommended. If you are using Maui or Moab, refer to the Moab-PBS Integration Guide. If using pbs_sched, simply start the pbs_sched daemon.

If you are installing Moab Cluster Suite, TORQUE and Moab were configured at installation for interoperability and no further action is required.

Chapter 6: Configuring data management

This section contains information about SCP-based data management with TORQUE. It describes how to use TORQUE with NFS and other networked filesystems. It also outlines file staging requirements. For details, see these topics:

- **SCP setup** on page 95
- **NFS and other networked filesystems** on page 98
- **File stage-in/stage-out** on page 99

6.1 SCP setup

To use SCP-based data management, TORQUE must be authorized to migrate data to any of the compute nodes. If this is not already enabled within the cluster, this can be achieved with the process described below. This process enables uni-directional access for a particular user from a *source* host to a *destination* host.

These directions were written using OpenSSH version 3.6 and may not transfer correctly to older versions.

To set up TORQUE for SCP, follow the directions in each of these topics:

- **Generating SSH key on source host** on page 96
- **Copying public SSH key to each destination host** on page 96
- **Configuring the SSH daemon on each destination host** on page 96
- **Validating correct SSH configuration** on page 97
- **Enabling bi-directional SCP access** on page 97
- **Compiling TORQUE to support SPC** on page 97
- **Troubleshooting** on page 98

Related topics

- **Configuring data management** on page 95

6.1.1 Generating SSH key on source host

On the source host as the transfer user, execute the following:

```
> ssh-keygen -t rsa
```

This will prompt for a passphrase (optional) and create two files (`id_rsa` and `id_rsa.pub`) inside `~/.ssh/`.

Related topics

- **SCP setup** on page 95
- **Copying public SSH key to each destination host** on page 96

6.1.2 Copying public SSH key to each destination host

Transfer public key to each destination host as the transfer user:

Easy key copy:

```
ssh-copy-id [-i [identity_file]] [user@]machine
```

Manual steps to copy keys:

```
> scp ~/.ssh/id_rsa.pub destHost:~ (enter password)
```

Create an authorized_keys file on each destination host:

```
> ssh destHost (enter password)
> cat id_rsa.pub >> .ssh/authorized_keys
```

If the .ssh directory does not exist, create it with 700 privileges (`mkdir .ssh`; `chmod 700 .ssh`):

```
> chmod 700 .ssh/authorized_keys
```

Related topics

- **Generating SSH key on source host** on page 96
- **SCP setup** on page 95

6.1.3 Configuring the SSH daemon on each destination host

Some configuration of the SSH daemon may be required on the destination host. (Because this is not always the case, see **Validating correct SSH configuration** on page 97 and test the changes made to this point. If the tests fail, proceed with this step and then try testing again.) Typically, this is done by editing the `/etc/ssh/sshd_config` file (root access needed). To verify correct configuration, see that the following attributes are set (not commented):

```
RSAAuthentication yes
PubkeyAuthentication yes
```

If configuration changes were required, the SSH daemon will need to be restarted (root access needed):

```
> /etc/init.d/sshd restart
```

Related topics

- **SCP setup** on page 95

6.1.4 Validating correct SSH configuration

If all is properly configured, the following command issued on the *source* host should succeed and not prompt for a password:

```
> scp destHost:/etc/motd /tmp
```

If this is your first time accessing *destination* from *source*, it may ask you if you want to add the fingerprint to a file of known hosts. If you specify yes, this message should no longer appear and should not interfere with scp copying via TORQUE. Also, it is important that the full hostname appear in the `known_hosts` file. To do this, use the full hostname for *destHost*, as in `machine.domain.org` instead of just `machine`.

Related topics

- **SCP setup** on page 95

6.1.5 Enabling bi-directional SCP access

The preceding steps allow *source* access to destination without prompting for a password. The reverse, however, is not true. Repeat the steps, but this time using the *destination* as the *source*, etc. to enable bi-directional SCP access (i.e. *source* can send to *destination* and *destination* can send to *source* without password prompts.)

Related topics

- **SCP setup** on page 95

6.1.6 Compiling TORQUE to support SPC

In TORQUE 2.1 and later, SCP is the default remote copy protocol. These instructions are only necessary for earlier versions.

TORQUE must be re-configured (and then rebuilt) to use SCP by passing in the `--with-scp` flag to the configure script:

```
> ./configure --prefix=xxx --with-scp
> make
```

If special SCP flags are required in your local setup, these can be specified using the $rcpcmd parameter.

Related topics

- **SCP setup** on page 95

6.1.7 Troubleshooting

If, after following all of the instructions in this section (see **SCP setup** on page 95), TORQUE is still having problems transferring data with SCP, set the `PBSDEBUG` environment variable and restart the pbs_mom for details about copying. Also check the MOM log files for more details.

Related topics

- **SCP setup** on page 95

6.2 NFS and other networked filesystems

When a batch job starts, its `stdin` file (if specified) is copied from the submission directory on the remote submission host. This file is placed in the `$PBSMOMHOME` directory on the mother superior node (i.e., `/usr/spool/PBS/spool`). As the job runs, stdout and stderr files are generated and placed in this directory using the naming convention **$JOBID.OU** and **$JOBID.ER**.

When the job completes, the MOM copies the files into the directory from which the job was submitted. By default, this file copying will be accomplished using a remote copy facility such as **rcp** or **scp**.

If a shared file system such as NFS, DFS, or AFS is available, a site can specify that the MOM should take advantage of this by specifying the `$usecp` directive inside the MOM configuration file (located in the `$PBSMOMHOME/mom_priv` directory) using the following format:

```
$usecp <HOST>:<SRCDIR> <DSTDIR>
```

`<HOST>` can be specified with a leading wildcard ('*') character. The following example demonstrates this directive:

```
mom_priv/config

# /home is NFS mounted on all hosts
$usecp *:/home /home
# submission hosts in domain fte.com should map '/data' directory on submit host to
# '/usr/local/data' on compute host
$usecp *.fte.com:/data /usr/local/data
```

If for any reason the MOM daemon is unable to copy the output or error files to the submission directory, these files are instead copied to the `undelivered` directory also located in `$PBSMOMHOME`.

Related topics

- **Configuring data management** on page 95

6.3 File stage-in/stage-out

File staging requirements are specified using the `stagein` and `stageout` directives of the **qsub** command. Stagein requests occur before the job starts execution, while stageout requests happen after a job completes.

On completion of the job, all staged-in and staged-out files are removed from the execution system. The `file_list` is in the `form local_file@hostname:remote_file[,...]` regardless of the direction of the copy. The name `local_file` is the name of the file on the system where the job executed. It may be an absolute path or relative to the home directory of the user. The name `remote_file` is the destination name on the host specified by hostname. The name may be absolute or relative to the user's home directory on the destination host. The use of wildcards in the file name is not recommended.

The file names map to a remote copy program (rcp/scp/cp, depending on configuration) called on the execution system in the following manner:

For stagein: `rcp/scp hostname:remote_file local_file`

For stageout: `rcp/scp local_file hostname:remote_file`

Examples

```
# stage /home/john/input_source.txt from node13.fsc to /home/john/input_destination.txt
on master compute node
> qsub -l nodes=1,walltime=100 -W stagein=input_source.txt@node13.fsc:/home/john/input_
destination.txt
```

```
stage /home/bill/output_source.txt on master compute node to /tmp/output_destination.txt
on node15.fsc
> qsub -l nodes=1,walltime=100 -W stageout=/tmp/output_
source.txt@node15.fsc:/home/bill/output_destination.txt
```

```
$ fortune >xxx;echo cat xxx|qsub -W stagein=xxx@`hostname`:xxx
199.myhost.mydomain
$ cat STDIN*199
Anyone who has had a bull by the tail knows five or six more things
than someone who hasn't.
-- Mark Twain
```

Related topics

- **Configuring data management** on page 95

Chapter 7: MPI (Message Passing Interface) support

A message passing library is used by parallel jobs to augment communication between the tasks distributed across the cluster. TORQUE can run with any message passing library and provides limited integration with some MPI libraries.

For more information, see these topics:

- **MPICH** on page 101
- **MPICH-VMI** on page 102
- **Open MPI** on page 103

7.1 MPICH

One of the most popular MPI libraries is MPICH available from Argonne National Lab. If using this release, you may want to consider also using the mpiexec tool for launching MPI applications. Support for mpiexec has been integrated into TORQUE.

MPIExec Overview

mpiexec is a replacement program for the script **mpirun**, which is part of the **mpich** package. It is used to initialize a parallel job from within a PBS batch or interactive environment. mpiexec uses the task manager library of PBS to spawn copies of the executable on the nodes in a PBS allocation.

Reasons to use mpiexec rather than a script (mpirun) or an external daemon (mpd):

- Starting tasks with the task manager (TM) interface is much faster than invoking a separate rsh * once for each process.
- Resources used by the spawned processes are accounted correctly with mpiexec, and reported in the PBS logs, because all the processes of a parallel job remain under the control of PBS, unlike when using mpirun-like scripts.
- Tasks that exceed their assigned limits of CPU time, wallclock time, memory usage, or disk space are killed cleanly by PBS. It is quite hard for processes to escape control of the resource manager when using mpiexec.
- You can use mpiexec to enforce a security policy. If all jobs are forced to spawn using mpiexec and the PBS execution environment, it is not necessary to enable rsh or ssh access to the compute nodes in the cluster.

For more information, see the mpiexec homepage.

MPIExec Troubleshooting

Although problems with mpiexec are rare, if issues do occur, the following steps may be useful:

- Determine current version using `mpiexec --version` and review the change log available on the MPI homepage to determine if the reported issue has already been corrected.
- Send email to the mpiexec mailing list at mpiexec@osc.edu.
- Browse the mpiexec user list archives for similar problems and resolutions.
- Read the FAQ contained in the `README` file and the mpiexec man pages contained within the mpiexec distribution.
- Increase the logging of mpiexec operation with `mpiexec --verbose` (reports messages to `stderr`).
- Increase logging of the master and slave resource manager execution daemons associated with the job (with TORQUE, use `$loglevel` to 5 or higher `in $TORQUEROOT/mom_priv/config` and look for '**tm**' messages after associated `join job` messages).
- Use `tracejob` (included with TORQUE) or `qtracejob` (included with OSC's pbstools package) to isolate failures within the cluster.
- If the message '`exec: Error: get_hosts: pbs_connect: Access from host not allowed, or unknown host`' appears, this indicates that mpiexec cannot communicate with the pbs_server daemon. In most cases, this indicates that the `$TORQUEROOT/server_name` file points to the wrong server or the node cannot resolve the server's name. The **qstat** command can be run on the node to test this.

General MPI Troubleshooting

When using MPICH, some sites have issues with orphaned MPI child processes remaining on the system after the master MPI process has been terminated. To address this, TORQUE epilogue scripts can be created that properly clean up the orphaned processes (see **Prologue and epilogue scripts** on page 257).

Related topics

- **MPI (Message Passing Interface) support** on page 101

7.2 MPICH-VMI

MPICH-VMI is a highly-optimized open-source message passing layer available from NCSA. Additional information can be found in the VMI tutorial.

Related topics

- **MPI (Message Passing Interface) support** on page 101

7.3 Open MPI

Open MPI is a new MPI implementation that combines technologies from multiple projects to create the best possible library. It supports the TM interface for integration with TORQUE. More information is available in the FAQ.

Related topics

- **MPI (Message Passing Interface) support** on page 101

Chapter 8: Resources

A primary task of any resource manager is to monitor the state, health, configuration, and utilization of managed resources. TORQUE is specifically designed to monitor compute hosts for use in a batch environment. TORQUE is not designed to monitor non-compute host resources such as software licenses, networks, file systems, and so forth, although these resources can be integrated into the cluster using some scheduling systems.

With regard to monitoring compute nodes, TORQUE reports about a number of attributes broken into three major categories:

- **Configuration** on page 105
- **Utilization** on page 106
- **Node states** on page 106

Configuration

Configuration includes both detected hardware configuration and specified batch attributes.

Attribute	Description	Details
Architecture (arch)	operating system of the node	The value reported is a derivative of the operating system installed.
Node Features (properties)	arbitrary string attributes associated with the node	No node features are specified by default. If required, they are set using the `nodes` file located in the `TORQUE_HOME/server_priv` directory. They may specify any string and are most commonly used to allow users to request certain subsets of nodes when submitting jobs.
Local Disk (size)	configured local disk	By default, local disk space is not monitored. If the MOM configuration size [fs=<FS>] parameter is set, TORQUE will report, in kilobytes, configured disk space within the specified directory.
Memory (physmem)	local memory/RAM	Local memory/RAM is monitored and reported in kilobytes.

Attribute	Description	Details
Processors (ncpus/np)	real/virtual processors	The number of processors detected by TORQUE is reported via the **ncpus** attribute. However, for scheduling purposes, other factors are taken into account. In its default configuration, TORQUE operates in "dedicated" mode with each node possessing a single virtual processor. In dedicated mode, each job task will consume one virtual processor and TORQUE will accept workload on each node until all virtual processors on that node are in use. While the number of virtual processors per node defaults to 1, this may be configured using the nodes file located in the `TORQUE_HOME/server_priv` directory. An alternative to dedicated mode is "timeshared" mode. If TORQUE's timeshared mode is enabled, TORQUE will accept additional workload on each node until the node's **maxload** limit is reached.
Swap (totmem)	virtual memory/Swap	Virtual memory/Swap is monitored and reported in kilobytes.

Utilization

Utilization includes information regarding the amount of node resources currently in use as well as information about who or what is consuming it.

Attribute	Description	Details
Disk (size)	local disk availability	By default, local disk space is not monitored. If the MOM configuration size [fs=<FS>] parameter is set, TORQUE will report configured and currently available disk space within the specified directory in kilobytes.
Memory (availmem)	real memory/RAM	Available real memory/RAM is monitored and reported in kilobytes.
Network (netload)	local network adapter usage	Reports total number of bytes transferred in or out by the network adapter.
Processor Utilization (loadave)	node's cpu load average	Reports the node's 1 minute bsd load average.

Node states

State information includes administrative status, general node health information, and general usage status.

Attribute	Description	Details
Idle Time (idletime)	time since local keyboard/mouse activity has been detected	Time in seconds since local keyboard/mouse activity has been detected.
State (state)	monitored/admin node state	A node can be in one or more of the following states: • **busy** - node is full and will not accept additional work • **down** - node is failing to report, is detecting local failures with node • **free** - node is ready to accept additional work • **job-exclusive** - all available virtual processors are assigned to jobs • **job-sharing** - node has been allocated to run multiple shared jobs and will remain in this state until jobs are complete • **offline** - node has been instructed by an admin to no longer accept work • **reserve** - node has been reserved by the server • **time-shared** - node always allows multiple jobs to run concurrently • **unknown** - node has not been detected

Chapter 9: Accounting records

TORQUE maintains accounting records for batch jobs in the following directory:

`$TORQUEROOT/server_priv/accounting/<TIMESTAMP>`

`$TORQUEROOT` defaults to `/usr/spool/PBS` and <*TIMESTAMP*> is in the format: *YYYYMMDD*.

These records include events, time stamps, and information on resources requested and used.

Records for four different event types are produced and are described in the following table:

Record marker	Record type	Description
A	abort	Job has been aborted by the server
C	checkpoint	Job has been checkpointed and held
D	delete	Job has been deleted
E	exit	Job has exited (either successfully or unsuccessfully)
Q	queue	Job has been submitted/queued
R	rerun	Attempt to rerun the job has been made
S	start	Attempt to start the job has been made (if the job fails to properly start, it may have multiple job start records)
T	restart	Attempt to restart the job (from checkpoint) has been made (if the job fails to properly start, it may have multiple job start records)

Accounting Variables

The following table offers accounting variable descriptions. Descriptions for accounting variables not indicated in the table, particularly those prefixed with **Resources_List**, are available at **Job submission** on page 35.

Variable	Description
ctime	Time job was created
etime	Time job became eligible to run
qtime	Time job was queued
start	Time job started to run

A sample record in this file can look like the following:

```
06/06/2005 14:04:25;D;408.ign1.zeta2.org;requestor=guest@ign1.zeta2.org
06/06/2005 14:04:35;Q;409.ign1.zeta2.org;queue=batch
06/06/2005 14:04:44;Q;410.ign1.zeta2.org;queue=batch
06/06/2005 14:06:06;S;407.ign1.zeta2.org;user=guest group=guest jobname=STDIN
queue=batch ctime=1118087915 qtime=1118087915 etime=1118087915 start=1118088366 exec_
host=ign1.zeta2.org/0 Resource_List.neednodes=ign1.zeta2.org Resource_List.nodect=1
Resource_List.nodes=1 Resource_List.walltime=00:16:40
06/06/2005 14:07:17;D;407.ign1.zeta2.org;requestor=guest@ign1.zeta2.org
06/06/2005 14:07:17;E;407.ign1.zeta2.org;user=guest group=guest jobname=STDIN
queue=batch ctime=1118087915 qtime=1118087915 etime=1118087915 start=1118088366 exec_
host=ign1.zeta2.org/0 Resource_List.nodect=1 Resource_List.nodes=1 Resource_
List.walltime=00:16:40 session=6365 end=1118088437 Exit_status=271 resources_
used.cput=00:00:00 resources_used.mem=3068kb resources_used.vmem=16080kb resources_
used.walltime=00:01:11
```

Chapter 10: Job logging

New in TORQUE 2.5.3 is the ability to log job information for completed jobs. The information stored in the log file is the same information produced with the command **qstat** `-f`. The log file data is stored using an XML format. Data can be extracted from the log using the utility `showjobs` found in the `contrib/` directory of the TORQUE source tree. Custom scripts that can parse the XML data can also be used.

For details about job logging, see these topics:

- **Job log location and name** on page 111
- **Enabling job logs** on page 111

10.1 Job log location and name

The job log is kept at `$TORQUE_HOME/job_logs`. The naming convention for the job log is the same as for the server log or MOM log. The log name is created from the current year/month/day.

For example, if today's date is 26 October, 2010 the log file is named 20101026.

A new log file is created each new day that data is written to the log.

Related topics

- **Enabling job logs** on page 111
- **Job logging** on page 111

10.2 Enabling job logs

There are five new server parameters used to enable job logging. These parameters control what information is stored in the log and manage the log files.

Parameter	Description
record_job_info	This must be set to true in order for job logging to be enabled. If not set to true, the remaining server parameters are ignored.

Parameter	Description
record_job_script	If set to true, this adds the contents of the script executed by a job to the log.
job_log_file_max_size	This specifies a soft limit (in kilobytes) for the job log's maximum size. The file size is checked every five minutes and if the **current day** file size is greater than or equal to this value, it is rolled from *<filename>* to *<filename.1>* and a new empty log is opened. If the current day file size exceeds the maximum size a second time, the *<filename.1>* log file is rolled to *<filename.2>*, the current log is rolled to *<filename.1>*, and a new empty log is opened. Each new log causes all other logs to roll to an extension that is one greater than its current number. Any value less than 0 is ignored by pbs_server (meaning the log will not be rolled).
job_log_file_roll_depth	This sets the maximum number of new log files that are kept in a day if the **job_log_file_max_size** parameter is set. For example, if the roll depth is set to 3, no file can roll higher than *<filename.3>*. If a file is already at the specified depth, such as *<filename.3>*, the file is deleted so it can be replaced by the incoming file roll, *<filename.2>*.
job_log_keep_days	This maintains logs for the number of days designated. If set to 4, any log file older than 4 days old is deleted.

Related topics

- **Job log location and name** on page 111
- **Job logging** on page 111

Chapter 11: Troubleshooting

There are a few general strategies that can be followed to determine the cause of unexpected behavior. These are a few of the tools available to help determine where problems occur. See these topics for details:

- **Host resolution** on page 113
- **Firewall configuration** on page 114
- **TORQUE log files** on page 114
- **Using "tracejob" to locate job failures** on page 114
- **Using GDB to locate job failures** on page 117
- **Other diagnostic options** on page 117
- **Stuck jobs** on page 118
- **Frequently asked questions (FAQ)** on page 119
- **Compute node health check** on page 124
- **Debugging** on page 126

11.1 Host resolution

The TORQUE server host must be able to perform both forward and reverse name lookup on itself and on all compute nodes. Likewise, each compute node must be able to perform forward and reverse name lookup on itself, the TORQUE server host, and all other compute nodes. In many cases, name resolution is handled by configuring the node's `/etc/hosts` file although **DNS** and **NIS** services may also be used. Commands such as `nslookup` or `dig` can be used to verify proper host resolution.

Invalid host resolution may exhibit itself with compute nodes reporting as down within the output of **pbsnodes**`-a` and with failure of the **momctl** `-d3` command.

Related topics

- **Troubleshooting** on page 113

11.2 Firewall configuration

Be sure that, if you have firewalls running on the server or node machines, you allow connections on the appropriate ports for each machine. TORQUE pbs_mom daemons use UDP ports 1023 and below if privileged ports are configured (privileged ports is the default). The pbs_server and pbs_mom daemons use TCP and UDP ports 15001-15004 by default.

Firewall based issues are often associated with server to MOM communication failures and messages such as 'premature end of message' in the log files.

Also, the `tcpdump` program can be used to verify the correct network packets are being sent.

Related topics

- **Troubleshooting** on page 113

11.3 TORQUE log files

The pbs_server keeps a daily log of all activity in the `TORQUE_HOME/server_logs` directory. The pbs_mom also keeps a daily log of all activity in the `TORQUE_HOME/mom_logs/` directory. These logs contain information on communication between server and MOM as well as information on jobs as they enter the queue and as they are dispatched, run, and terminated. These logs can be very helpful in determining general job failures. For MOM logs, the verbosity of the logging can be adjusted by setting the $loglevel parameter in the `mom_priv/config` file. For server logs, the verbosity of the logging can be adjusted by setting the server log_level attribute in **qmgr**.

For both pbs_mom and pbs_server daemons, the log verbosity level can also be adjusted by setting the environment variable **PBSLOGLEVEL** to a value between 0 and 7. Further, to dynamically change the log level of a running daemon, use the `SIGUSR1` and `SIGUSR2` signals to increase and decrease the active loglevel by one. Signals are sent to a process using the `kill` command.

For example, `kill -USR1 `pgrep pbs_mom`` would raise the log level up by one.

The current loglevel for pbs_mom can be displayed with the command **momctl** `-d3`.

Related topics

- **Troubleshooting** on page 113

11.4 Using "tracejob" to locate job failures

Overview

The **tracejob** utility extracts job status and job events from accounting records, MOM log files, server log files, and scheduler log files. Using it can help identify where, how, a why a job failed. This tool takes a job id as a parameter as well as arguments to specify which logs to search, how far into the past to search, and other conditions.

Syntax

```
tracejob [-a|s|l|m|q|v|z] [-c count] [-w size] [-p path] [ -n <DAYS>] [-f
filter_type] <JOBID>
```

```
-p  :  path to PBS_SERVER_HOME
-w  :  number of columns of your terminal
-n  :  number of days in the past to look for job(s) [default 1]
-f  :  filter out types of log entries, multiple -f's can be specified
       error, system, admin, job, job_usage, security, sched, debug,
       debug2, or absolute numeric hex equivalent
-z  :  toggle filtering excessive messages
-c  :  what message count is considered excessive
-a  :  don't use accounting log files
-s  :  don't use server log files
-l  :  don't use scheduler log files
-m  :  don't use MOM log files
-q  :  quiet mode - hide all error messages
-v  :  verbose mode - show more error messages
```

Example

```
> tracejob -n 10 1131

Job: 1131.icluster.org

03/02/2005 17:58:28  S    enqueuing into batch, state 1 hop 1
03/02/2005 17:58:28  S    Job Queued at request of dev@icluster.org, owner =
                          dev@icluster.org, job name = STDIN, queue = batch
03/02/2005 17:58:28  A    queue=batch
03/02/2005 17:58:41  S    Job Run at request of dev@icluster.org
03/02/2005 17:58:41  M    evaluating limits for job
03/02/2005 17:58:41  M    phase 2 of job launch successfully completed
03/02/2005 17:58:41  M    saving task (TMomFinalizeJob3)
03/02/2005 17:58:41  M    job successfully started
03/02/2005 17:58:41  M    job 1131.koa.icluster.org reported successful start on 1 node
(s)
03/02/2005 17:58:41  A    user=dev group=dev jobname=STDIN queue=batch ctime=1109811508
                          qtime=1109811508 etime=1109811508 start=1109811521
                          exec_host=icluster.org/0 Resource_List.neednodes=1 Resource_
List.nodect=1
                          Resource_List.nodes=1 Resource_List.walltime=00:01:40
03/02/2005 18:02:11  M    walltime 210 exceeded limit 100
03/02/2005 18:02:11  M    kill_job
03/02/2005 18:02:11  M    kill_job found a task to kill
03/02/2005 18:02:11  M    sending signal 15 to task
03/02/2005 18:02:11  M    kill_task: killing pid 14060 task 1 with sig 15
03/02/2005 18:02:11  M    kill_task: killing pid 14061 task 1 with sig 15
03/02/2005 18:02:11  M    kill_task: killing pid 14063 task 1 with sig 15
03/02/2005 18:02:11  M    kill_job done
03/02/2005 18:04:11  M    kill_job
03/02/2005 18:04:11  M    kill_job found a task to kill
03/02/2005 18:04:11  M    sending signal 15 to task
03/02/2005 18:06:27  M    kill_job
03/02/2005 18:06:27  M    kill_job done
03/02/2005 18:06:27  M    performing job clean-up
03/02/2005 18:06:27  A    user=dev group=dev jobname=STDIN queue=batch ctime=1109811508
                          qtime=1109811508 etime=1109811508 start=1109811521
                          exec_host=icluster.org/0 Resource_List.neednodes=1 Resource_
List.nodect=1
                          Resource_List.nodes=1 Resource_List.walltime=00:01:40
session=14060
                          end=1109811987 Exit_status=265 resources_used.cput=00:00:00
                          resources_used.mem=3544kb resources_used.vmem=10632kb
                          resources_used.walltime=00:07:46

...
```

The `tracejob` command operates by searching the pbs_server accounting records and the pbs_server, mom, and scheduler logs. To function properly, it must be run on a node and as a user which can access these files. By default, these files are all accessible by the user root and only available on the cluster management node. In particular, the files required by `tracejob` are located in the following directories:

`TORQUE_HOME/server_priv/accounting`

`TORQUE_HOME/server_logs`

`TORQUE_HOME/mom_logs`

`TORQUE_HOME/sched_logs`

`tracejob` may only be used on systems where these files are made available. Non-root users may be able to use this command if the permissions on these directories or files is changed appropriately.

Related topics

- **Troubleshooting** on page 113

11.5 Using GDB to locate job failures

If either the pbs_mom or pbs_server fail unexpectedly (and the log files contain no information on the failure) gdb can be used to determine whether or not the program is crashing. To start pbs_mom or pbs_server under GDB export the environment variable `PBSDEBUG=yes` and start the program (i.e., `gdb pbs_mom` and then issue the `run` subcommand at the gdb prompt).

GDB may run for some time until a failure occurs and at which point, a message will be printed to the screen and a gdb prompt again made available. If this occurs, use the gdb `where` subcommand to determine the exact location in the code. The information provided may be adequate to allow local diagnosis and correction. If not, this output may be sent to the mailing list or to help for further assistance.

See the PBSCOREDUMP parameter for enabling creation of core files (see **Debugging** on page 126).

Related topics

- **Troubleshooting** on page 113

11.6 Other diagnostic options

When **PBSDEBUG** is set, some client commands will print additional diagnostic information.

```
$ export PBSDEBUG=yes
$ cmd
```

To debug different kinds of problems, it can be useful to see where in the code time is being spent. This is called profiling and there is a Linux utility "gprof" that will output a listing of routines and the amount of time spent in these routines. This does require that the code be compiled with special options to instrument the code and to produce a file, gmon.out, that will be written at the end of program execution.

The following listing shows how to build TORQUE with profiling enabled. Notice that the output file for pbs_mom will end up in the `mom_priv` directory because its startup code changes the default directory to this location.

```
# ./configure "CFLAGS=-pg -lgcov -fPIC"
# make -j5
# make install
# pbs_mom ... do some stuff for a while ...
# momctl -s
# cd /var/spool/torque/mom_priv
# gprof -b `which pbs_mom` gmon.out |less
#
```

Another way to see areas where a program is spending most of its time is with the valgrind program. The advantage of using valgrind is that the programs do not have to be specially compiled.

```
# valgrind --tool=callgrind pbs_mom
```

Related topics

- **Troubleshooting** on page 113

11.7 Stuck jobs

If a job gets stuck in TORQUE, try these suggestions to resolve the issue:

- Use the **qdel** command to cancel the job.
- Force the MOM to send an obituary of the job ID to the server.

  ```
  > qsig -s 0 <JOBID>
  ```

- You can try clearing the stale jobs by using the **momctl** command on the compute nodes where the jobs are still listed.

  ```
  > momctl -c 58925 -h compute-5-20
  ```

- Setting the **qmgr** server setting `mom_job_sync` to **True** might help prevent jobs from hanging.

  ```
  > qmgr -c "set server mom_job_sync = True"
  ```

  To check and see if this is already set, use:

  ```
  > qmgr -c "p s"
  ```

- If the suggestions above cannot remove the stuck job, you can try **qdel** `-p`. However, since the `-p` option purges all information generated by the job, this is not a recommended option unless the above suggestions fail to remove the stuck job.

  ```
  > qdel -p <JOBID>
  ```

- The last suggestion for removing stuck jobs from compute nodes is to restart the pbs_mom.

For additional troubleshooting, run a tracejob on one of the stuck jobs. You can then create an online support ticket with the full server log for the time period displayed in the trace job.

Related topics

- **Troubleshooting** on page 113

11.8 Frequently asked questions (FAQ)

- **Cannot connect to server: error=15034** on page 119
- **Deleting 'stuck' jobs** on page 119
- **Which user must run TORQUE?** on page 120
- **Scheduler cannot run jobs - rc: 15003** on page 120
- **PBS_Server: pbsd_init, Unable to read server database** on page 120
- **qsub will not allow the submission of jobs requesting many processors** on page 121
- **qsub reports 'Bad UID for job execution'** on page 122
- **Why does my job keep bouncing from running to queued?** on page 122
- **How do I use PVM with TORQUE?** on page 122
- **My build fails attempting to use the TCL library** on page 123
- **My job will not start, failing with the message 'cannot send job to mom, state=PRERUN'** on page 123
- **I want to submit and run jobs as root** on page 123
- **How do I determine what version of TORQUE I am using?** on page 123
- **How do I resolve autogen.sh errors that contain "error: possibly undefined macro: AC_MSG_ERROR"?** on page 123
- **How do I resolve compile errors with libssl or libcrypto for TORQUE 4.0 on Ubuntu 10.04?** on page 124

Cannot connect to server: error=15034

This error occurs in TORQUE clients (or their APIs) because TORQUE cannot find the `server_name` file and/or the PBS_DEFAULT environment variable is not set. The `server_name` file or PBS_DEFAULT variable indicate the pbs_server's hostname that the client tools should communicate with. The `server_name` file is usually located in TORQUE's local state directory. Make sure the file exists, has proper permissions, and that the version of TORQUE you are running was built with the proper directory settings. Alternatively you can set the PBS_DEFAULT environment variable. Restart TORQUE daemons if you make changes to these settings.

Deleting 'stuck' jobs

To manually delete a "stale" job which has no process, and for which the mother superior is still alive, sending a sig 0 with qsig will often cause MOM to realize the job is stale and issue the proper JobObit notice. Failing that, use **momctl** `-c` to forcefully cause MOM to purge the job. The following process should never be necessary:

- Shut down the MOM on the mother superior node.
- Delete all files and directories related to the job from `TORQUE_HOME/mom_priv/jobs`.
- Restart the MOM on the mother superior node.

If the mother superior MOM has been lost and cannot be recovered (i.e, hardware or disk failure), a job running on that node can be purged from the output of **qstat** using the **qdel** on page 168 `-p` command or can be removed manually using the following steps:

To remove job X

1. Shutdown pbs_server.

```
> qterm
```

2. Remove job spool files.

```
> rm TORQUE_HOME/server_priv/jobs/X.SC TORQUE_HOME/server_priv/jobs/X.JB
```

3. Restart pbs_server

```
> pbs_server
```

Which user must run TORQUE?

TORQUE (pbs_server & pbs_mom) must be started by a user with root privileges.

Scheduler cannot run jobs - rc: 15003

For a scheduler, such as Moab or Maui, to control jobs with TORQUE, the scheduler needs to be run be a user in the server operators / managers list (see **qmgr**). The default for the server operators / managers list is root@localhost. For TORQUE to be used in a grid setting with Silver, the scheduler needs to be run as root.

PBS_Server: pbsd_init, Unable to read server database

If this message is displayed upon starting pbs_server it means that the local database cannot be read. This can be for several reasons. The most likely is a version mismatch. Most versions of TORQUE can read each others' databases. However, there are a few incompatibilities between OpenPBS and TORQUE. Because of enhancements to TORQUE, it cannot read the job database of an OpenPBS server (job structure sizes have been altered to increase functionality). Also, a compiled in 32-bit mode cannot read a database generated by a 64-bit pbs_server and vice versa.

To reconstruct a database (excluding the job database)

1. First, print out the old data with this command:

```
%> qmgr -c "p s"
#
# Create queues and set their attributes.
#
#
# Create and define queue batch
# create queue batch
set queue batch queue_type = Execution
set queue batch acl_host_enable = False
set queue batch resources_max.nodect = 6
set queue batch resources_default.nodes = 1
set queue batch resources_default.walltime = 01:00:00
set queue batch resources_available.nodect = 18
set queue batch enabled = True
set queue batch started = True
#
# Set server attributes.
#
set server scheduling = True
set server managers = griduser@oahu.icluster.org
set server managers += scott@*.icluster.org
set server managers += wightman@*.icluster.org
set server operators = griduser@oahu.icluster.org
set server operators += scott@*.icluster.org
set server operators += wightman@*.icluster.org
set server default_queue = batch
set server log_events = 511
set server mail_from = adm
set server resources_available.nodect = 80
set server scheduler_iteration = 600
set server node_ping_rate = 300
set server node_check_rate = 600
set server tcp_timeout = 6
```

2. Copy this information somewhere.
3. Restart pbs_server with the following command:

```
> pbs_server -t create
```

4. When you are prompted to overwrite the previous database, enter **y**, then enter the data exported by the qmgr command as in this example:

```
> cat data | qmgr
```

5. Restart pbs_server without the flags:

```
> qterm
> pbs_server
```

This will reinitialize the database to the current version.

Reinitializing the server database will reset the next jobid to 1

qsub will not allow the submission of jobs requesting many processors

TORQUE's definition of a node is context sensitive and can appear inconsistent. The **qsub** `-l nodes=<X>` expression can at times indicate a request for X processors and other time be interpreted as a request for

X nodes. While qsub allows multiple interpretations of the keyword nodes, aspects of the TORQUE server's logic are not so flexible. Consequently, if a job is using `-l nodes` to specify processor count and the requested number of processors exceeds the available number of physical nodes, the server daemon will reject the job.

To get around this issue, the server can be told it has an inflated number of nodes using the `resources_available` attribute. To take affect, this attribute should be set on both the server and the associated queue as in the example below. (See **resources_available** for more information.)

```
> qmgr
Qmgr: set server resources_available.nodect=2048
Qmgr: set queue batch resources_available.nodect=2048
```

The pbs_server daemon will need to be restarted before these changes will take affect.

qsub reports 'Bad UID for job execution'

```
[guest@login2]$ qsub test.job
qsub: Bad UID for job execution
```

Job submission hosts must be explicitly specified within TORQUE or enabled via RCmd security mechanisms in order to be trusted. In the example above, the host '`login2`' is not configured to be trusted. This process is documented in **Configuring job submission hosts** on page 19.

Why does my job keep bouncing from running to queued?

There are several reasons why a job will fail to start. Do you see any errors in the MOM logs? Be sure to increase the loglevel on MOM if you don't see anything. Also be sure TORQUE is configured with `--enable-syslog` and look in `/var/log/messages` (or wherever your syslog writes).

Also verify the following on all machines:

- DNS resolution works correctly with matching forward and reverse
- Time is synchronized across the head and compute nodes
- User accounts exist on all compute nodes
- User home directories can be mounted on all compute nodes
- Prologue scripts (if specified) exit with 0

If using a scheduler such as Moab or Maui, use a scheduler tool such as checkjob to identify job start issues.

How do I use PVM with TORQUE?

- Start the master pvmd on a compute node and then add the slaves
- mpiexec can be used to launch slaves using rsh or ssh (use export `PVM_RSH=/usr/bin/ssh` to use ssh)

Access can be managed by rsh/ssh without passwords between the batch nodes, but denying it from anywhere else, including the interactive nodes. This can be done with xinetd and sshd configuration (root is allowed to ssh everywhere). This way, the pvm daemons can be started and killed from the job script.

The problem is that this setup allows the users to bypass the batch system by writing a job script that uses rsh/ssh to launch processes on the batch nodes. If there are relatively few users and they can more or less be trusted, this setup can work.

My build fails attempting to use the TCL library

TORQUE builds can fail on TCL dependencies even if a version of TCL is available on the system. TCL is only utilized to support the xpbsmon client. If your site does not use this tool (most sites do not use xpbsmon), you can work around this failure by rerunning `configure` with the `--disable-gui` argument.

My job will not start, failing with the message 'cannot send job to mom, state=PRERUN'

If a node crashes or other major system failures occur, it is possible that a job may be stuck in a corrupt state on a compute node. TORQUE 2.2.0 and higher automatically handle this when the `mom_job_sync` parameter is set via **qmgr** (the default). For earlier versions of TORQUE, set this parameter and restart the pbs_mom daemon.

This error can also occur if not enough free space is available on the partition that holds TORQUE.

I want to submit and run jobs as root

While this can be a very bad idea from a security point of view, in some restricted environments this can be quite useful and can be enabled by setting the **acl_roots** parameter via **qmgr** command as in the following example:

```
qmgr

> qmgr -c 's s acl_roots+=root@*'
```

How do I determine what version of TORQUE I am using?

There are times when you want to find out what version of TORQUE you are using. An easy way to do this is to run the following command:

```
qmgr

> qmgr -c "p s" | grep pbs_ver
```

How do I resolve autogen.sh errors that contain "error: possibly undefined macro: AC_MSG_ERROR"?

Verify the `pkg-config` package is installed.

How do I resolve compile errors with libssl or libcrypto for TORQUE 4.0 on Ubuntu 10.04?

When compiling TORQUE 4.0 on Ubuntu 10.04 the following errors might occur:

```
libtool: link: gcc -Wall -pthread -g -D_LARGEFILE64_SOURCE -o .libs/trqauthd trq_auth_
daemon.o trq_main.o -ldl -lssl -lcrypto -L/home/adaptive/torques/torque-
4.0.0/src/lib/Libpbs/.libs /home/adaptive/torques/torque-
4.0.0/src/lib/Libpbs/.libs/libtorque.so -lpthread -lrt -pthread
/usr/bin/ld: cannot find -lssl
collect2: ld returned 1 exit status
make[3]: *** [trqauthd] Error 1

libtool: link: gcc -Wall -pthread -g -D_LARGEFILE64_SOURCE -o .libs/trqauthd trq_auth_
daemon.o trq_main.o -ldl -lssl -lcrypto -L/home/adaptive/torques/torque-
4.0.0/src/lib/Libpbs/.libs /home/adaptive/torques/torque-
4.0.0/src/lib/Libpbs/.libs/libtorque.so -lpthread -lrt -pthread
/usr/bin/ld: cannot find -lcrypto
collect2: ld returned 1 exit status
make[3]: *** [trqauthd] Error 1
```

To resolve the compile issue, use these commands:

```
> cd /usr/lib
> ln -s /lib/libcrypto.so.0.9. libcrypto.so
> ln -s /lib/libssl.so.0.9.8 libssl.so
```

Related topics

- **Troubleshooting** on page 113

11.9 Compute node health check

TORQUE provides the ability to perform health checks on each compute node. If these checks fail, a failure message can be associated with the node and routed to the scheduler. Schedulers (such as Moab) can forward this information to administrators by way of scheduler triggers, make it available through scheduler diagnostic commands, and automatically mark the node down until the issue is resolved. (See the RMMSGIGNORE parameter in the "Parameters" Appendix of the Moab Workload Manager Administrator's Guide for more information.)

For more information about node health checks, see these topics:

- **Configuring MOMs to launch a health check** on page 125
- **Creating the health check script** on page 125
- **Adjusting node state based on the health check output** on page 125
- **Example health check script** on page 126

Related topics

- **Troubleshooting** on page 113

11.9.1 Configuring MOMs to launch a health check

The health check feature is configured via the `mom_priv/config` file using the parameters described below:

Parameter	Format	Default	Description
$node_check_script	<STRING>	N/A	(Required) Specifies the fully qualified pathname of the health check script to run
$node_check_interval	<INTEGER>	1	(Optional) Specifies the number of MOM intervals between health checks (by default, each MOM interval is 45 seconds long - this is controlled via the **DEFAULT_SERVER_STAT_UPDATES** #define located in `TORQUE_HOME/src/resmom/mom_main.c`). The integer may be followed by a list of event names (currently supported are `jobstart` and `jobend`). (For more information, see **pbs_mom**.)

Related topics

- **Compute node health check** on page 124

11.9.2 Creating the health check script

The health check script is executed directly by the pbs_mom daemon under the root user id. It must be accessible from the compute node and may be a script or compile executable program. It may make any needed system calls and execute any combination of system utilities but should not execute resource manager client commands. Also, as of TORQUE 1.0.1, the pbs_mom daemon blocks until the health check is completed and does not possess a built-in timeout. Consequently, it is advisable to keep the launch script execution time short and verify that the script will not block even under failure conditions.

If the script detects a failure, it should return the keyword **ERROR** to stdout followed by an error message. When a failure is detected, the ERROR keyword should be printed to stdout before any other data. The message (up to 1024 characters) immediately following the ERROR keyword must all be contained on the same line. The message is assigned to the node attribute 'message' of the associated node.

Related topics

- **Compute node health check** on page 124

11.9.3 Adjusting node state based on the health check output

If the health check reports an error, the node attribute "message" is set to the error string returned. Cluster schedulers can be configured to adjust a given node's state based on this information. For example,

by default, Moab sets a node's state to down if a node error message is detected and restores the state as soon as the failure disappears.

Related topics

- **Compute node health check** on page 124

11.9.4 Example health check script

As mentioned, the health check can be a shell script, PERL, Python, C-executable, or anything which can be executed from the command line capable of setting STDOUT. The example below demonstrates a very simple health check:

```
#!/bin/sh
/bin/mount | grep global
if [ $? != "0" ]
  then
    echo "ERROR cannot locate filesystem global"
fi
```

Related topics

- **Compute node health check** on page 124

11.10 Debugging

TORQUE supports a number of diagnostic and debug options including the following:

PBSDEBUG environment variable - If set to 'yes', this variable will prevent pbs_server, pbs_mom, and/or pbs_sched from backgrounding themselves allowing direct launch under a debugger. Also, some client commands will provide additional diagnostic information when this value is set.

PBSLOGLEVEL environment variable - Can be set to any value between 0 and 7 and specifies the logging verbosity level (default = 0)

PBSCOREDUMP environment variable - If set, it will cause the offending resource manager daemon to create a core file if a **SIGSEGV**, **SIGILL**, **SIGFPE**, **SIGSYS**, or **SIGTRAP** signal is received. The core dump will be placed in the daemon's home directory (`$PBSHOME/mom_priv` for pbs_mom).

To enable core dumping in a Red Hat system, you must add the following line to the `/etc/init.d/pbs_mom` script:

```
export DAEMON_COREFILE_LIMIT=unlimited
```

NDEBUG #define - if set at build time, will cause additional low-level logging information to be output to stdout for pbs_server and pbs_mom daemons.

tracejob reporting tool - can be used to collect and report logging and accounting information for specific jobs (for more information, see **Using "tracejob" to locate job failures** on page 114)

PBSLOGLEVEL and **PBSCOREDUMP** must be added to the `$PBSHOME/pbs_environment` file, not just the current environment. To set these variables, add a line to the pbs_environment file as either "variable=value" or just "variable". In the case of "variable=value", the environment variable is set up as the value specified. In the case of "variable", the environment variable is set based upon its value in the current environment.

TORQUE error codes

Error code name	Number	Description
PBSE_FLOOR	15000	No error
PBSE_UNKJOBID	15001	Unknown job identifier
PBSE_NOATTR	15002	Undefined attribute
PBSE_ATTRRO	15003	Attempt to set READ ONLY attribute
PBSE_IVALREQ	15004	Invalid request
PBSE_UNKREQ	15005	Unknown batch request
PBSE_TOOMANY	15006	Too many submit retries
PBSE_PERM	15007	No permission
PBSE_IFF_NOT_FOUND	15008	"pbs_iff" not found; unable to authenticate
PBSE_MUNGE_NOT_FOUND	15009	"munge" executable not found; unable to authenticate
PBSE_BADHOST	15010	Access from host not allowed
PBSE_JOBEXIST	15011	Job already exists
PBSE_SYSTEM	15012	System error occurred
PBSE_INTERNAL	15013	Internal server error occurred
PBSE_REGROUTE	15014	Parent job of dependent in rte queue
PBSE_UNKSIG	15015	Unknown signal name

Error code name	Number	Description
PBSE_BADATVAL	15016	Bad attribute value
PBSE_MODATRRUN	15017	Cannot modify attribute in run state
PBSE_BADSTATE	15018	Request invalid for job state
PBSE_UNKQUE	15020	Unknown queue name
PBSE_BADCRED	15021	Invalid credential in request
PBSE_EXPIRED	15022	Expired credential in request
PBSE_QUNOENB	15023	Queue not enabled
PBSE_QACESS	15024	No access permission for queue
PBSE_BADUSER	15025	Bad user - no password entry
PBSE_HOPCOUNT	15026	Max hop count exceeded
PBSE_QUEEXIST	15027	Queue already exists
PBSE_ATTRTYPE	15028	Incompatible queue attribute type
PBSE_QUEBUSY	15029	Queue busy (not empty)
PBSE_QUENBIG	15030	Queue name too long
PBSE_NOSUP	15031	Feature/function not supported
PBSE_QUENOEN	15032	Cannot enable queue,needs add def
PBSE_PROTOCOL	15033	Protocol (ASN.1) error
PBSE_BADATLST	15034	Bad attribute list structure
PBSE_NOCONNECTS	15035	No free connections
PBSE_NOSERVER	15036	No server to connect to

Error code name	Number	Description
PBSE_UNKRESC	15037	Unknown resource
PBSE_EXCQRESC	15038	Job exceeds queue resource limits
PBSE_QUENODFLT	15039	No default queue defined
PBSE_NORERUN	15040	Job not rerunnable
PBSE_ROUTEREJ	15041	Route rejected by all destinations
PBSE_ROUTEEXPD	15042	Time in route queue expired
PBSE_MOMREJECT	15043	Request to MOM failed
PBSE_BADSCRIPT	15044	(qsub) Cannot access script file
PBSE_STAGEIN	15045	Stage-In of files failed
PBSE_RESCUNAV	15046	Resources temporarily unavailable
PBSE_BADGRP	15047	Bad group specified
PBSE_MAXQUED	15048	Max number of jobs in queue
PBSE_CKPBSY	15049	Checkpoint busy, may be retries
PBSE_EXLIMIT	15050	Limit exceeds allowable
PBSE_BADACCT	15051	Bad account attribute value
PBSE_ALRDYEXIT	15052	Job already in exit state
PBSE_NOCOPYFILE	15053	Job files not copied
PBSE_CLEANEDOUT	15054	Unknown job id after clean init
PBSE_NOSYNCMSTR	15055	No master in sync set
PBSE_BADDEPEND	15056	Invalid dependency

Error code name	Number	Description
PBSE_DUPLIST	15057	Duplicate entry in list
PBSE_DISPROTO	15058	Bad DIS based request protocol
PBSE_EXECTHERE	15059	Cannot execute there
PBSE_SISREJECT	15060	Sister rejected
PBSE_SISCOMM	15061	Sister could not communicate
PBSE_SVRDOWN	15062	Requirement rejected -server shutting down
PBSE_CKPSHORT	15063	Not all tasks could checkpoint
PBSE_UNKNODE	15064	Named node is not in the list
PBSE_UNKNODEATR	15065	Node-attribute not recognized
PBSE_NONODES	15066	Server has no node list
PBSE_NODENBIG	15067	Node name is too big
PBSE_NODEEXIST	15068	Node name already exists
PBSE_BADNDATVAL	15069	Bad node-attribute value
PBSE_MUTUALEX	15070	State values are mutually exclusive
PBSE_GMODERR	15071	Error(s) during global modification of nodes
PBSE_NORELYMOM	15072	Could not contact Mom
PBSE_NOTSNODE	15073	No time-shared nodes
PBSE_JOBTYPE	15074	Wrong job type
PBSE_BADACLHOST	15075	Bad ACL entry in host list
PBSE_MAXUSERQUED	15076	Maximum number of jobs already in queue for user

Error code name	Number	Description
PBSE_BADDISALLOWTYPE	15077	Bad type in "disallowed_types" list
PBSE_NOINTERACTIVE	15078	Interactive jobs not allowed in queue
PBSE_NOBATCH	15079	Batch jobs not allowed in queue
PBSE_NORERUNABLE	15080	Rerunable jobs not allowed in queue
PBSE_NONONRERUNABLE	15081	Non-rerunable jobs not allowed in queue
PBSE_UNKARRAYID	15082	Unknown array ID
PBSE_BAD_ARRAY_REQ	15083	Bad job array request
PBSE_TIMEOUT	15084	Time out
PBSE_JOBNOTFOUND	15085	Job not found
PBSE_NOFAULTTOLERANT	15086	Fault tolerant jobs not allowed in queue
PBSE_NOFAULTINTOLERANT	15087	Only fault tolerant jobs allowed in queue
PBSE_NOJOBARRAYS	15088	Job arrays not allowed in queue
PBSE_RELAYED_TO_MOM	15089	Request was relayed to a MOM
PBSE_MEM_MALLOC	15090	Failed to allocate memory for memmgr
PBSE_MUTEX	15091	Failed to allocate controlling mutex (lock/unlock)
PBSE_TRHEADATTR	15092	Failed to set thread attributes
PBSE_THREAD	15093	Failed to create thread
PBSE_SELECT	15094	Failed to select socket
PBSE_SOCKET_FAULT	15095	Failed to get connection to socket
PBSE_SOCKET_WRITE	15096	Failed to write data to socket

Error code name	Number	Description
PBSE_SOCKET_READ	15097	Failed to read data from socket
PBSE_SOCKET_CLOSE	15098	Socket closed
PBSE_SOCKET_LISTEN	15099	Failed to listen in on socket
PBSE_AUTH_INVALID	15100	Invalid auth type in request
PBSE_NOT_IMPLEMENTED	15101	Functionality not yet implemented
PBSE_QUENOTAVAILABLE	15102	Queue is not available

Related topics

- **Troubleshooting** on page 113

Appendices

The appendices provide tables of commands, parameters, configuration options, error codes, the Quick Start Guide, and so forth.

- A.1 Commands overview on page 135
- B.1 Server parameters on page 209
- C.1 Node manager (MOM) configuration on page 225
- D.1 Diagnostics and error codes on page 243
- E.1 Considerations before upgrading on page 249
- F.1 Large cluster considerations on page 251
- G.1 Prologue and epilogue scripts on page 257
- H.1 Running multiple TORQUE servers and MOMs on the same node on page 265
- I.1 Security overview on page 267
- J.1 Job submission filter ("qsub wrapper") on page 269
- K.1 "torque.cfg" configuration file on page 271
- L.1 TORQUE Quick Start Guide on page 275
- M.1 BLCR acceptance tests on page 279

A.1 Commands overview

Client commands

Command	Description
momctl	Manage/diagnose MOM (node execution) daemon
pbsdsh	Launch tasks within a parallel job
pbsnodes	View/modify batch status of compute nodes
qalter	Modify queued batch jobs
qchkpt	Checkpoint batch jobs
qdel	Delete/cancel batch jobs
qgpumode	Specifies new mode for GPU
qgpureset	Reset the GPU
qhold	Hold batch jobs
qmgr	Manage policies and other batch configuration
qrerun	Rerun a batch job
qrls	Release batch job holds
qrun	Start a batch job
qsig	Send a signal to a batch job

Command	Description
qstat	View queues and jobs
qsub	Submit jobs
qterm	Shutdown pbs server daemon
tracejob	Trace job actions and states recorded in TORQUE logs (see Using "tracejob" to locate job failures on page 114)

Binary executables

Command	Description
pbs_iff	Interprocess authentication service
pbs_mom	Start MOM (node execution) daemon
pbs_server	Start server daemon
pbs_track	Tell pbs_mom to track a new process

Related topics

- **Node manager (MOM) configuration** on page 225
- **Server parameters** on page 209

momctl

(PBS MOM Control)

Synopsis

```
momctl -c { <JOBID> | all }
momctl -C
momctl -d { <INTEGER> | <JOBID> }
momctl -f <FILE>
momctl -h <HOST>[,<HOST>]...
momctl -p <PORT_NUMBER>
momctl -q <ATTRIBUTE>
momctl -r { <FILE> | LOCAL:<FILE> }
momctl -s
```

Overview

The momctl command allows remote shutdown, reconfiguration, diagnostics, and querying of the pbs_mom daemon.

Format

-c — Clear	
Format	{ <JOBID> | **all** }
Default	---
Description	Clear stale job information
Example	`momctl - node1 -c 15406`

-C — Cycle	
Format	---
Default	---
Description	Cycle pbs_mom(s)
Example	`momctl - node1 -C` Cycle pbs_mom on node1.

-d — Diagnose	
Format	{ <INTEGER> | <JOBID> }
Default	0
Description	Diagnose mom(s) (For more details, see **Diagnose detail** on page 140 below.)
Example	`momctl - node1 -d 2` Print level 2 and lower diagnose information for the MOM on node1.

-f — Host File	
Format	<FILE>
Default	---
Description	A file containing only comma or whitespace (space, tab, or new line) delimited hostnames
Example	`momctl -f hosts.txt -d` Print diagnose information for the MOMs running on the hosts specified in `hosts.txt`.

-h — Host List	
Format	<HOST>[,<HOST>]...
Default	localhost
Description	A comma separated list of hosts
Example	`momctl -h node1,node2,node3 -d` Print diagnose information for the MOMs running on node1, node2, and node3.

-p — Port	
Format	<PORT_NUMBER>
Default	TORQUE's default port number
Description	The port number for the specified MOM(s)
Example	`momctl -p 5455 -h node1 -d` Request diagnose information over port 5455 on node1.

-q — Query	
Format	<ATTRIBUTE>
Default	---

-q — Query	
Description	Query <ATTRIBUTE> on specified MOM, where *<ATTRIBUTE>* is a property listed by **pbsnodes** `-a` (see **Query attributes** on page 139 for a list of attributes)
Example	`momctl -q physmem` Print the amount of physmem on localhost.

-r — Reconfigure	
Format	{ <FILE> | LOCAL:<FILE> }
Default	---
Description	Reconfigure MOM(s) with remote or local config file, <FILE>. This does not work if $remote_reconfig is not set to true when the MOM is started.
Example	`momctl -r /home/user1/new.config -h node1` Reconfigure MOM on node1 with `/home/user1/new.cofig` on node1.

-s — Shutdown	
Format	
Default	---
Description	Shutdown pbs_mom
Example	`momctl -s` Terminates pbs_mom process on localhost.

Query attributes

Attribute	Description
arch	node hardware architecture
availmem	available RAM

Attribute	Description
loadave	1 minute load average
ncpus	number of CPUs available on the system
netload	total number of bytes transferred over all network interfaces
nsessions	number of sessions active
nusers	number of users active
physmem	configured RAM
sessions	list of active sessions
totmem	configured RAM plus configured swap

Diagnose detail

Level	Description
0	Display the following information: • Local hostname • Expected server hostname • Execution version • MOM home directory • MOM config file version (if specified) • Duration MOM has been executing • Duration since last request from pbs_server daemon • Duration since last request to pbs_server daemon • RM failure messages (if any) • Log verbosity level • Local job list

Level	Description
1	All information for level 0 plus the following: • Interval between updates sent to server • Number of initialization messages sent to pbs_server daemon • Number of initialization messages received from pbs_server daemon • Prolog/epilog alarm time • List of trusted clients
2	All information from level 1 plus the following: • PID • Event alarm status
3	All information from level 2 plus the following: • syslog enabled

Example A-1: MOM diagnostics

```
momctl -d 1

Host: nsrc/nsrc.fllcl.com    Server: 10.10.10.113    Version: torque_1.1.0p4
HomeDirectory:          /usr/spool/PBS/mom_priv
ConfigVersion:          147
MOM active:             7390 seconds
Last Msg From Server:   7389 seconds (CLUSTER_ADDRS)
Server Update Interval: 20 seconds
Server Update Interval: 20 seconds
Init Msgs Received:     0 hellos/1 cluster-addrs
Init Msgs Sent:         1 hellos
LOGLEVEL:               0 (use SIGUSR1/SIGUSR2 to adjust)
Prolog Alarm Time:      300 seconds
Trusted Client List:    12.14.213.113,127.0.0.1
JobList:                NONE

diagnostics complete
```

Example A-2: System shutdown

```
> momctl -s -f /opt/clusterhostfile

shutdown request successful on node001
shutdown request successful on node002
shutdown request successful on node003
shutdown request successful on node004
shutdown request successful on node005
shutdown request successful on node006
```

pbs_mom

Start a pbs batch execution mini-server.

Synopsis

```
pbs_mom [-a alarm] [-A alias] [-C chkdirectory] [-c config] [-d directory] [-h
hostname]
        [-L logfile] [-M MOMport] [-R RPPport] [-p|-r] [-P purge] [-w] [-x]
```

Description

The pbs_mom command is located within the TORQUE_HOME directory and starts the operation of a batch Machine Oriented Mini-server (MOM) on the execution host. To insure that the pbs_mom command is not runnable by the general user community, the server will only execute if its real and effective uid is zero.

The first function of pbs_mom is to place jobs into execution as directed by the server, establish resource usage limits, monitor the job's usage, and notify the server when the job completes. If they exist, pbs_mom will execute a prologue script before executing a job and an epilogue script after executing the job.

The second function of pbs_mom is to respond to resource monitor requests. This was done by a separate process in previous versions of PBS but has now been combined into one process. It provides information about the status of running jobs, memory available etc.

The last function of pbs_mom is to respond to task manager requests. This involves communicating with running tasks over a TCP socket as well as communicating with other MOMs within a job (a.k.a. a "sisterhood").

pbs_mom will record a diagnostic message in a log file for any error occurrence. The log files are maintained in the mom_logs directory below the home directory of the server. If the log file cannot be opened, the diagnostic message is written to the system console.

Options

Flag	Name	Description
-a	alarm	Used to specify the alarm timeout in seconds for computing a resource. Every time a resource request is processed, an alarm is set for the given amount of time. If the request has not completed before the given time, an alarm signal is generated. The default is 5 seconds.
-A	alias	Used to specify this multimom's alias name. The alias name needs to be the same name used in the mom.hierarchy file. It is only needed when running multiple MOMs on the same machine. For more information, see **TORQUE Multi-MOM** on page 32.
-C	chkdirectory	Specifies The path of the directory used to hold checkpoint files. (Currently this is only valid on Cray systems.) The default directory is TORQUE_HOME/spool/checkpoint (see the **-d** option). The directory specified with the -C option must be owned by root and accessible (rwx) only by root to protect the security of the checkpoint files.

Flag	Name	Description
-c	config	Specifies an alternative configuration file, see description below. If this is a relative file name it will be relative to `TORQUE_HOME/mom_priv`, (see the **-d** option). If the specified file cannot be opened, pbs_mom will abort. If the **-C** option is not supplied, pbs_mom will attempt to open the default configuration file "config" in `TORQUE_HOME/mom_priv`. If this file is not present, pbs_mom will log the fact and continue.
-d	directory	Specifies the path of the directory which is the home of the server's working files, `TORQUE_HOME`. This option is typically used along with **-M** when debugging MOM. The default directory is given by `$PBS_SERVER_HOME` which is typically `/usr/spool/PBS`.
-h	hostname	Set MOM's hostname. This can be useful on multi-homed networks.
-L	logfile	Specify an absolute path name for use as the log file. If not specified, MOM will open a file named for the current date in the `TORQUE_HOME/mom_logs` directory (see the **-d** option).
-M	port	Specifies the port number on which the mini-server (MOM) will listen for batch requests.
-p	n/a	Specifies the impact on jobs which were in execution when the mini-server shut down. On any restart of MOM, the new mini-server will not be the parent of any running jobs, MOM has lost control of her offspring (not a new situation for a mother). With the -p option, MOM will allow the jobs to continue to run and monitor them indirectly via polling. This flag is redundant in that this is the default behavior when starting the server. The -p option is mutually exclusive with the **-R** and **-q** options.
-P	purge	Specifies the impact on jobs which were in execution when the mini-server shut down. With the -P option, it is assumed that either the entire system has been restarted or the MOM has been down so long that it can no longer guarantee that the pid of any running process is the same as the recorded job process pid of a recovering job. Unlike the **-p** option, no attempt is made to try and preserve or recover running jobs. All jobs are terminated and removed from the queue.
-q	n/a	Specifies the impact on jobs which were in execution when the mini-server shut down. With the -q option, MOM will allow the processes belonging to jobs to continue to run, but will not attempt to monitor them. The -q option is mutually exclusive with the **-p** and **-R** options.
-R	port	Specifies the port number on which the mini-server (MOM) will listen for resource monitor requests, task manager requests and inter-MOM messages. Both a UDP and a TCP port of this number will be used.

Flag	Name	Description
-r	n/a	Specifies the impact on jobs which were in execution when the mini-server shut down. With the -r option, MOM will kill any processes belonging to jobs, mark the jobs as terminated, and notify the batch server which owns the job. The -r option is mutually exclusive with the **-p** and **-q** options. Normally the mini-server is started from the system boot file without the **-p** or the -r option. The mini-server will make no attempt to signal the former session of any job which may have been running when the mini-server terminated. It is assumed that on reboot, all processes have been killed. If the -r option is used following a reboot, process IDs (pids) may be reused and MOM may kill a process that is not a batch session.
-w	wait_for_ server	When started with -w, pbs_moms wait until they get their MOM hierarchy file from pbs_ server to send their first update, or until 10 minutes pass. This reduces network traffic on startup and can bring up clusters faster.
-x	n/a	Disables the check for privileged port resource monitor connections. This is used mainly for testing since the privileged port is the only mechanism used to prevent any ordinary user from connecting.

Configuration file

The configuration file may be specified on the command line at program start with the **-C** flag. The use of this file is to provide several types of run time information to pbs_mom: static resource names and values, external resources provided by a program to be run on request via a shell escape, and values to pass to internal set up functions at initialization (and re-initialization).

Each item type is on a single line with the component parts separated by white space. If the line starts with a hash mark (pound sign, #), the line is considered to be a comment and is skipped.

Static Resources

For static resource names and values, the configuration file contains a list of resource names/values pairs, one pair per line and separated by white space. An example of static resource names and values could be the number of tape drives of different types and could be specified by:

- tape3480 4
- tape3420 2
- tapedat 1
- tape8mm 1

Shell Commands

If the first character of the value is an exclamation mark (**!**), the entire rest of the line is saved to be executed through the services of the system(3) standard library routine.

The shell escape provides a means for the resource monitor to yield arbitrary information to the scheduler. Parameter substitution is done such that the value of any qualifier sent with the query, as

explained below, replaces a token with a percent sign (**%**) followed by the name of the qualifier. For example, here is a configuration file line which gives a resource name of "escape":

```
escape !echo %xxx %yyy
```

If a query for "escape" is sent with no qualifiers, the command executed would be `echo %xxx %yyy`.

If one qualifier is sent, `escape[xxx=hi there]`, the command executed would be `echo hi there %yyy`.

If two qualifiers are sent, `escape[xxx=hi][yyy=there]`, the command executed would be `echo hi there`.

If a qualifier is sent with no matching token in the command line, `escape[zzz=snafu]`, an error is reported.

size[fs=<FS>]

Specifies that the available and configured disk space in the <FS> filesystem is to be reported to the pbs_server and scheduler. To request disk space on a per job basis, specify the file resource, as in `qsub -l nodes=1,file=1000kb`. For example, the available and configured disk space in the /localscratch filesystem will be reported:

```
size[fs=/localscratch]
```

Initialization Value

An initialization value directive has a name which starts with a dollar sign (**$**) and must be known to the MOM via an internal table. The entries in this table now are:

Entry	Description

pbsclient	Causes a host name to be added to the list of hosts which will be allowed to connect to theMOM as long as they are using a privilaged port for the purposes of resource monitor requests. For example, here are two configuration file lines which will allow the hosts "fred" and "wilma" to connect:
	`$pbsclient fred`
	`$pbsclient wilma`
	Two host names are always allowed to connect to pbs_mom "localhost" and the name returned to pbs_mom by the system call gethostname(). These names need not be specified in the configuration file. The hosts listed as "clients" can issue Resource Manager (RM) requests. Other MOM nodes and servers do not need to be listed as clients.

Entry	Description
restricted	Causes a host name to be added to the list of hosts which will be allowed to connect to the MOM without needing to use a privilaged port. These names allow for wildcard matching. For example, here is a configuration file line which will allow queries from any host from the domain "ibm.com". `$restricted *.ibm.com` The restriction which applies to these connections is that only internal queries may be made. No resources from a config file will be found. This is to prevent any shell commands from being run by a non-root process. This parameter is generally not required except for some versions of OSX.
logevent	Sets the mask that determines which event types are logged by pbs_mom. For example: `$logevent 0x1fff $logevent 255` The first example would set the log event mask to 0x1ff (511) which enables logging of all events including debug events. The second example would set the mask to 0x0ff (255) which enables all events except debug events.
cputmult	Sets a factor used to adjust cpu time used by a job. This is provided to allow adjustment of time charged and limits enforced where the job might run on systems with different cpu performance. If the MOM's system is faster than the reference system, set cputmult to a decimal value greater than 1.0. If the MOM's system is slower, set cputmult to a value between 1.0 and 0.0. For example: `$cputmult 1.5 $cputmult 0.75`
usecp	Specifies which directories should be staged with cp instead of rcp/scp. If a shared filesystem is available on all hosts in a cluster, this directive is used to make these filesystems known to the MOM. For example, if `/home` is NFS mounted on all nodes in a cluster: `$usecp *:/home /home`
wallmult	Sets a factor to adjust wall time usage by to job to a common reference system. The factor is used for walltime calculations and limits in the same way that cputmult is used for cpu time.
configversion	Specifies the version of the config file data, a string.
check_poll_time	Specifies the MOM interval in seconds. The MOM checks each job for updated resource usages, exited processes, over-limit conditions, etc., once per interval. This value should be equal or lower to pbs_server's `job_stat_rate`. High values result in stale information reported to pbs_server. Low values result in increased system usage by the MOM. Default is 45 seconds.
down_on_error	Causes the MOM to report itself as state "down" to pbs_server in the event of a failed health check. This feature is experimental. (For more information, see **Health check** on page 149.)

Entry	Description
ideal_load	Ideal processor load. Represents a low water mark for the load average. A node that is currently busy will consider itself free after falling below ideal_load.
loglevel	Specifies the verbosity of logging with higher numbers specifying more verbose logging. Values may range between 0 and 7.
log_file_max_size	If this is set to a value > 0, then pbs_mom will roll the current log file to log-file-name.1 when its size is greater than or equal to the value of log_file_max_size. This value is interpreted as kilobytes.
log_file_roll_depth	If this is set to a value >=1 and log_file_max_size is set, then pbs_mom will allow logs to be rolled up to the specified number of logs. At every roll, the oldest log will be the one to be deleted to make room for rolling. pbs_mom will continue rolling the log files to `log-file-name.log_file_roll_depth`.
max_load	Maximum processor load. Nodes over this load average are considered busy (see **ideal_load** above).
enablemomrestart	Enables automatic restarts of the MOM. If enabled, the MOM will check if its binary has been updated and restart itself at a safe point when no jobs are running; thus making upgrades easier. The check is made by comparing the mtime of the pbs_mom executable. Command-line args, the process name, and the PATH env variable are preserved across restarts. It is recommended that this not be enabled in the config file, but enabled when desired with momctl (see **Resources** on page 149 for more information.)
node_check_script	Specifies the fully qualified pathname of the health check script to run (see **Health check** on page 149 for more information).

node_check_interval	Specifies when to run the MOM health check. The check can be either periodic, event-driven, or both. The value starts with an integer specifying the number of MOM intervals between subsequent executions of the specified health check. After the integer is an optional comma-separated list of event names. Currently supported are "jobstart" and "jobend". This value defaults to 1 with no events indicating the check is run every MOM interval. (see **Health check** on page 149 for more information.)
	`$node_check_interval 0,Disabled`
	`$node_check_interval 0,jobstartOnly`
	`$node_check_interval 10,jobstart,jobend`

prologalarm	Specifies maximum duration (in seconds) which the MOM will wait for the job prolog or job job epilog to complete. This parameter defaults to 300 seconds (5 minutes).

Entry	Description

rcpcmd	Specify the full path and argument to be used for remote file copies. This overrides the compile-time default found in configure. This must contain 2 words: the full path to the command and the options. The copy command must be able to recursively copy files to the remote host and accept arguments of the form "user@host:files." For example:
	`$rcpcmd /usr/bin/rcp -rp`
	`$rcpcmd /usr/bin/scp -rpB`

remote_reconfig	Enables the ability to remotely reconfigure pbs_mom with a new config file. Default is disabled. This parameter accepts various forms of true, yes, and 1.
timeout	Specifies the number of seconds before TCP messages will time out. TCP messages include job obituaries, and TM requests if RPP is disabled. Default is 60 seconds.
tmpdir	Sets the directory basename for a per-job temporary directory. Before job launch, the MOM will append the jobid to the tmpdir basename and create the directory. After the job exit, the MOM will recursively delete it. The env variable TMPDIR will be set for all prolog/epilog scripts, the job script, and TM tasks. Directory creation and removal is done as the job owner and group, so the owner must have write permission to create the directory. If the directory already exists and is owned by the job owner, it will not be deleted after the job. If the directory already exists and is NOT owned by the job owner, the job start will be rejected.
status_update_time	Specifies (in seconds) how often the MOM updates its status information to pbs_server. This value should correlate with the server's scheduling interval and its "node_check_rate" attribute. High values for "status_update_time" cause pbs_server to report stale information, while low values increase the load of pbs_server and the network. Default is 45 seconds.

varattr	This is similar to a shell escape above, but includes a TTL. The command will only be run every TTL seconds. A TTL of -1 will cause the command to be executed only once. A TTL of 0 will cause the command to be run every time varattr is requested. This parameter may be used multiple times, but all output will be grouped into a single "varattr" attribute in the request and status output. If the command has no output, the name will be skipped in the output.
	`$varattrseta`
	`$varattrsetb`

xauthpath	Specifies the path to the xauth binary to enable X11 fowarding.
ignvmem	If set to true, then pbs_mom will ignore vmem/pvmem limit enforcement.
ignwalltime	If set to true, then pbs_mom will ignore walltime limit enforcement.
mom_host	Sets the local hostname as used by pbs_mom.

Resources

Resource Manager queries can be made with **momctl** `-q` options to retrieve and set pbs_mom options. Any configured static resource may be retrieved with a request of the same name. These are resource requests not otherwise documented in the PBS ERS.

Request	Description
cycle	Forces an immediate MOM cycle.
status_update_time	Retrieve or set the $status_update_time parameter.
check_poll_time	Retrieve or set the $check_poll_time parameter.
configversion	Retrieve the config version.
jobstartblocktime	Retrieve or set the $jobstartblocktime parameter.
enablemomrestart	Retrieve or set the $enablemomrestart parameter.
loglevel	Retrieve or set the $loglevel parameter.
down_on_error	Retrieve or set the EXPERIMENTAL $down_on_error parameter.
diag0 - diag4	Retrieves various diagnostic information.
rcpcmd	Retrieve or set the $rcpcmd parameter.
version	Retrieves the pbs_mom version.

Health check

The health check script is executed directly by the pbs_mom daemon under the root user id. It must be accessible from the compute node and may be a script or compiled executable program. It may make any needed system calls and execute any combination of system utilities but should not execute resource manager client commands. Also, as of TORQUE 1.0.1, the pbs_mom daemon blocks until the health check is completed and does not possess a built-in timeout. Consequently, it is advisable to keep the launch script execution time short and verify that the script will not block even under failure conditions.

If the script detects a failure, it should return the keyword **Error** to `stdout` followed by an error message. The message (up to 256 characters) immediately following the Error string will be assigned to the node attribute message of the associated node.

If the script detects a failure when run from "jobstart", then the job will be rejected. This should probably only be used with advanced schedulers like Moab so that the job can be routed to another node.

TORQUE currently ignores Error messages by default, but advanced schedulers like Moab can be configured to react appropriately.

If the experimental `$down_on_error` MOM setting is enabled, the MOM will set itself to state down and report to pbs_server, and pbs_server will report the node as "down". Additionally, the experimental "down_on_error" server attribute can be enabled which has the same effect but moves the decision to pbs_server. It is redundant to have MOM's `$down_on_error` and pbs_servers `down_on_error` features enabled. See "down_on_error" in pbs_server_attributes(7B).

Files

File	Description
$PBS_SERVER_HOME/server_name	Contains the hostname running pbs_server
$PBS_SERVER_HOME/mom_priv	The default directory for configuration files, typically (`/usr/spool/pbs`) `/mom_priv`
$PBS_SERVER_HOME/mom_logs	Directory for log files recorded by the server
$PBS_SERVER_HOME/mom_priv/prologue	The administrative script to be run before job execution
$PBS_SERVER_HOME/mom_priv/epilogue	The administrative script to be run after job execution

Signal handling

pbs_mom handles the following signals:

Signal	Description
SIGHUP	Causes pbs_mom to re-read its configuration file, close and reopen the log file, and reinitialize resource structures.
SIGALRM	Results in a log file entry. The signal is used to limit the time taken by certain children processes, such as the prologue and epilogue.
SIGINT and SIGTERM	Results in pbs_mom exiting without terminating any running jobs. This is the action for the following signals as well: SIGXCPU, SIGXFSZ, SIGCPULIM, and SIGSHUTDN.
SIGUSR1, SIGUSR2	Causes the MOM to increase and decrease logging levels, respectively.
SIGPIPE, SIGINFO	Are ignored.

Signal	Description
SIGBUS, SIGFPE, SIGILL, SIGTRAP, and SIGSYS	Cause a core dump if the PBSCOREDUMP environmental variable is defined.

All other signals have their default behavior installed.

Exit status

If the pbs_mom command fails to begin operation, the server exits with a value greater than zero.

Related topics

- pbs_server(8B)

Non-Adaptive Computing topics

- pbs_scheduler_basl(8B)
- pbs_scheduler_tcl(8B)
- PBS External Reference Specification
- PBS Administrators Guide

pbs_server

(**PBS Server**) pbs batch system manager

Synopsis

```
pbs_server [-a active] [-c] [-d config_path] [-p port] [-A acctfile]
           [-L logfile] [-S scheduler_port]
           [-h hostname] [-t type] [--ha]
```

Description

The pbs_server command starts the operation of a batch server on the local host. Typically, this command will be in a local boot file such as `/etc/rc.local`. If the batch server is already in execution, pbs_server will exit with an error. To ensure that the pbs_server command is not runnable by the general user community, the server will only execute if its real and effective uid is zero.

The server will record a diagnostic message in a log file for any error occurrence. The log files are maintained in the server_logs directory below the home directory of the server. If the log file cannot be opened, the diagnostic message is written to the system console.

As of TORQUE 4.0, the pbs_server is multi-threaded which leads to quicker response to client commands, is more robust, and allows for higher job throughput.

Options

Option	Name	Description
-A	acctfile	Specifies an absolute path name of the file to use as the accounting file. If not specified, the file name will be the current date in the `PBS_HOME/server_priv/accounting` directory.
-a	active	Specifies if scheduling is active or not. This sets the server attribute scheduling. If the option argument is "true" ("True", "t", "T", or "1"), the server is active and the PBS job scheduler will be called. If the argument is "false" ("False", "f", "F", or "0), the server is idle, and the scheduler will not be called and no jobs will be run. If this option is not specified, the server will retain the prior value of the scheduling attribute.
-c	wait_for_moms	This directs pbs_server to send the MOM hierarchy only to MOMs that request it for the first 10 minutes. After 10 minutes, it attempts to send the MOM hierarchy to MOMs that haven't requested it already. This greatly reduces traffic on start up.
-d	config_directory	Specifies the path of the directory which is home to the servers configuration files, PBS_HOME. A host may have multiple servers. Each server must have a different configuration directory. The default configuration directory is given by the symbol `$PBS_SERVER_HOME` which is typically `var/spool/torque`.
-h	hostname	Causes the server to start under a different hostname as obtained from gethostname(2). Useful for servers with multiple network interfaces to support connections from clients over an interface that has a hostname assigned that differs from the one that is returned by gethost name(2).
-ha	high_availability	Starts server in high availability mode (for details, see **Server high availability** on page 88).
-L	logfile	Specifies an absolute path name of the file to use as the log file. If not specified, the file will be the current date in the `PBS_HOME/server_logs` directory (see the **-d** option).
-p	port	Specifies the port number on which the server will listen for batch requests. If multiple servers are running on a single host, each must have its own unique port number. This option is for use in testing with multiple batch systems on a single host.
-S	scheduler_port	Specifies the port number to which the server should connect when contacting the scheduler. The argument scheduler_conn is of the same syntax as under the -M option.

Option	Name	Description
-t	type	Specifies the impact on jobs which were in execution, running, when the server shut down. If the running job is not rerunnable or restartable from a checkpoint image, the job is aborted. If the job is rerunnable or restartable, then the actions described below are taken. When the type argument is: • **hot** – All jobs are requeued except non-rerunnable jobs that were executing. Any rerunnable job which was executing when the server went down will be run immediately. This returns the server to the same state as when it went down. After those jobs are restarted, then normal scheduling takes place for all remaining queued jobs. If a job cannot be restarted immediately because of a missing resource, such as a node being down, the server will attempt to restart it periodically for up to 5 minutes. After that period, the server will revert to a normal state, as if warm started, and will no longer attempt to restart any remaining jobs which were running prior to the shutdown. • **warm** – All rerunnable jobs which were running when the server went down are requeued. All other jobs are maintained. New selections are made for which jobs are placed into execution. Warm is the default if `-t` is not specified. • **cold** – All jobs are deleted. Positive confirmation is required before this direction is accepted. • **create** – The server will discard any existing configuration files, queues and jobs, and initialize configuration files to the default values. The server is idled.

Files

File	Description
TORQUE_HOME/server_priv	Default directory for configuration files, typically `/usr/spool/pbs/server_priv`
TORQUE_HOME/server_logs	Directory for log files recorded by the server

Signal handling

On receipt of the following signals, the server performs the defined action:

Action	Description
SIGHUP	The current server log and accounting log are closed and reopened. This allows for the prior log to be renamed and a new log started from the time of the signal.
SIGINT	Causes an orderly shutdown of pbs_server.

Action	Description
SIGUSR1, SIGURS2	Causes server to increase and decrease logging levels, respectively.
SIGTERM	Causes an orderly shutdown of pbs_server.
SIGSHUTDN	On systems (Unicos) where SIGSHUTDN is defined, it also causes an orderly shutdown of the server.
SIGPIPE	This signal is ignored.

All other signals have their default behavior installed.

Exit status

If the server command fails to begin batch operation, the server exits with a value greater than zero.

Related topics

- pbs_mom(8B)
- pbsnodes(8B)
- qmgr(1B)
- qrun(8B)
- qsub(1B)
- qterm(8B)

Non-Adaptive Computing topics

- pbs_connect(3B)
- pbs_sched_basl(8B)
- pbs_sched_tcl(8B)
- qdisable(8B)
- qenable(8B)
- qstart(8B)
- qstop(8B)
- PBS External Reference Specification

pbs_track

Starts a new process and informs pbs_mom to start tracking it.

Synopsis

```
pbs_track -j <JOBID> [-b] <executable> [args]
```

Description

The pbs_track command tells a pbs_mom daemon to monitor the lifecycle and resource usage of the process that it launches using exec(). The pbs_mom is told about this new process via the Task Manager API, using tm_adopt(). The process must also be associated with a job that already exists on the pbs_mom.

By default, pbs_track will send its PID to TORQUE via tm_adopt(). It will then perform an exec(), causing <executable> to run with the supplied arguments. pbs_track will not return until the launched process has completed because it becomes the launched process.

This command can be considered related to the **pbsdsh** command which uses the tm_spawn() API call. The pbsdsh command asks a pbs_mom to launch and track a new process on behalf of a job. When it is not desirable or possible for the pbs_mom to spawn processes for a job, pbs_track can be used to allow an external entity to launch a process and include it as part of a job.

This command improves integration with TORQUE and SGI's MPT MPI implementation.

Options

Option	Description
-j <JOBID>	Job ID the new process should be associated with.
-b	Instead of having pbs_track send its PID to TORQUE, it will fork() first, send the child PID to TORQUE, and then execute from the forked child. This essentially "backgrounds" pbs_track so that it will return after the new process is launched.

Operands

The pbs_track command accepts a path to a program/executable (<executable>) and, optionally, one or more arguments to pass to that program.

Exit status

Because the pbs_track command becomes a new process (if used without **-b**), its exit status will match that of the new process. If the **-b** option is used, the exit status will be zero if no errors occurred before launching the new process.

If pbs_track fails, whether due to a bad argument or other error, the exit status will be set to a non-zero value.

Related topics

- pbsdsh(1B)

Non-Adaptive Computing topics

- tm_spawn(3B)

pbsdsh

Distribute tasks to nodes under pbs.

Synopsis

```
pbsdsh [-c copies] [-o] [-s] [-u] [-v] program [args]
pbsdsh [-n node] [-o] [-s] [-u] [-v] program [args]
pbsdsh [-h nodename] [-o] [-v] program [args]
```

Description

Executes (spawns) a normal Unix® program on one or more nodes under control of the Portable Batch System, PBS. Pbsdsh uses the Task Manager API (see tm_spawn(3)) to distribute the program on the allocated nodes.

When run without the **-c** or the **-n** option, pbsdsh will spawn the program on all nodes allocated to the PBS job. The spawns take place concurrently – all execute at (about) the same time.

Users will find the PBS_TASKNUM, PBS_NODENUM, and the PBS_VNODENUM environmental variables useful. They contain the TM task id, the node identifier, and the cpu (virtual node) identifier.

Options

Option	Name	Description
-c	copies	The program is spawned on the first Copies nodes allocated. This option is mutual exclusive with **-n**.
-h	hostname	The program is spawned on the node specified.
-n	node	The program is spawned on one node which is the n-th node allocated. This option is mutual exclusive with **-c**.
-o	---	Capture stdout of the spawned program. Normally stdout goes to the job's output.
-s	---	If this option is given, the program is run in turn on each node, one after the other.
-u	---	The program is run once on each node (unique). This ignores the number of allocated processors on a given node.
-v	---	Verbose output about error conditions and task exit status is produced.

Operands

The first operand, program, is the program to execute.

Additional operands are passed as arguments to the program.

Standard error

The pbsdsh command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the command, the exit status will be a value of zero.

If the pbsdsh command fails to process any operand, or fails to contact the MOM daemon on the localhost the command exits with a value greater than zero.

Related topics

- qsub(1B)

Non-Adaptive Computing topics

- tm_spawn(3B)

pbsnodes

PBS node manipulation.

Synopsis

```
pbsnodes [-{a|x}] [-q] [-s server] [node|:property]
pbsnodes -l [-q] [-s server] [state] [nodename|:property ...]
pbsnodes [-{c|d|o|r}] [-q] [-s server] [-n -l] [-N "note"] [node|:property]
```

Description

The pbsnodes command is used to mark nodes down, free or offline. It can also be used to list nodes and their state. Node information is obtained by sending a request to the PBS job server. Sets of nodes can be operated on at once by specifying a node property prefixed by a colon. (For more information, see Node states.)

Nodes do not exist in a single state, but actually have a set of states. For example, a node can be simultaneously "busy" and "offline". The "free" state is the absence of all other states and so is never combined with other states.

In order to execute pbsnodes with other than the **-a** or **-l** options, the user must have PBS Manager or Operator privilege.

Options

Option	Description
-a	All attributes of a node or all nodes are listed. This is the default if no flag is given.

Option	Description
-x	Same as **-a**, but the output has an XML-like format.
-c	Clear OFFLINE from listed nodes.
-d	Print MOM diagnosis on the listed nodes. Not yet implemented. Use **momctl** instead.
-o	Add the OFFLINE state. This is different from being marked DOWN. OFFLINE prevents new jobs from running on the specified nodes. This gives the administrator a tool to hold a node out of service without changing anything else. The OFFLINE state will never be set or cleared automatically by pbs_server; it is purely for the manager or operator.
-p	Purge the node record from pbs_server. Not yet implemented.
-r	Reset the listed nodes by clearing OFFLINE and adding DOWN state. pbs_server will ping the node and, if they communicate correctly, free the node.
-l	List node names and their state. If no state is specified, only nodes in the DOWN, OFFLINE, or UNKNOWN states are listed. Specifying a state string acts as an output filter. Valid state strings are "active", "all", "busy", "down", "free", "job-exclusive", "job-sharing", "offline", "reserve", "state-unknown", "time-shared", and "up". • Using **all** displays all nodes and their attributes. • Using **active** displays all nodes which are job-exclusive, job-sharing, or busy. • Using **up** displays all nodes in an "up state". Up states include job-exclusive, job-sharing, reserve, free, busy and time-shared. • All other strings display the nodes which are currently in the state indicated by the string.
-N	Specify a "note" attribute. This allows an administrator to add an arbitrary annotation to the listed nodes. To clear a note, use `-N ""` or `-N n`.
-n	Show the "note" attribute for nodes that are DOWN, OFFLINE, or UNKNOWN. This option requires **-l**.
-q	Suppress all error messages.
-s	Specify the PBS server's hostname or IP address.

Related topics

- pbs_server(8B)

Non-Adaptive Computing topics

- PBS External Reference Specification

qalter

Alter batch job.

Synopsis

```
qalter [-a date_time][-A account_string][-c interval][-e path_name]
       [-h hold_list][-j join_list][-k keep_list][-l resource_list]
       [-m mail_options][-M mail_list][-n][-N name][-o path_name]
       [-p priority][-q ][-r y|n][-S path_name_list][-u user_list]
       [-v variable_list][-W additional_attributes]
       [-t array_range]
       job_identifier ...
```

Description

The qalter command modifies the attributes of the job or jobs specified by job_identifier on the command line. Only those attributes listed as options on the command will be modified. If any of the specified attributes cannot be modified for a job for any reason, none of that job's attributes will be modified.

The qalter command accomplishes the modifications by sending a Modify Job batch request to the batch server which owns each job.

Options

Option	Name	Description
-a	date_time	Replaces the time at which the job becomes eligible for execution. The date_time argument syntax is: [[[[CC]YY]MM]DD]hhmm[.SS] If the month, MM, is not specified, it will default to the current month if the specified day DD, is in the future. Otherwise, the month will be set to next month. Likewise, if the day, DD, is not specified, it will default to today if the time hhmm is in the future. Otherwise, the day will be set to tomorrow. This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun.
-A	account_ string	Replaces the account string associated with the job. This attribute cannot be altered once the job has begun execution.

Option	Name	Description
-c	checkpoint_interval	Replaces the interval at which the job will be checkpointed. If the job executes upon a host which does not support checkpointing, this option will be ignored. The interval argument is specified as: • **n** – No checkpointing is to be performed. • **s** – Checkpointing is to be performed only when the server executing the job is shutdown. • **c** – Checkpointing is to be performed at the default minimum cpu time for the queue from which the job is executing. • **c=minutes** – Checkpointing is performed at intervals of the specified amount of time in minutes. Minutes are the number of minutes of CPU time used, not necessarily clock time. This value must be greater than zero. If the number is less than the default checkpoint time, the default time will be used. This attribute can be altered once the job has begun execution, but the new value does not take effect unless the job is rerun.
-e	path_name	Replaces the path to be used for the standard error stream of the batch job. The path argument is of the form: `[hostname:]path_name` where *hostname* is the name of a host to which the file will be returned and *path_name* is the path name on that host in the syntax recognized by POSIX 1003.1. The argument will be interpreted as follows: • **path_name** – Where *path_name* is not an absolute path name, then the qalter command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component. • **hostname:path_name** – Where *path_name* is not an absolute path name, then the qalter command will not expand the path name. The execution server will expand it relative to the home directory of the user on the system specified by hostname. This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun.

Option	Name	Description
-h	hold_list	Updates the the types of holds on the job. The hold_list argument is a string of one or more of the following characters: • **u** – Add the USER type hold. • **s** – Add the SYSTEM type hold if the user has the appropriate level of privilege. (Typically reserved to the batch administrator.) • **o** – Add the OTHER (or OPERATOR) type hold if the user has the appropriate level of privilege. (Typically reserved to the batch administrator and batch operator.) • **n** – Set to none and clear the hold types which could be applied with the users level of privilege. Repetition of characters is permitted, but "n" may not appear in the same option argument with the other three characters. This attribute can be altered once the job has begun execution, but the hold will not take effect unless the job is rerun.
-j	join	Declares which standard streams of the job will be merged together. The join argument value may be the characters "oe" and "eo", or the single character "n". A argument value of oe directs that the standard output and standard error streams of the job will be merged, intermixed, and returned as the standard output. A argument value of eo directs that the standard output and standard error streams of the job will be merged, intermixed, and returned as the standard error. A value of n directs that the two streams will be two separate files. This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun.

Option	Name	Description

-k	keep	Defines which if either of standard output or standard error of the job will be retained on the execution host. If set for a stream, this option overrides the path name for that stream.
		The argument is either the single letter "e", "o", or "n", or one or more of the letters "e" and "o" combined in either order.
		• **n** – No streams are to be retained.
		• **e** – The standard error stream is to retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by:
		`job_name.esequence`
		where job_name is the name specified for the job, and sequence is the sequence number component of the job identifier.
		• **o** – The standard output stream is to be retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by:
		`job_name.osequence`
		where job_name is the name specified for the job, and sequence is the sequence number component of the job identifier.
		• **eo** – Both the standard output and standard error streams will be retained.
		• **oe** – Both the standard output and standard error streams will be retained.
		This attribute cannot be altered once the job has begun execution.

-l	resource_ list	Modifies the list of resources that are required by the job. The resource_list argument is in the following syntax: `resource_name[=[value]][,resource_name[=[value]],...]` If a requested modification to a resource would exceed the resource limits for jobs in the current queue, the server will reject the request. If the job is running, only certain resources can be altered. Which resources can be altered in the run state is system dependent. A user may only lower the limit for those resources.
-m	mail_ options	Replaces the set of conditions under which the execution server will send a mail message about the job. The mail_options argument is a string which consists of the single character "n", or one or more of the characters "a", "b", and "e". If the character "n" is specified, no mail will be sent. For the letters "a", "b", and "e": • **a** – Mail is sent when the job is aborted by the batch system. • **b** – Mail is sent when the job begins execution. • **e** – Mail is sent when the job ends.

Option	Name	Description
-M	user_list	Replaces the list of users to whom mail is sent by the execution server when it sends mail about the job. The user_list argument is of the form: `user[@host][,user[@host],...]`

-n	node-exclu-sive	Sets or unsets exclusive node allocation on a job. Use the y and n options to enable or disable the feature. This affects only cpusets and compatible schedulers.
		`> qalter ... -n y #enables exclusive node allocation on a job`
		`> qalter ... -n n #disables exclusive node allocation on a job`

-N	name	Renames the job. The name specified may be up to and including 15 characters in length. It must consist of printable, non white space characters with the first character alphabetic.
-o	path	Replaces the path to be used for the standard output stream of the batch job. The path argument is of the form: `[hostname:]path_name` where *hostname* is the name of a host to which the file will be returned and *path_name* is the path name on that host in the syntax recognized by POSIX. The argument will be interpreted as follows: • **path_name** – Where *path_name* is not an absolute path name, then the qalter command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component. • **hostname:path_name** – Where *path_name* is not an absolute path name, then the qalter command will not expand the path name. The execution server will expand it relative to the home directory of the user on the system specified by hostname. This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun.
-p	priority	Replaces the priority of the job. The priority argument must be a integer between -1024 and +1023 inclusive. This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun.
-r	[y/n]	Declares whether the job is rerunable (see the **qrerun** command). The option argument c is a single character. PBS recognizes the following characters: y and n. If the argument is "y", the job is marked rerunable. If the argument is "n", the job is marked as not rerunable.

Option	Name	Description
-S	path	Declares the shell that interprets the job script. The option argument path_list is in the form: `path[@host][,path[@host],...]` Only one path may be specified for any host named. Only one path may be specified without the corresponding host name. The path selected will be the one with the host name that matched the name of the execution host. If no matching host is found, then the path specified (without a host) will be selected. If the `-S` option is not specified, the option argument is the null string, or no entry from the path_list is selected, the execution will use the login shell of the user on the execution host. This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun.
-t	array_ range	The array_range argument is an integer id or a range of integers. Multiple ids or id ranges can be combined in a comma delimted list. Examples: `-t 1-100` or `-t 1,10, 50-100` If an array range isn't specified, the command tries to operate on the entire array. The command acts on the array (or specified range of the array) just as it would on an individual job. An optional "slot limit" can be specified to limit the amount of jobs that can run concurrently in the job array. The default value is unlimited. The slot limit must be the last thing specified in the array_request and is delimited from the array by a percent sign (**%**). `qalter weatherSimulationArray[] -t %20` Here, the array weatherSimulationArray[] is configured to allow a maximum of 20 concurrently running jobs. Slot limits can be applied at job submit time with **qsub**, or can be set in a global server parameter policy with **max_slot_limit**.
-u	user_list	Replaces the user name under which the job is to run on the execution system. The user_list argument is of the form: `user[@host][,user[@host],...]` Only one user name may be given for per specified host. Only one of the user specifications may be supplied without the corresponding host specification. That user name will be used for execution on any host not named in the argument list. This attribute cannot be altered once the job has begun execution.
-W	additional_ attributes	The `-W` option allows for the modification of additional job attributes. Note if white space occurs anywhere within the option argument string or the equal sign, "**=**", occurs within an attribute_value string, then the string must be enclosed with either single or double quote marks. To see the attributes PBS currently supports within the `-W` option, see Table A-1: -W additional_attributes on page 165.

Table A-1: -W additional_attributes

Attribute	Description
depend=dependency_list	Redefines the dependencies between this and other jobs. The dependency_list is in the form: `type[:argument[:argument...][,type:argument...]` The argument is either a numeric count or a PBS job id accord ing to type. If argument is a count, it must be greater than 0. If it is a job id and is not fully specified in the form: `seq_number.server.name`, it will be expanded according to the default server rules. If argument is null (the preceding colon need not be specified), the dependency of the corresponding type is cleared (unset). • **synccount:count** – This job is the first in a set of jobs to be executed at the same time. Count is the number of additional jobs in the set. • **syncwith:jobid** – This job is an additional member of a set of jobs to be executed at the same time. In the above and following dependency types, jobid is the job identifier of the first job in the set. • **after:jobid [:jobid...]** – This job may be scheduled for execution at any point after jobs jobid have started execution. • **afterok:jobid [:jobid...]** – This job may be scheduled for execution only after jobs jobid have terminated with no errors. See the csh warning under "Extended Description". • **afternotok:jobid [:jobid...]** – This job may be scheduled for execution only after jobs jobid have terminated with errors. See the csh warning under "Extended Description". • **afterany:jobid [:jobid...]** – This job may be scheduled for execution after jobs jobid have terminated, with or without errors. • **on:count** – This job may be scheduled for execution after count dependencies on other jobs have been satisfied. This dependency is used in conjunction with any of the 'before' dependencies shown below. If job A has on:2, it will wait for two jobs with 'before' dependencies on job A to be fulfilled before running. • **before:jobid [:jobid...]** – When this job has begun execution, then jobs jobid... may begin. • **beforeok:jobid [:jobid...]** – If this job terminates execution without errors, then jobs jobid... may begin. See the csh warning under "Extended Description". • **beforenotok:jobid [:jobid...]** – If this job terminates execution with errors, then jobs jobid... may begin. See the csh warning under "Extended Description". • **beforeany:jobid [:jobid...]** – When this job terminates execution, jobs jobid... may begin. If any of the before forms are used, the job referenced by jobid must have been submitted with a dependency type of on. If any of the before forms are used, the jobs referenced by jobid must have the same owner as the job being altered. Otherwise, the dependency will not take effect. Error processing of the existence, state, or condition of the job specified to qalter is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the job will be deleted by the server. Mail will be sent to the job submitter stating the error.

Attribute	Description
group_list=g_list	Alters the group name under which the job is to run on the execution system. The g_list argument is of the form: `group[@host][,group[@host],...]` Only one group name may be given per specified host. Only one of the group specifications may be supplied without the corresponding host specification. That group name will used for execution on any host not named in the argument list.
stagein=file_list **stageout=file_list**	Alters which files are staged (copied) in before job start or staged out after the job completes execution. The file_list is in the form: `local_file@hostname:remote_file[,...]` The name local_file is the name on the system where the job executes. It may be an absolute path or a path relative to the home directory of the user. The name remote_file is the destination name on the host specified by hostname. The name may be absolute or relative to the users home directory on the destination host.

Operands

The qalter command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Standard error

Any error condition, either in processing the options or the operands, or any error received in reply to the batch requests will result in an error message being written to standard error.

Exit status

Upon successful processing of all the operands presented to the qalter command, the exit status will be a value of zero.

If the qalter command fails to process any operand, the command exits with a value greater than zero.

Copyright

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1, 2003 Edition, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 6, Copyright © 2001-2003 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html.

Related topics

- qdel
- qhold

- qrls
- qsub

Non-Adaptive Computing topics

- Batch Environment Services
- qmove
- touch

qchkpt

Checkpoint pbs batch jobs.

Synopsis

```
qchkpt <JOBID>[ <JOBID>] ...
```

Description

The qchkpt command requests that the PBS Mom generate a checkpoint file for a running job.

This is an extension to POSIX.2d.

The qchkpt command sends a Chkpt Job batch request to the server as described in the general section.

Options

None.

Operands

The qchkpt command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Examples

```
> qchkpt 3233 request a checkpoint for job 3233
```

Standard error

The qchkpt command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qchkpt command, the exit status will be a value of zero.

If the qchkpt command fails to process any operand, the command exits with a value greater than zero.

Related topics

- qhold(1B)
- qrls(1B)
- qalter(1B)
- qsub(1B)

Non-Adaptive Computing topics

- pbs_alterjob(3B)
- pbs_holdjob(3B),
- pbs_rlsjob(3B)
- pbs_job_attributes(7B)
- pbs_resources_unicos8(7B)

qdel

(delete job)

Synopsis

```
qdel [{-a <asynchronous delete>|-m <message>|-p|-W <delay>|-t <array_range>}]
<JOBID>[ <JOBID>]... | 'all' | 'ALL'
```

Description

The qdel command deletes jobs in the order in which their job identifiers are presented to the command. A job is deleted by sending a Delete Job batch request to the batch server that owns the job. A job that has been deleted is no longer subject to management by batch services.

A batch job may be deleted by its owner, the batch operator, or the batch administrator.

A batch job being deleted by a server will be sent a SIGTERM signal following by a SIGKILL signal. The time delay between the two signals is an attribute of the execution queue from which the job was run (set table by the administrator). This delay may be overridden by the **-W** option.

See the PBS ERS section 3.1.3.3, "Delete Job Request", for more information.

Options

Option	Name	Description
-a	asynchronous delete	Performs an asynchronous delete. The server responds to the user before contacting the MOM. The option `qdel -a all` performs `qdel all` due to restrictions from being single-threaded.

Option	Name	Description
-W	delay	Specifies the wait delay between the sending of the SIGTERM and SIGKILL signals. The argument is the length of time in seconds of the delay.
-p	purge	Forcibly purges the job from the server. This should only be used if a running job will not exit because its allocated nodes are unreachable. The admin should make every attempt at resolving the problem on the nodes. If a job's mother superior recovers after purging the job, any epilogue scripts may still run. This option is only available to a batch operator or the batch administrator.
-m	message	Specify a comment to be included in the email. The argument message specifies the comment to send. This option is only available to a batch operator or the batch administrator.
-t	array_range	The array_range argument is an integer id or a range of integers. Multiple ids or id ranges can be combined in a comma delimited list (examples: -t 1-100 or -t 1,10,50-100). If an array range isn't specified, the command tries to operate on the entire array. The command acts on the array (or specified range of the array) just as it would on an individual job.

Operands

The qdel command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

or

```
all
```

Examples

```
> qdel 1324
> qdel 1324-3 To delete one job of a job array
> qdel all To delete all jobs (Version 2.3.0 and later)
```

Standard error

The qdel command will write a diagnostic messages to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qdel command, the exit status will be a value of zero.

If the qdel command fails to process any operand, the command exits with a value greater than zero.

Related topics

- qsub(1B)
- qsig(1B)

Non-Adaptive Computing topics

- pbs_deljob(3B)

qgpumode

(GPU mode)

Synopsis

```
qgpumode -H host -g gpuid -m mode
```

Description

The qgpumode command specifies the mode for the GPU. This command triggers an immediate update of the pbs_server.

Options

Option	Description
-H	Specifies the host where the GPU i located.
-g	Specifies the ID of the GPU. This varies depending on the version of the Nvidia driver used. For driver 260.x, it is 0, 1, and so on. For driver 270.x, it is the PCI bus address, i.e., 0:5:0.
-m	Specifies the new mode for the GPU.

Related topics

- **qgpureset** on page 170

qgpureset

(reset GPU)

Synopsis

```
qgpureset -H host -g gpuid -p -v
```

Description

The qgpureset command resets the GPU.

Options

Option	Description
-H	Specifies the host where the GPU is located.
-g	Specifies the ID of the GPU. This varies depending on the version of the Nvidia driver used. For driver 260.x, it is 0, 1, and so on. For driver 270.x, it is the PCI bus address, i.e., 0:5:0.
-p	Specifies to reset the GPU's permanent ECC error count.
-v	Specifies to reset the GPU's volatile ECC error count.

Related topics

- **qgpumode** on page 170

qhold

(hold job)

Synopsis

```
qhold [{-h <HOLD LIST>|-t <array_range>}] <JOBID>[ <JOBID>] ...
```

Description

The qhold command requests that the server place one or more holds on a job. A job that has a hold is not eligible for execution. There are three supported holds: USER, OTHER (also known as operator), and SYSTEM.

A user may place a USER hold upon any job the user owns. An "operator", who is a user with "operator privilege," may place ether an USER or an OTHER hold on any job. The batch administrator may place any hold on any job.

If no **-h** option is given, the USER hold will be applied to the jobs described by the job_identifier operand list.

If the job identified by job_identifier is in the queued, held, or waiting states, then the hold type is added to the job. The job is then placed into held state if it resides in an execution queue.

If the job is in running state, then the following additional action is taken to interrupt the execution of the job. If checkpoint/restart is supported by the host system, requesting a hold on a running job will (1) cause

the job to be checkpointed, (2) the resources assigned to the job will be released, and (3) the job is placed in the held state in the execution queue.

If checkpoint/restart is not supported, qhold will only set the requested hold attribute. This will have no effect unless the job is rerun with the **qrerun** command.

Options

Option	Name	Description
-h	hold_list	The hold_list argument is a string consisting of one or more of the letters "u", "o", or "s" in any combination. The hold type associated with each letter is: • **u** – USER • **o** – OTHER • **s** – SYSTEM
-t	array_range	The array_range argument is an integer id or a range of integers. Multiple ids or id ranges can be combined in a comma delimited list (examples: -t 1-100 or -t 1,10,50-100) . If an array range isn't specified, the command tries to operate on the entire array. The command acts on the array (or specified range of the array) just as it would on an individual job.

Operands

The qhold command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Example

```
> qhold -h u 3233 place user hold on job 3233
```

Standard error

The qhold command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qhold command, the exit status will be a value of zero.

If the qhold command fails to process any operand, the command exits with a value greater than zero.

Related topics

- qrls(1B)
- qalter(1B)
- qsub(1B)

Non-Adaptive Computing topics

- pbs_alterjob(3B)
- pbs_holdjob(3B)
- pbs_rlsjob(3B)
- pbs_job_attributes(7B)
- pbs_resources_unicos8(7B)

qmgr

(PBS Queue Manager) PBS batch system manager.

Synopsis

```
qmgr [-a] [-c command] [-e] [-n] [-z] [server...]
```

Description

The qmgr command provides an administrator interface to query and configure batch system parameters (see **Server parameters** on page 209).

The command reads directives from standard input. The syntax of each directive is checked and the appropriate request is sent to the batch server or servers.

The list or print subcommands of qmgr can be executed by general users. Creating or deleting a queue requires PBS Manager privilege. Setting or unsetting server or queue attributes requires PBS Operator or Manager privilege.

By default, the user root is the only PBS Operator and Manager. To allow other users to be privileged, the server attributes operators and managers will need to be set (i.e., as root, issue 'qmgr -c 'set server managers += <USER1>@<HOST>'). See "TORQUE/PBS Integration Guide - RM Access Control" in the Moab Workload Manager Administrator's Guide.

If qmgr is invoked without the **-c** option and standard output is connected to a terminal, qmgr will write a prompt to standard output and read a directive from standard input.

Commands can be abbreviated to their minimum unambiguous form. A command is terminated by a new line character or a semicolon, "**;**", character. Multiple commands may be entered on a single line. A command may extend across lines by escaping the new line character with a back-slash "\".

Comments begin with the "#" character and continue to end of the line. Comments and blank lines are ignored by qmgr.

Options

Option	Name	Description
-a	---	Abort qmgr on any syntax errors or any requests rejected by a server.
-c	command	Execute a single command and exit qmgr.
-e	---	Echo all commands to standard output.
-n	---	No commands are executed, syntax checking only is performed.
-z	---	No errors are written to standard error.

Operands

The *server* operands identify the name of the batch server to which the administrator requests are sent. Each *server* conforms to the following syntax:

```
host_name[:port]
```

where *host_name* is the network name of the host on which the server is running and *port* is the port number to which to connect. If *port* is not specified, the default port number is used.

If *server* is not specified, the administrator requests are sent to the local server.

Standard input

The qmgr command reads standard input for directives until end of file is reached, or the exit or quit directive is read.

Standard output

If Standard Output is connected to a terminal, a command prompt will be written to standard output when qmgr is ready to read a directive.

If the **-e** option is specified, qmgr will echo the directives read from standard input to standard output.

Standard error

If the **-z** option is not specified, the qmgr command will write a diagnostic message to standard error for each error occurrence.

Directive syntax

A qmgr directive is one of the following forms:

```
command server [names] [attr OP value[,attr OP value,...]]
command queue [names] [attr OP value[,attr OP value,...]]
command node [names] [attr OP value[,attr OP value,...]]
```

where *command* is the command to perform on a object.

Commands are:

Command	Description
active	Sets the active objects. If the active objects are specified, and the name is not given in a qmgr cmd the active object names will be used.
create	Is to create a new object, applies to queues and nodes.
delete	Is to destroy an existing object, applies to queues and nodes.
set	Is to define or alter attribute values of the object.
unset	Is to clear the value of attributes of the object. This form does not accept an OP and value, only the attribute name.
list	Is to list the current attributes and associated values of the object.
print	Is to print all the queue and server attributes in a format that will be usable as input to the qmgr command.
names	Is a list of one or more names of specific objects The name list is in the form: `[name][@server][,queue_name[@server]...]` with no intervening white space. The name of an object is declared when the object is first created. If the name is @server, then all the objects of specified type at the server will be affected.
attr	Specifies the name of an attribute of the object which is to be set or modified. If the attribute is one which consist of a set of resources, then the attribute is specified in the form: `attribute_name.resource_name`
OP	Operation to be performed with the attribute and its value: • "**=**" – set the value of the attribute. If the attribute has a existing value, the current value is replaced with the new value. • "**+=**" – increase the current value of the attribute by the amount in the new value. • "**-=**" – decrease the current value of the attribute by the amount in the new value.
value	The value to assign to an attribute. If the value includes white space, commas or other special characters, such as the "#" character, the value string must be enclosed in quote marks (").

The following are examples of qmgr directives:

```
create queue fast priority=10,queue_type=e,enabled = true,max_running=0
set queue fast max_running +=2
create queue little
set queue little resources_max.mem=8mw,resources_max.cput=10
unset queue fast max_running
set node state = "down,offline"
active server s1,s2,s3
list queue @server1
set queue max_running = 10       - uses active queues
```

Exit status

Upon successful processing of all the operands presented to the qmgr command, the exit status will be a value of zero.

If the qmgr command fails to process any operand, the command exits with a value greater than zero.

Related topics

- pbs_server(8B)

Non-Adaptive Computing topics

- pbs_queue_attributes (7B)
- pbs_server_attributes (7B)
- qstart (8B), qstop (8B)
- qenable (8B), qdisable (8)
- PBS External Reference Specification

qrerun

(Rerun a batch job)

Synopsis

```
qrerun [{-f}] <JOBID>[ <JOBID>] ...
```

Description

The qrerun command directs that the specified jobs are to be rerun if possible. To rerun a job is to terminate the session leader of the job and return the job to the queued state in the execution queue in which the job currently resides.

If a job is marked as not rerunable then the rerun request will fail for that job. If the mini-server running the job is down, or it rejects the request, the Rerun Job batch request will return a failure unless **-f** is used.

Using **-f** violates IEEE Batch Processing Services Standard and should be handled with great care. It should only be used under exceptional circumstances. The best practice is to fix the problem mini-server host and let qrerun run normally. The nodes may need manual cleaning (see the `-r` option on the **qsub** and **qalter** commands).

Options

Option	Description
-f	Force a rerun on a job

```
qrerun -f 15406
```

Operands

The qrerun command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Standard error

The qrerun command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qrerun command, the exit status will be a value of zero.

If the qrerun command fails to process any operand, the command exits with a value greater than zero.

Examples

```
> qrerun 3233
```

(Job 3233 will be re-run.)

Related topics

- qsub(1B)
- qalter(1B)

Non-Adaptive Computing topics

- pbs_alterjob(3B)
- pbs_rerunjob(3B)

qrls

(Release hold on PBS batch jobs)

Synopsis

```
qrls [{-h <HOLD LIST>|-t <array_range>}] <JOBID>[ <JOBID>] ...
```

Description

The qrls command removes or releases holds which exist on batch jobs.

A job may have one or more types of holds which make the job ineligible for execution. The types of holds are USER, OTHER, and SYSTEM. The different types of holds may require that the user issuing the qrls command have special privileges. A user may always remove a USER hold on their own jobs, but only privileged users can remove OTHER or SYSTEM holds. An attempt to release a hold for which the user does not have the correct privilege is an error and no holds will be released for that job.

If no **-h** option is specified, the USER hold will be released.

If the job has no execution_time pending, the job will change to the queued state. If an execution_time is still pending, the job will change to the waiting state.

Options

Command	Name	Description
-h	hold_list	Defines the types of hold to be released from the jobs. The hold_list option argument is a string consisting of one or more of the letters "u", "o", and "s" in any combination. The hold type associated with each letter is: • **u** – USER • **o** – OTHER • **s** – SYSTEM
-t	array_range	The array_range argument is an integer id or a range of integers. Multiple ids or id ranges can be combined in a comma delimited list. Examples: -t 1-100 or -t 1,10,50-100 If an array range isn't specified, the command tries to operate on the entire array. The command acts on the array (or specified range of the array) just as it would on an individual job.

Operands

The qrls command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Examples

```
> qrls -h u 3233 release user hold on job 3233
```

Standard error

The qrls command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qrls command, the exit status will be a value of zero.

If the qrls command fails to process any operand, the command exits with a value greater than zero.

Related topics

Related topics

- qsub(1B)
- qalter(1B)
- qhold(1B)

Non-Adaptive Computing topics)

- pbs_alterjob(3B)
- pbs_holdjob(3B)
- pbs_rlsjob(3B)

qrun

(Run a batch job)

Synopsis

```
qrun [{-H <HOST>|-a}] <JOBID>[ <JOBID>] ...
```

Overview

The qrun command runs a job.

Format

-H	
Format	<STRING> Host Identifier
Default	---
Description	Specifies the host within the cluster on which the job(s) are to be run. The host argument is the name of a host that is a member of the cluster of hosts managed by the server. If the option is not specified, the server will select the "worst possible" host on which to execute the job.
Example	`qrun -H hostname 15406`

-a	
Format	---
Default	---
Description	Run the job(s) asynchronously.
Example	`qrun -a 15406`

Command details

The qrun command is used to force a batch server to initiate the execution of a batch job. The job is run regardless of scheduling position or resource requirements.

In order to execute qrun, the user must have PBS Operation or Manager privileges.

Examples

```
> qrun 3233
```

(Run job 3233.)

qsig

(Signal a job)

Synopsis

```
qsig [{-s <SIGNAL>}] <JOBID>[ <JOBID>] ...
     [-a]
```

Description

The qsig command requests that a signal be sent to executing batch jobs. The signal is sent to the session leader of the job. If the **-s** option is not specified, SIGTERM is sent. The request to signal a batch job will be rejected if:

- The user is not authorized to signal the job.
- The job is not in the running state.
- The requested signal is not supported by the system upon which the job is executing.

The qsig command sends a Signal Job batch request to the server which owns the job.

Options

Option	Name	Description
-s	signal	Declares which signal is sent to the job. The signal argument is either a signal name, e.g. SIGKILL, the signal name without the SIG prefix, e.g. KILL, or a unsigned signal number, e.g. 9. The signal name SIGNULL is allowed; the server will send the signal 0 to the job which will have no effect on the job, but will cause an obituary to be sent if the job is no longer executing. Not all signal names will be recognized by qsig. If it doesn't recognize the signal name, try issuing the signal number instead. Two special signal names, "suspend" and "resume", are used to suspend and resume jobs. Cray systems use the Cray-specific suspend()/resume() calls. On non-Cray system, suspend causes a SIGTSTP to be sent to all processes in the job's top task, wait 5 seconds, and then send a SIGSTOP to all processes in all tasks on all nodes in the job. This differs from TORQUE 2.0.0 which did not have the ability to propagate signals to sister nodes. Resume sends a SIGCONT to all processes in all tasks on all nodes. When suspended, a job continues to occupy system resources but is not executing and is not charged for walltime. The job will be listed in the "S" state. Manager or operator privilege is required to suspend or resume a job. Interactive jobs may not resume properly because the top-level shell will background the suspended child process.
-a	asynchronously	Makes the command run asynchronously.

Operands

The qsig command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Examples

```
> qsig -s SIGKILL 3233    send a SIGKILL to job 3233
> qsig -s KILL 3233       send a SIGKILL to job 3233
> qsig -s 9 3233          send a SIGKILL to job 3233
```

Standard error

The qsig command will write a diagnostic messages to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qsig command, the exit status will be a value of zero.

If the qsig command fails to process any operand, the command exits with a value greater than zero.

Related topics

- qsub(1B)

Non-Adaptive Computing topics

- pbs_sigjob(3B)
- pbs_resources_*(7B) where * is system type
- PBS ERS

qstat

Show status of PBS batch jobs.

Synopsis

```
qstat [-f [-1]][-W site_specific] [job_identifier... | destination...] [time]
qstat [-a|-i|-r|-e] [-n [-1]] [-s] [-G|-M] [-R] [-u user_list]
[job_identifier... | destination...]
qstat -Q [-f [-1]][-W site_specific] [destination...]
qstat -q [-G|-M] [destination...]
qstat -B [-f [-1]][-W site_specific] [server_name...]
qstat -t
```

Description

The qstat command is used to request the status of jobs, queues, or a batch server. The requested status is written to standard out.

When requesting job status, synopsis format 1 or 2, qstat will output information about each job_identifier or all jobs at each destination. Jobs for which the user does not have status privilege are not displayed.

When requesting queue or server status, synopsis format 3 through 5, qstat will output information about each destination.

You can configure TORQUE with CFLAGS='DTXT' to change the alignment of text in qstat output. This noticeably improves qstat -r output.

Options

Option	Description
-f	Specifies that a full status display be written to standard out. The [time] value is the amount of walltime, in seconds, remaining for the job. [time] does not account for walltime multipliers.

Option	Description
-a	All jobs are displayed in the alternative format (see **Standard output** on page 185). If the operand is a destination id, all jobs at that destination are displayed. If the operand is a job id, information about that job is displayed.
-e	If the operand is a job id or not specified, only jobs in executable queues are displayed. Setting the PBS_QSTAT_EXECONLY environment variable will also enable this option.
-i	Job status is displayed in the alternative format. For a destination id operand, status for jobs at that destination which are not running are displayed. This includes jobs which are queued, held or waiting. If an operand is a job id, status for that job is displayed regardless of its state.
-r	If an operand is a job id, status for that job is displayed. For a destination id operand, status for jobs at that destination which are running are displayed, this includes jobs which are suspended.
-n	In addition to the basic information, nodes allocated to a job are listed.
-1	In combination with **-n**, the -1 option puts all of the nodes on the same line as the job ID. In combination with **-f**, attributes are not folded to fit in a terminal window. This is intended to ease the parsing of the qstat output.
-s	In addition to the basic information, any comment provided by the batch administrator or scheduler is shown.
-G	Show size information in giga-bytes.
-M	Show size information, disk or memory in mega-words. A word is considered to be 8 bytes.
-R	In addition to other information, disk reservation information is shown. Not applicable to all systems.

-t	Normal qstat output displays a summary of the array instead of the entire array, job for job. `qstat -t` expands the output to display the entire array. Note that arrays are now named with brackets following the array name; for example:
	`dbeer@napali:~/dev/torque/array_changes$ echo sleep 20 | qsub -t 0-299 189[].napali`
	Individual jobs in the array are now also noted using square brackets instead of dashes; for example, here is part of the output of `qstat -t` for the preceding array:
	`189[299].napali STDIN[299] dbeer 0 Q batch`

Option	Description
-u	Job status is displayed in the alternative format. If an operand is a job id, status for that job is displayed. For a destination id operand, status for jobs at that destination which are owned by the user(s) listed in user_list are displayed. The syntax of the user_list is: `user_name[@host][,user_name[@host],...]` Host names may be wild carded on the left end, e.g. "*.nasa.gov". User_name without a "@host" is equivalent to "user_name@*", that is at any host.
-Q	Specifies that the request is for queue status and that the operands are destination identifiers.
-q	Specifies that the request is for queue status which should be shown in the alternative format.
-B	Specifies that the request is for batch server status and that the operands are the names of servers.

Operands

If neither the **-Q** nor the **-B** option is given, the operands on the qstat command must be either job identifiers or destinations identifiers.

If the operand is a job identifier, it must be in the following form:

```
sequence_number[.server_name][@server]
```

where *sequence_number.server_name* is the job identifier assigned at submittal time (see **qsub**). If the *.server_name* is omitted, the name of the default server will be used. If *@server* is supplied, the request will be for the job identifier currently at that Server.

If the operand is a destination identifier, it is one of the following three forms:

- queue
- @server
- queue@server

If queue is specified, the request is for status of all jobs in that queue at the default server. If the @server form is given, the request is for status of all jobs at that server. If a full destination identifier, queue@server, is given, the request is for status of all jobs in the named queue at the named server.

If the **-Q** option is given, the operands are destination identifiers as specified above. If queue is specified, the status of that queue at the default server will be given. If queue@server is specified, the status of the named queue at the named server will be given. If @server is specified, the status of all queues at the named server will be given. If no destination is specified, the status of all queues at the default server will be given.

If the **-B** option is given, the operand is the name of a server.

Standard output

Displaying job status

If job status is being displayed in the default format and the **-f** option is not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the job identifier assigned by PBS.
- the job name given by the submitter.
- the job owner.
- the CPU time used.
- the job state:

Item	Description
C	Job is completed after having run.
E	Job is exiting after having run.
H	Job is held.
Q	Job is queued, eligible to run or routed.
R	Job is queued, eligible to run or routed.
T	Job is being moved to new location.
W	Job is waiting for its execution time (**-a** option) to be reached.
S	(Unicos only) Job is suspended.

- the queue in which the job resides.

If job status is being displayed and the **-f** option is specified, the output will depend on whether qstat was compiled to use a Tcl interpreter. See **Configuration** on page 188 for details. If Tcl is not being used, full display for each job consists of the header line:

```
Job Id: job identifier
```

Followed by one line per job attribute of the form:

```
attribute_name = value
```

If any of the options **-a**, **-i**, **-r**, **-u**, **-n**, **-s**, **-G**, or **-M** are provided, the alternative display format for jobs is used. The following items are displayed on a single line, in the specified order, separated by white space:

- the job identifier assigned by PBS
- the job owner
- the queue in which the job currently resides
- the job name given by the submitter
- the session id (if the job is running)
- the number of nodes requested by the job
- the number of cpus or tasks requested by the job
- the amount of memory requested by the job
- either the cpu time, if specified, or wall time requested by the job, (hh:mm)
- the jobs current state
- the amount of cpu time or wall time used by the job (hh:mm)

If the **-r** option is provided, the line contains:

- the job identifier assigned by PBS
- the job owner
- the queue in which the job currently resides
- the number of nodes requested by the job
- the number of cpus or tasks requested by the job
- the amount of memory requested by the job
- either the cpu time or wall time requested by the job
- the jobs current state
- the amount of cpu time or wall time used by the job
- the amount of SRFS space requested on the big file system
- the amount of SRFS space requested on the fast file system
- the amount of space requested on the parallel I/O file system

The last three fields may not contain useful information at all sites or on all systems

Displaying queue status

If queue status is being displayed and the **-f** option was not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the queue name
- the maximum number of jobs that may be run in the queue concurrently
- the total number of jobs in the queue
- the enable or disabled status of the queue
- the started or stopped status of the queue

- for each job state, the name of the state and the number of jobs in the queue in that state
- the type of queue, execution or routing

If queue status is being displayed and the **-f** option is specified, the output will depend on whether qstat was compiled to use a Tcl interpreter. See the configuration section for details. If Tcl is not being used, the full display for each queue consists of the header line:

```
Queue: queue_name
```

Followed by one line per queue attribute of the form:

```
attribute_name = value
```

If the **-Q** option is specified, queue information is displayed in the alternative format: The following information is displayed on a single line:

- the queue name
- the maximum amount of memory a job in the queue may request
- the maximum amount of cpu time a job in the queue may request
- the maximum amount of wall time a job in the queue may request
- the maximum amount of nodes a job in the queue may request
- the number of jobs in the queue in the running state
- the number of jobs in the queue in the queued state
- the maximum number (limit) of jobs that may be run in the queue concurrently
- the state of the queue given by a pair of letters:
    - either the letter **E** if the queue is Enabled or **D** if Disabled

      and
    - either the letter **R** if the queue is Running (started) or **S** if Stopped.

Displaying server status

If batch server status is being displayed and the **-f** option is not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the server name
- the maximum number of jobs that the server may run concurrently
- the total number of jobs currently managed by the server
- the status of the server
- for each job state, the name of the state and the number of jobs in the server in that state

If server status is being displayed and the **-f** option is specified, the output will depend on whether qstat was compiled to use a Tcl interpreter. See the configuration section for details. If Tcl is not being used, the full display for the server consist of the header line:

```
Server: server name
```

Followed by one line per server attribute of the form:

```
attribute_name = value
```

Standard error

The qstat command will write a diagnostic message to standard error for each error occurrence.

Configuration

If qstat is compiled with an option to include a Tcl interpreter, using the **-f** flag to get a full display causes a check to be made for a script file to use to output the requested information. The first location checked is `$HOME/.qstatrc`. If this does not exist, the next location checked is administrator configured. If one of these is found, a Tcl interpreter is started and the script file is passed to it along with three global variables. The command line arguments are split into two variable named flags and operands . The status information is passed in a variable named objects . All of these variables are Tcl lists. The flags list contains the name of the command (usually "qstat") as its first element. Any other elements are command line option flags with any options they use, presented in the order given on the command line. They are broken up individually so that if two flags are given together on the command line, they are separated in the list. For example, if the user typed:

```
qstat -QfWbigdisplay
```

the flags list would contain

```
qstat -Q -f -W bigdisplay
```

The operands list contains all other command line arguments following the flags. There will always be at least one element in operands because if no operands are typed by the user, the default destination or server name is used. The objects list contains all the information retrieved from the server(s) so the Tcl interpreter can run once to format the entire output. This list has the same number of elements as the operands list. Each element is another list with two elements.

The first element is a string giving the type of objects to be found in the second. The string can take the values "server", "queue", "job" or "error".

The second element will be a list in which each element is a single batch status object of the type given by the string discussed above. In the case of "error", the list will be empty. Each object is again a list. The first element is the name of the object. The second is a list of attributes.

The third element will be the object text.

All three of these object elements correspond with fields in the structure batch_status which is described in detail for each type of object by the man pages for pbs_statjob(3), pbs_statque(3), and pbs_statserver(3). Each attribute in the second element list whose elements correspond with the attrl structure. Each will be a list with two elements. The first will be the attribute name and the second will be the attribute value.

Exit status

Upon successful processing of all the operands presented to the qstat command, the exit status will be a value of zero.

If the qstat command fails to process any operand, the command exits with a value greater than zero.

Related topics

- qalter(1B)
- qsub(1B)

Non-Adaptive Computing topics

- pbs_alterjob(3B)
- pbs_statjob(3B)
- pbs_statque(3B)
- pbs_statserver(3B)
- pbs_submit(3B)
- pbs_job_attributes(7B)
- pbs_queue_attributes(7B)
- pbs_server_attributes(7B)
- qmgr query_other_jobs parameter (allow non-admin users to see other users' jobs
- pbs_resources_*(7B) where * is system type
- PBS ERS

qsub

Submit PBS job.

Synopsis

```
qsub [-a date_time] [-A account_string] [-b secs] [-c checkpoint_options]
     [-C directive_prefix] [-d path] [-D path] [-e path] [-f] [-F] [-h]
     [-I ] [-j join ] [-k keep ] [-l resource_list ]
     [-m mail_options] [-M user_list] [-n] [-N name] [-o path]
     [-p priority] [-P user[:group]] [-q destination] [-r c] [-S path_to_shell
(s)]
     [-t array_request] [-u user_list]
     [-v variable_list] [-V] [-W additional_attributes] [-x] [-X] [-z] [script]
```

Description

To create a job is to submit an executable script to a batch server. The batch server will be the default server unless the **-q** option is specified. The command parses a script prior to the actual script execution; it does not execute a script itself. All script-writing rules remain in effect, including the "**#!**" at the head of the file (see discussion of PBS_DEFAULT under **Environment variables** on page 203). Typically, the script is a shell script which will be executed by a command shell such as sh or csh.

Options on the qsub command allow the specification of attributes which affect the behavior of the job.

The qsub command will pass certain environment variables in the Variable_List attribute of the job. These variables will be available to the job. The value for the following variables will be taken from the environment of the qsub command: HOME, LANG, LOGNAME, PATH, MAIL, SHELL, and TZ. These values will be assigned to a new name which is the current name prefixed with the string "PBS_O_". For example, the

job will have access to an environment variable named PBS_O_HOME which have the value of the variable HOME in the qsub command environment.

In addition to the above, the following environment variables will be available to the batch job:

Variable	Description
PBS_O_HOST	The name of the host upon which the qsub command is running.
PBS_SERVER	The hostname of the pbs_server which qsub submits the job to.
PBS_O_QUEUE	The name of the original queue to which the job was submitted.
PBS_O_WORK-DIR	The absolute path of the current working directory of the qsub command.
PBS_ARRAYID	Each member of a job array is assigned a unique identifier (see **-t** option).
PBS_ENVI-RONMENT	Set to PBS_BATCH to indicate the job is a batch job, or to PBS_INTERACTIVE to indicate the job is a PBS interactive job (see **-I** option).
PBS_GPUFILE	The name of the file containing the list of assigned GPUs.
PBS_JOBID	The job identifier assigned to the job by the batch system. It can be used in the stdout and stderr paths. TORQUE replaces `$PBS_JOBID` with the job's jobid (for example, `#PBS -o /tmp/$PBS_JOBID.output`).
PBS_JOBNAME	The job name supplied by the user.
PBS_NODEFILE	The name of the file contain the list of nodes assigned to the job (for parallel and cluster systems).
PBS_QUEUE	The name of the queue from which the job is executed.

Options

Option	Name	Description
-a	date_time	Declares the time after which the job is eligible for execution. The date_time argument is in the form: `[[[[CC]YY]MM]DD]hhmm[.SS]` where *CC* is the first two digits of the year (the century), *YY* is the second two digits of the year, *MM* is the two digits for the month, *DD* is the day of the month, *hh* is the hour, *mm* is the minute, and the optional *SS* is the seconds. If the month (MM) is not specified, it will default to the current month if the specified day (DD) is in the future. Otherwise, the month will be set to next month. Likewise, if the day (DD) is not specified, it will default to today if the time (hhmm) is in the future. Otherwise, the day will be set to tomorrow. For example, if you submit a job at 11:15 am with a time of `-a 1110`, the job will be eligible to run at 11:10 am tomorrow.
-A	account_ string	Defines the account string associated with the job. The account_string is an undefined string of characters and is interpreted by the server which executes the job. See section 2.7.1 of the PBS ERS.
-b	seconds	Defines the maximum number of seconds qsub will block attempting to contact pbs_ server. If pbs_server is down, or for a variety of communication failures, qsub will continually retry connecting to pbs_server for job submission. This value overrides the CLIENTRETRY parameter in `torque.cfg`. This is a non-portable TORQUE extension. Portability-minded users can use the PBS_CLIENTRETRY environmental variable. A negative value is interpreted as infinity. The default is 0.
-c	checkpoint_ options	Defines the options that will apply to the job. If the job executes upon a host which does not support checkpoint, these options will be ignored. Valid checkpoint options are: • **none** – No checkpointing is to be performed. • **enabled** – Specify that checkpointing is allowed but must be explicitly invoked by either the **qhold** or **qchkpt** commands. • **shutdown** – Specify that checkpointing is to be done on a job at pbs_mom shutdown. • **periodic** – Specify that periodic checkpointing is enabled. The default interval is 10 minutes and can be changed by the $checkpoint_interval option in the MOM config file or by specifying an interval when the job is submitted • **interval=minutes** – Checkpointing is to be performed at an interval of minutes, which is the integer number of minutes of wall time used by the job. This value must be greater than zero. • **depth=number** – Specify a number (depth) of checkpoint images to be kept in the checkpoint directory. • **dir=path** – Specify a checkpoint directory (default is `/var/spool/torque/checkpoint`).

Option	Name	Description
-C	directive_ prefix	Defines the prefix that declares a directive to the qsub command within the script file. (See the paragraph on script directives under **Extended description** on page 204.) If the -C option is presented with a directive_prefix argument that is the null string, qsub will not scan the script file for directives.
-d	path	Defines the working directory path to be used for the job. If the -d option is not specified, the default working directory is the home directory. This option sets the environment variable PBS_O_INITDIR.
-D	path	Defines the root directory to be used for the job. This option sets the environment variable PBS_O_ROOTDIR.
-e	path	Defines the path to be used for the standard error stream of the batch job. The path argument is of the form: `[hostname:]path_name` where *hostname* is the name of a host to which the file will be returned, and *path_name* is the path name on that host in the syntax recognized by POSIX. The argument will be interpreted as follows: • **path_name** – where *path_name* is not an absolute path name, then the qsub command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component. • **hostname:path_name** – where *path_name* is not an absolute path name, then the qsub command will not expand the path name relative to the current working directory of the command. On delivery of the standard error, the path name will be expanded relative to the users home directory on the hostname system. • **path_name** – where *path_name* specifies an absolute path name, then the qsub will supply the name of the host on which it is executing for the hostname. • **hostname:path_name** – where *path_name* specifies an absolute path name, the path will be used as specified. If the -e option is not specified, the default file name for the standard error stream will be used. The default name has the following form: • **job_name.esequence_number** – where *job_name* is the name of the job (see the **-n** name option) and *sequence_number* is the job number assigned when the job is submitted.

Option	Name	Description
-f	---	Job is made fault tolerant. Jobs running on multiple nodes are periodically polled by mother superior. If one of the nodes fails to report, the job is canceled by mother superior and a failure is reported. If a job is fault tolerant, it will not be canceled based on failed polling (no matter how many nodes fail to report). This may be desirable if transient network failures are causing large jobs not to complete, where ignoring one failed polling attempt can be corrected at the next polling attempt. If TORQUE is compiled with PBS_NO_POSIX_VIOLATION (there is no config option for this), you have to use `-W fault_tolerant=true` to mark the job as fault tolerant.
-F	---	Specfies the arguments that will be passed to the job script when the script is launched. The accepted syntax is: `qsub -F "myarg1 myarg2 myarg3=myarg3value" myscript2.sh` Quotation marks are required. qsub will fail with an error message if the argument following `-F` is not a quoted value. The pbs_mom server will pass the quoted value as arguments to the job script when it launches the script.
-h	---	Specifies that a user hold be applied to the job at submission time.
-I	---	Declares that the job is to be run "interactively". The job will be queued and scheduled as any PBS batch job, but when executed, the standard input, output, and error streams of the job are connected through qsub to the terminal session in which qsub is running. Interactive jobs are forced to not rerunable. See **Extended description** on page 204 for additional information of interactive jobs.
-j	join	Declares if the standard error stream of the job will be merged with the standard output stream of the job. An option argument value of **oe** directs that the two streams will be merged, intermixed, as standard output. An option argument value of **eo** directs that the two streams will be merged, intermixed, as standard error. If the join argument is **n** or the option is not specified, the two streams will be two separate files.

Option	Name	Description

-k	keep	Defines which (if either) of standard output or standard error will be retained on the execution host. If set for a stream, this option overrides the path name for that stream. If not set, neither stream is retained on the execution host.
		The argument is either the single letter "e" or "o", or the letters "e" and "o" combined in either order. Or the argument is the letter "n".
		• **e** – The standard error stream is to retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by:
		`job_name.esequence`
		where *job_name* is the name specified for the job, and *sequence* is the sequence number component of the job identifier.
		• **o** – The standard output stream is to retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by:
		`job_name.osequence`
		where *job_name* is the name specified for the job, and *sequence* is the sequence number component of the job identifier.
		• **eo** – Both the standard output and standard error streams will be retained.
		• **oe** – Both the standard output and standard error streams will be retained.
		• **n** – Neither stream is retained.

-l	resource_ list	Defines the resources that are required by the job and establishes a limit to the amount of resource that can be consumed. If not set for a generally available resource, such as CPU time, the limit is infinite. The resource_list argument is of the form: `resource_name[=[value]][,resource_name[=[value]],...]` In this situation, you should request the more inclusive resource first. For example, a request for procs should come before a gres request. In TORQUE 3.0.2 or later, qsub supports the mapping of `-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` indicating they are not concerned with the GPUs in relation to the NUMA nodes they request, they only want a total of 20 cores and 5 GPUs. For more information, see **Requesting resources** on page 38. For information on specifying multiple types of resources for allocation, see "Multi-Req Support" under "General Job Policies" in the Moab Workload Manager documentation.

Option	Name	Description
-m	mail_ options	Defines the set of conditions under which the execution server will send a mail message about the job. The mail_options argument is a string which consists of either the single character "n", or one or more of the characters "a", "b", and "e". If the character "n" is specified, no normal mail is sent. Mail for job cancels and other events outside of normal job processing are still sent. For the letters "a", "b", and "e": • **a** – Mail is sent when the job is aborted by the batch system. • **b** – Mail is sent when the job begins execution. • **e** – Mail is sent when the job terminates. If the `-m` option is not specified, mail will be sent if the job is aborted.
-M	user_list	Declares the list of users to whom mail is sent by the execution server when it sends mail about the job. The user_list argument is of the form: `user[@host][,user[@host],...]` If unset, the list defaults to the submitting user at the `qsub` host, i.e. the job owner.
-n	node-exclu- sive	Allows a user to specify an exclusive-node access/allocation request for the job. This affects only cpusets and compatible schedulers (see **Linux cpuset support** on page 66).
-N	name	Declares a name for the job. The name specified may be an unlimited number of characters in length. It must consist of printable, non white space characters with the first character alphabetic. If the `-N` option is not specified, the job name will be the base name of the job script file specified on the command line. If no script file name was specified and the script was read from the standard input, then the job name will be set to STDIN.

Option	Name	Description
-o	path	Defines the path to be used for the standard output stream of the batch job. The path argument is of the form: `[hostname:]path_name` where *hostname* is the name of a host to which the file will be returned, and *path_name* is the path name on that host in the syntax recognized by POSIX. The argument will be interpreted as follows: • **path_name** – where *path_name* is not an absolute path name, then the qsub command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component. • **hostname:path_name** – where *path_name* is not an absolute path name, then the qsub command will not expand the path name relative to the current working directory of the command. On delivery of the standard output, the path name will be expanded relative to the users home directory on the hostname system. • **path_name** – where *path_name* specifies an absolute path name, then the qsub will supply the name of the host on which it is executing for the hostname. • **hostname:path_name**where *path_name* specifies an absolute path name, the path will be used as specified. If the `-o` option is not specified, the default file name for the standard output stream will be used. The default name has the following form: • **job_name.osequence_number** – where *job_name* is the name of the job (see the **-n** name option) and *sequence_number* is the job number assigned when the job is submitted.
-p	priority	Defines the priority of the job. The priority argument must be a integer between -1024 and +1023 inclusive. The default is no priority which is equivalent to a priority of zero.
-P	user [:group]	Allows a root user or manager to submit a job as another user. TORQUE treats proxy jobs as though the jobs were submitted by the supplied username. This feature is available in TORQUE 2.4.7 and later, however, TORQUE 2.4.7 does not have the ability to supply the `[:group]` option; it is available in TORQUE 2.4.8 and later.

Option	Name	Description
-q	destination	Defines the destination of the job. The destination names a queue, a server, or a queue at a server. The qsub command will submit the script to the server defined by the destination argument. If the destination is a routing queue, the job may be routed by the server to a new destination. If the -q option is not specified, the qsub command will submit the script to the default server. (See **Environment variables** on page 203 and the PBS ERS section 2.7.4, "Default Server".) If the -q option is specified, it is in one of the following three forms: • queue • @server • queue@server If the destination argument names a queue and does not name a server, the job will be submitted to the named queue at the default server. If the destination argument names a server and does not name a queue, the job will be submitted to the default queue at the named server. If the destination argument names both a queue and a server, the job will be submitted to the named queue at the named server.
-r	y/n	Declares whether the job is rerunable (see the **qrerun** command). The option argument is a single character, either y or n. If the argument is "y", the job is rerunable. If the argument is "n", the job is not rerunable. The default value is y, rerunable.

-S	path_list	Declares the path to the desires shell for this job.
		`qsub script.sh -S /bin/tcsh`
		If the shell path is different on different compute nodes, use the following syntax:
		`path[@host][,path[@host],...]`
		`qsub script.sh -S /bin/tcsh@node1,/usr/bin/tcsh@node2`
		Only one path may be specified for any host named. Only one path may be specified without the corresponding host name. The path selected will be the one with the host name that matched the name of the execution host. If no matching host is found, then the path specified without a host will be selected, if present.
		If the -S option is not specified, the option argument is the null string, or no entry from the path_list is selected, the execution will use the users login shell on the execution host.

Option	Name	Description
-t	array_ request	Specifies the task ids of a job array. Single task arrays are allowed. The array_request argument is an integer id or a range of integers. Multiple ids or id ranges can be combined in a comma delimted list. Examples: `-t 1-100 or -t 1,10, 50-100` An optional **slot limit** can be specified to limit the amount of jobs that can run concurrently in the job array. The default value is unlimited. The slot limit must be the last thing specified in the array_request and is delimited from the array by a percent sign (**%**). `qsub script.sh -t 0-299%5` This sets the slot limit to 5. Only 5 jobs from this array can run at the same time. You can use **qalter** to modify slot limits on an array. The server parameter **max_slot_ limit** can be used to set a global slot limit policy.
-u	user_list	Defines the user name under which the job is to run on the execution system. The user_list argument is of the form: `user[@host][,user[@host],...]` Only one user name may be given per specified host. Only one of the user specifications may be supplied without the corresponding host specification. That user name will used for execution on any host not named in the argument list. If unset, the user list defaults to the user who is running qsub.
-v	variable_ list	Expands the list of environment variables that are exported to the job. In addition to the variables described in the "Description" section above, variable_list names environment variables from the qsub command environment which are made available to the job when it executes. The variable_list is a comma separated list of strings of the form `variable` or `variable=value`. These variables and their values are passed to the job.
-V	---	Declares that all environment variables in the qsub commands environment are to be exported to the batch job.

Option	Name	Description

-W	additional_ attributes	The `-W` option allows for the specification of additional job attributes. The general syntax of `-W` is in the form:
		`-W attr_name=attr_value[,attr_name=attr_value...]`
		If white space occurs anywhere within the option argument string or the equal sign, "**=**", occurs within an attribute_value string, then the string must be enclosed with either single or double quote marks.
		PBS currently supports the following attributes within the `-W` option:
		• **depend=dependency_list** – Defines the dependency between this and other jobs. The dependency_list is in the form:
		`type[:argument[:argument...][,type:argument...]`
		The argument is either a numeric count or a PBS job id according to type. If argument is a count, it must be greater than 0. If it is a job id and not fully specified in the form `seq_number.server.name`, it will be expanded according to the default server rules which apply to job IDs on most commands. If argument is null (the preceding colon need not be specified), the dependency of the corresponding type is cleared (unset). For more information, see **depend=dependency_list valid dependencies** on page 200.
		• **group_list=g_list** – Defines the group name under which the job is to run on the execution system. The g_list argument is of the form:
		`group[@host][,group[@host],...]`
		Only one group name may be given per specified host. Only one of the group specifications may be supplied without the corresponding host specification. That group name will used for execution on any host not named in the argument list. If not set, the group_list defaults to the primary group of the user under which the job will be run.
		• **interactive=true** – If the interactive attribute is specified, the job is an interactive job. The **-I** option is a alternative method of specifying this attribute.
		• **job_radix=*<int>*** – To be used with parallel jobs. It directs the Mother Superior of the job to create a distribution radix of size *<int>* between sisters. See **Managing multi-node jobs** on page 37.
		• **stagein=file_list**
		• **stageout=file_list** – Specifies which files are staged (copied) in before job start or staged out after the job completes execution. On completion of the job, all staged-in and staged-out files are removed from the execution system. The file_ list is in the form:
		`local_file@hostname:remote_file[,...]`
		regardless of the direction of the copy. The name local_file is the name of the file on the system where the job executed. It may be an absolute path or relative to the home directory of the user. The name remote_file is the destination name on the host specified by hostname. The name may be absolute or relative to the users home directory on the destination host. The use of wildcards in the file name is not recommended. The file names map to a remote copy program (rcp) call on the execution system in the follow manner:
		◦ For stagein: `rcp hostname:remote_file local_file`

Option	Name	Description

		○ For stageout: `rcp local_file hostname:remote_file`
		Data staging examples:
		`-W stagein=/tmp/input.txt@headnode:/home/user/input.txt`
		`-W stageout=/tmp/output.txt@headnode:/home/user/output.txt`
		If TORQUE has been compiled with wordexp support, then variables can be used in the specified paths. Currently only `$PBS_JOBID`, `$HOME`, and `$TMPDIR` are supported for stagein.
		• **umask=XXX** – Sets umask used to create stdout and stderr spool files in pbs_mom spool directory. Values starting with 0 are treated as octal values, otherwise the value is treated as a decimal umask value.

-x	---	By default, if you submit an interactive job with a script, the script will be parsed for PBS directives but the rest of the script will be ignored since it's an interactive job. The `-x` option allows the script to be executed in the interactive job and then the job completes. For example:
		`script.sh`
		`#!/bin/bash`
		`ls`
		`---end script---`
		`qsub -I script.sh`
		`qsub: waiting for job 5.napali to start`
		`dbeer@napali:#`
		`<displays the contents of the directory, because of the ls command>`
		`qsub: job 5.napali completed`

-X	---	Enables X11 forwarding. The DISPLAY environment variable must be set.
-z	---	Directs that the qsub command is not to write the job identifier assigned to the job to the commands standard output.

depend=dependency_list valid dependencies

Dependency	Description
`synccount:count`	This job is the first in a set of jobs to be executed at the same time. Count is the number of additional jobs in the set.
`syncwith:jobid`	This job is an additional member of a set of jobs to be executed at the same time. In the above and following dependency types, jobid is the job identifier of the first job in the set.

Dependency	Description
`after:jobid[:jobid...]`	This job may be scheduled for execution at any point after jobs jobid have started execution.
`afterok:jobid[:jobid...]`	This job may be scheduled for execution only after jobs jobid have terminated with no errors. See the csh warning under **Extended description** on page 204.
`afternotok:jobid[:jobid...]`	This job may be scheduled for execution only after jobs jobid have terminated with errors. See the csh warning under **Extended description** on page 204.
`afterany:jobid[:jobid...]`	This job may be scheduled for execution after jobs jobid have terminated, with or without errors.
`on:count`	This job may be scheduled for execution after count dependencies on other jobs have been satisfied. This form is used in conjunction with one of the "before" forms (see below).
`before:jobid[:jobid...]`	When this job has begun execution, then jobs jobid... may begin.
`beforeok:jobid[:jobid...]`	If this job terminates execution without errors, then jobs jobid... may begin. See the csh warning under **Extended description** on page 204.
`beforenotok:jobid[:jobid...]`	If this job terminates execution with errors, then jobs jobid... may begin. See the csh warning under **Extended description** on page 204.
`beforeany:jobid[:jobid...]`	When this job terminates execution, jobs jobid... may begin. If any of the before forms are used, the jobs referenced by jobid must have been submitted with a dependency type of on. If any of the before forms are used, the jobs referenced by jobid must have the same owner as the job being submitted. Otherwise, the dependency is ignored.

Array dependencies make a job depend on an array or part of an array. If no count is given, then the entire array is assumed. For examples, see **Dependency examples** on page 202.

`afterstartarray:arrayid[count]`	After this many jobs have started from arrayid, this job may start.

Dependency	Description
`afterokarray:arrayid[count]`	This job may be scheduled for execution only after jobs in arrayid have terminated with no errors.
`afternotokarray:arrayid[count]`	This job may be scheduled for execution only after jobs in arrayid have terminated with errors.
`afteranyarray:arrayid[count]`	This job may be scheduled for execution after jobs in arrayid have terminated, with or without errors.
`beforestartarray:arrayid[count]`	Before this many jobs have started from arrayid, this job may start.
`beforeokarray:arrayid[count]`	If this job terminates execution without errors, then jobs in arrayid may begin.
`beforenotokarray:arrayid[count]`	If this job terminates execution with errors, then jobs in arrayid may begin.
`beforeanyarray:arrayid[count]`	When this job terminates execution, jobs in arrayid may begin. If any of the before forms are used, the jobs referenced by arrayid must have been submitted with a dependency type of on. If any of the before forms are used, the jobs referenced by arrayid must have the same owner as the job being submitted. Otherwise, the dependency is ignored.

Error processing of the existence, state, or condition of he job on which the newly submitted job is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the new job will be deleted by the server. Mail will be sent to the job submitter stating the error.

Dependency examples

```
qsub -W depend=afterok:123.big.iron.com /tmp/script
```

```
qsub -W depend=before:234.hunk1.com:235.hunk1.com
```

```
/tmp/script
```

```
qsub script.sh -W depend=afterokarray:427[]
```

(This assumes every job in array 427 has to finish successfully for the dependency to be satisfied.)

```
qsub script.sh -W depend=afterokarray:427[][5]
```

(This means that 5 of the jobs in array 427 have to successfully finish in order for the dependency to be satisfied.)

Operands

The qsub command accepts a script operand that is the path to the script of the job. If the path is relative, it will be expanded relative to the working directory of the qsub command.

If the script operand is not provided or the operand is the single character "**-**", the qsub command reads the script from standard input. When the script is being read from Standard Input, qsub will copy the file to a temporary file. This temporary file is passed to the library interface routine pbs_submit. The temporary file is removed by qsub after pbs_submit returns or upon the receipt of a signal which would cause qsub to terminate.

Standard input

The qsub command reads the script for the job from standard input if the script operand is missing or is the single character "**-**".

Input files

The script file is read by the qsub command. qsub acts upon any directives found in the script.

When the job is created, a copy of the script file is made and that copy cannot be modified.

Standard output

Unless the **-z** option is set, the job identifier assigned to the job will be written to standard output if the job is successfully created.

Standard error

The qsub command will write a diagnostic message to standard error for each error occurrence.

Environment variables

The values of some or all of the variables in the qsub commands environment are exported with the job (see the **-v** and **-v** options).

The environment variable PBS_DEFAULT defines the name of the default server. Typically, it corresponds to the system name of the host on which the server is running. If PBS_DEFAULT is not set, the default is defined by an administrator established file.

The environment variable PBS_DPREFIX determines the prefix string which identifies directives in the script.

The environment variable PBS_CLIENTRETRY defines the maximum number of seconds qsub will block (see the **-b** option). Despite the name, currently qsub is the only client that supports this option.

torque.cfg

The `torque.cfg` file, located in PBS_SERVER_HOME (`/var/spool/torque` by default) controls the behavior of the qsub command. This file contains a list of parameters and values separated by whitespace.

- **QSUBSLEEP** – takes an integer operand which specifies time to sleep when running qsub command. Used to prevent users from overwhelming the scheduler.
- **SUBMITFILTER** – specifies the path to the submit filter used to pre-process job submission. The default path is libexecdir/qsub_filter, which falls back to /usr/local/sbin/torque_submitfilter for backwards compatibility. This torque.cfg parameter overrides this default.
- **SERVERHOST**
- **QSUBHOST**
- **QSUBSENDUID**
- **XAUTHPATH**
- **CLIENTRETRY**
- **VALIDATEGROUP**
- **DEFAULTCKPT**
- **VALIDATEPATH**
- **RERUNNABLEBYDEFAULT**

For example:

```
QSUBSLEEP 2
```

```
RERUNNABLEBYDEFAULT false
```

Extended description

Script Processing:

A job script may consist of PBS directives, comments and executable statements. A PBS directive provides a way of specifying job attributes in addition to the command line options. For example:

```
:
#PBS -N Job_name
#PBS -l walltime=10:30,mem=320kb
#PBS -m be
#
step1 arg1 arg2
step2 arg3 arg4
```

The qsub command scans the lines of the script file for directives. An initial line in the script that begins with the characters "**#!**" or the character "**:**" will be ignored and scanning will start with the next line. Scanning will continue until the first executable line, that is a line that is not blank, not a directive line, nor a line whose first non white space character is "#". If directives occur on subsequent lines, they will be ignored.

A line in the script file will be processed as a directive to qsub if and only if the string of characters starting with the first non white space character on the line and of the same length as the directive prefix matches the directive prefix.

The remainder of the directive line consists of the options to qsub in the same syntax as they appear on the command line. The option character is to be preceded with the "**-**" character.

If an option is present in both a directive and on the command line, that option and its argument, if any, will be ignored in the directive. The command line takes precedence.

If an option is present in a directive and not on the command line, that option and its argument, if any, will be processed as if it had occurred on the command line.

The directive prefix string will be determined in order of preference from:

- The value of the **-c** option argument if the option is specified on the command line.
- The value of the environment variable PBS_DPREFIX if it is defined.
- The four character string #PBS.

If the **-c** option is found in a directive in the script file, it will be ignored.

User Authorization:

When the user submits a job from a system other than the one on which the PBS Server is running, the name under which the job is to be executed is selected according to the rules listed under the **-u** option. The user submitting the job must be authorized to run the job under the execution user name. This authorization is provided if:

- The host on which qsub is run is trusted by the execution host (see `/etc/hosts.equiv`).
- The execution user has an `.rhosts` file naming the submitting user on the submitting host.

C-Shell .logout File:

The following warning applies for users of the c-shell, csh. If the job is executed under the csh and a `.logout` file exists in the home directory in which the job executes, the exit status of the job is that of the `.logout` script, not the job script. This may impact any inter-job dependencies. To preserve the job exit status, either remove the `.logout` file or place the following line as the first line in the `.logout` file:

```
set EXITVAL = $status
```

and the following line as the last executable line in `.logout`:

```
exit $EXITVAL
```

Interactive Jobs:

If the **-I** option is specified on the command line or in a script directive, or if the "interactive" job attribute declared true via the **-W** option, `-W interactive=true`, either on the command line or in a script directive, the job is an interactive job. The script will be processed for directives, but will not be included with the job. When the job begins execution, all input to the job is from the terminal session in which qsub is running.

When an interactive job is submitted, the qsub command will not terminate when the job is submitted. qsub will remain running until the job terminates, is aborted, or the user interrupts qsub with an SIGINT (the control-C key). If qsub is interrupted prior to job start, it will query if the user wishes to exit. If the user response "yes", qsub exits and the job is aborted.

One the interactive job has started execution, input to and output from the job pass through qsub. Keyboard generated interrupts are passed to the job. Lines entered that begin with the tilde (~) character and contain special sequences are escaped by qsub. The recognized escape sequences are:

Sequence	Description
~.	qsub terminates execution. The batch job is also terminated.
~susp	Suspend the qsub program if running under the C shell. "susp" is the suspend character (usually CNTL-Z).
~asusp	Suspend the input half of qsub (terminal to job), but allow output to continue to be displayed. Only works under the C shell. "asusp" is the auxiliary suspend character, usually CNTL-Y.

Exit status

Upon successful processing, the qsub exit status will be a value of zero.

If the qsub command fails, the command exits with a value greater than zero.

Related topics

- qalter(1B)
- qdel(1B)
- qhold(1B)
- qrls(1B)
- qsig(1B)
- qstat(1B)
- pbs_server(8B)

Non-Adaptive Computing topics

- pbs_connect(3B)
- pbs_job_attributes(7B)
- pbs_queue_attributes(7B)
- pbs_resources_irix5(7B)
- pbs_resources_sp2(7B)
- pbs_resources_sunos4(7B)
- pbs_resources_unicos8(7B)
- pbs_server_attributes(7B)
- qselect(1B)
- qmove(1B)

- qmsg(1B)
- qrerun(1B)

qterm

Terminate processing by a PBS batch server.

Synopsis

```
qterm [-t type] [server...]
```

Description

The qterm command terminates a batch server. When a server receives a terminate command, the server will go into the "Terminating" state. No new jobs will be allowed to be started into execution nor enqueued into the server. The impact on jobs currently being run by the server depends

In order to execute qterm, the user must have PBS Operation or Manager privileges.

Options

Option	Name	Decription
-t	type	Specifies the type of shut down. The types are: • **immediate** – All running jobs are to immediately stop execution. If checkpointing is supported, running jobs that can be checkpointed are checkpointed, terminated, and requeued. If checkpoint is not supported or the job cannot be checkpointed, running jobs are requeued if the rerunable attribute is true. Otherwise, jobs are killed. • **delay** – If checkpointing is supported, running jobs that can be checkpointed are checkpointed, terminated, and requeued. If a job cannot be checkpointed, but can be rerun, the job is terminated and requeued. Otherwise, running jobs are allowed to continue to run. Note, the operator or administrator may use the **qrerun** and **qdel** commands to remove running jobs. • **quick** – This is the default action if the `-t` option is not specified. This option is used when you wish that running jobs be left running when the server shuts down. The server will cleanly shutdown and can be restarted when desired. Upon restart of the server, jobs that continue to run are shown as running; jobs that terminated during the server's absence will be placed into the exiting state.

Operands

The server operand specifies which servers are to shutdown. If no servers are given, then the default server will be terminated.

Standard error

The qterm command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qterm command, the exit status will be a value of zero.

If the qterm command fails to process any operand, the command exits with a value greater than zero.

Related topics(non-Adaptive Computing topics)

- pbs_server(8B)
- qmgr(8B)
- pbs_resources_aix4(7B)
- pbs_resources_irix5(7B)
- pbs_resources_sp2(7B)
- pbs_resources_sunos4(7B)
- pbs_resources_unicos8(7B)

B.1 Server parameters

TORQUE server parameters are specified using the **qmgr** command. The `set` subcommand is used to modify the **server** object. For example:

```
> qmgr -c 'set server default_queue=batch'
```

Parameters

acl_hosts	
Format	<HOST>[,<HOST>]... **or** <HOST>[range] **or** <HOST*> where the asterisk (*) can appear anywhere in the host name
Default	(Only the host running pbs_server may submit jobs.)

Description	Specifies a list of hosts from which jobs may be submitted. Hosts in the server nodes file located at `$TORQUE/server_priv/nodes` cannot be added to the list using the acl_hosts parameter (see **Server node file configuration** on page 24). To submit batch or interactive jobs (see **Server configuration** on page 19) through hosts that are specified in the server nodes file, use the **submit_hosts** parameter.
	`Qmgr: set queue batch acl_hosts = "hostA,hostB"`
	`Qmgr: set queue batch acl_hosts += "hostE,hostF,hostG"`
	
	In version 2.5 and later, the wildcard (*) character can appear anywhere in the host name, and ranges are supported; these specifications also work for managers and operators.
	`Qmgr: set server acl_hosts = "galaxy*.tom.org"`
	`Qmgr: set server acl_hosts += "galaxy[0-50].tom.org"`
	`Qmgr: set server managers+=tom@galaxy[0-50].tom.org`

acl_host_enable	
Format	<BOOLEAN>
Default	**FALSE**
Description	Specifies if the **acl_hosts** value is enabled.

acl_logic_or	
Format	<BOOLEAN>
Default	**FALSE**
Description	Specifies if user and group queue ACL's should be logically AND'd or logically OR'd.

acl_roots	
Format	<username>@<domain>
Default	---
Description	Specifies which root users are allowed to submit and run jobs.

allow_node_submit	
Format	<BOOLEAN>
Default	**FALSE**
Description	Specifies if users can submit jobs directly from any trusted compute host directly or from within batch jobs (see **Configuring job submission hosts** on page 19).

allow_proxy_user	
Format	<BOOLEAN>
Default	**FALSE**
Description	Specifies if users can proxy from one user to another. Proxy requests will be either validated by `ruserok()` or by the scheduler (see **Job submission** on page 35).

auto_node_np	
Format	<BOOLEAN>
Default	**DISABLED**

auto_node_np

Description	Automatically configures a node's np (number of processors) value based on the ncpus value from the status update. Requires full manager privilege to set or alter.

checkpoint_defaults

Format	<STRING>
Default	---
Description	Specifies for a queue the default checkpoint values for a job that does not have checkpointing specified. The checkpoint_defaults parameter only takes effect on execution queues. `set queue batch checkpoint_defaults="enabled, periodic, interval=5"`

clone_batch_delay

Format	<INTEGER>
Default	1
Description	Specifies the delay (in seconds) between clone batches (see **clone_batch_size**).

clone_batch_size

Format	<INTEGER>
Default	256
Description	Job arrays are created in batches of size *X*. *X* jobs are created, and after the **clone_batch_delay**, *X* more are created. This repeats until all are created.

cray_enabled

Format	<BOOLEAN>
Default	**FALSE**
Description	Specifies that this instance of pbs_server has Cray hardware that reports to it. See Installation Notes for Moab and TORQUE for Cray in the Moab Workload Manager documentation.

default_queue

Format	<STRING>
Default	---
Description	Indicates the queue to assign to a job if no queue is explicitly specified by the submitter.

disable_server_id_check

Format	<BOOLEAN>
Default	**FALSE**
Description	Makes it so the user for the job doesn't have to exist on the server. The user must still exist on all the compute nodes or the job will fail when it tries to execute. If you have **disable_server_id_check** set to `TRUE`, a user could request a group to which they do not belong. Setting VALIDATEGROUP to `TRUE` in the `torque.cfg` file prevents such a scenario (see **"torque.cfg" configuration file** on page 271).

display_job_server_suffix

Format	<BOOLEAN>
Default	**TRUE**
Description	If this parameter is set to `TRUE`, TORQUE will display both the job ID and the host name. If it is set to `FALSE`, only the job ID will be displayed.

interactive_jobs_can_run

Format	<BOOLEAN>
Default	**FALSE**
Description	By default, interactive jobs run from the login node that they submitted from. When set to `TRUE`, interactive jobs may run on login nodes other than the one where the jobs were submitted to.

job_force_cancel_time

Format	<INTEGER>
Default	Disabled
Description	If a job has been deleted and is still in the system after *x* seconds, the job will be purged from the system. This is mostly useful when a job is running on a large number of nodes and one node goes down. The job cannot be deleted because the MOM cannot be contacted. The qdel fails and none of the other nodes can be reused. This parameter can used to remedy such situations.

job_log_file_max_size

Format	<INTEGER>
Default	---
Description	This specifies a soft limit (in kilobytes) for the job log's maximum size. The file size is checked every five minutes and if the **current day** file size is greater than or equal to this value, it is rolled from `<filename>` to `<filename.1>` and a new empty log is opened. If the current day file size exceeds the maximum size a second time, the <filename.1> log file is rolled to `<filename.2>`, the current log is rolled to `<filename.1>`, and a new empty log is opened. Each new log causes all other logs to roll to an extension that is one greater than its current number. Any value less than 0 is ignored by pbs_server (meaning the log will not be rolled).

job_log_file_roll_depth

Format	<INTEGER>
Default	---
Description	This sets the maximum number of new log files that are kept in a day if the **job_log_file_max_size** parameter is set. For example, if the roll depth is set to 3, no file can roll higher than `<filename.3>`. If a file is already at the specified depth, such as `<filename.3>`, the file is deleted so it can be replaced by the incoming file roll, `<filename.2>`.

job_log_keep_days

Format	<INTEGER>
Default	---

job_log_keep_days

Description	This maintains logs for the number of days designated. If set to 4, any log file older than 4 days old is deleted.

job_nanny

Format	<BOOLEAN>
Default	**FALSE**
Description	Enables the experimental "job deletion nanny" feature. All job cancels will create a repeating task that will resend KILL signals if the initial job cancel failed. Further job cancels will be rejected with the message "job cancel in progress." This is useful for temporary failures with a job's execution node during a job delete request.

job_stat_rate

Format	<INTEGER>
Default	45 (30 in TORQUE 1.2.0p5 and earlier)
Description	Specifies the maximum age of MOM level job data which is allowed when servicing a qstat request. If data is older than this value, the pbs_server daemon will contact the MOMs with stale data to request an update. For large systems, this value should be increased to 5 minutes or higher.

job_start_timeout

Format	<INTEGER>
Default	---
Description	Specifies the pbs_server to pbs_mom TCP socket timeout in seconds that is used when the pbs_server sends a job start to the pbs_mom. It is useful when the MOM has extra overhead involved in starting jobs. If not specified, then the **tcp_timeout** parameter is used.

lock_file

Format	<STRING>

lock_file	
Default	`torque/server_priv/server.lock`
Description	Specifies the name and location of the lock file used to determine which high availability server should be active. If a full path is specified, it is used verbatim by TORQUE. If a relative path is specified, TORQUE will prefix it with `torque/server_priv`.

lock_file_update_time	
Format	<INTEGER>
Default	3
Description	Specifies how often (in seconds) the thread will update the lockfile.

lock_file_check_time	
Format	<INTEGER>
Default	9
Description	Specifies how often (in seconds) a high availability server will check to see if it should become active.

log_events	
Format	Bitmap
Default	---

log_events

Description

By default, all events are logged. However, you can customize things so that only certain events show up in the log file. These are the bitmaps for the different kinds of logs:

```
#define PBSEVENT_ERROR 0x0001 /* internal errors */
#define PBSEVENT_SYSTEM 0x0002 /* system (server) events */
#define PBSEVENT_ADMIN 0x0004 /* admin events */
#define PBSEVENT_JOB 0x0008 /* job related events */
#define PBSEVENT_JOB_USAGE 0x0010 /* End of Job accounting */
#define PBSEVENT_SECURITY 0x0020 /* security violation events */
#define PBSEVENT_SCHED 0x0040 /* scheduler events */
#define PBSEVENT_DEBUG 0x0080 /* common debug messages */
#define PBSEVENT_DEBUG2 0x0100 /* less needed debug messages */
#define PBSEVENT_FORCE 0x8000 /* set to force a message */
```

If you want to log only error, system, and job information, use qmgr to set log_events to 11:

```
set server log_events = 11
```

log_file_max_size

Format	<INTEGER>
Default	0
Description	Specifies a soft limit, in kilobytes, for the server's log file. The filesize is checked every 5 minutes, and if the **current day** filesize is greater than or equal to this value then it will be rolled from *X* to *X.1* and a new empty log will be opened. Any value less than or equal to 0 will be ignored by pbs_server (the log will not be rolled).

log_file_roll_depth

Format	<INTEGER>
Default	1
Description	Controls how deep the current day log files will be rolled, if log_file_max_size is set, before they are deleted.

log_keep_days

Format	<INTEGER>

log_keep_days	
Default	0
Description	Specifies how long (in days) a server or MOM log should be kept.

log_level	
Format	<INTEGER>
Default	0
Description	Specifies the pbs_server logging verbosity. Maximum value is 7.

mail_body_fmt	
Format	A printf-like format string
Default	PBS Job Id: %i Job Name: %j Exec host: %h %m %d
Description	Override the default format for the body of outgoing mail messages. A number of printf-like format specifiers and escape sequences can be used: \n new line \t tab \\ backslash \' single quote \" double quote %d details concerning the message %h PBS host name %i PBS job identifier %j PBS job name %m long reason for message %r short reason for message %% a single %

mail_domain	
Format	<STRING>
Default	---

mail_domain

Description	Override the default domain for outgoing mail messages. If set, emails will be addressed to <user>@<hostdomain>. If unset, the job's Job_Owner attribute will be used. If set to `never`, TORQUE will never send emails.

mail_subject_fmt

Format	A printf-like format string
Default	PBS JOB %i
Description	Override the default format for the subject of outgoing mail messages. A number of printf-like format specifiers and escape sequences can be used: \n new line \t tab \\ backslash \' single quote \" double quote %d details concerning the message %h PBS host name %i PBS job identifier %j PBS job name %m long reason for message %r short reason for message %% a single %

managers

Format	`<user>@<host.sub.domain>[,<user>@<host.sub.domain>...]`
Default	root@localhost
Description	List of users granted batch administrator privileges. The host, sub-domain, or domain name may be wildcarded by the use of an asterisk character (*). Requires full manager privilege to set or alter.

max_job_array_size

Format	<INTEGER>
Default	Unlimited
Description	Sets the maximum number of jobs that can be in a single job array.

max_slot_limit

Format	<INTEGER>
Default	Unlimited
Description	This is the maximum number of jobs that can run concurrently in any job array. Slot limits can be applied at submission time with **qsub**, or it can be modifed with **qalter**. `qmgr -c 'set server max_slot_limit=10'` No array can request a slot limit greater than 10. Any array that does not request a slot limit receives a slot limit of 10. Using the example above, slot requests greater than 10 are rejected with the message: "Requested slot limit is too large, limit is 10."

max_threads

Format	<INTEGER>
Default	200
Description	This is the maximum number of threads that should exist in the threadpool at any time.

max_user_queuable

Format	<INTEGER>
Default	Unlimited
Description	When set, `max_user_queuable` places a system-wide limit on the amount of jobs that an individual user can queue. `qmgr -c 'set server max_user_queuable=500'`

min_threads

Format	<INTEGER>
Default	5
Description	This is the minimum number of threads that should exist in the threadpool at any time.

moab_array_compatible	
Format	<BOOLEAN>
Default	TRUE
Description	This parameter places a hold on jobs that exceed the slot limit in a job array. When one of the active jobs is completed or deleted, one of the held jobs goes to a queued state.

mom_job_sync	
Format	<BOOLEAN>
Default	**TRUE**
Description	Specifies that the pbs_server will synchronize its view of the job queue and resource allocation with compute nodes as they come online. If a job exists on a compute node in a pre-execution or corrupt state, it will be automatically cleaned up and purged. (Enabled by default in TORQUE 2.2.0 and higher.) Jobs that are no longer reported by the mother superior are automatically purged by pbs_server. Jobs that pbs_server instructs the mom to cancel have their processes killed in addition to being deleted (instead of leaving them running as in versions of TORQUE prior to 4.1.1).

node_check_rate	
Format	<INTEGER>
Default	600
Description	Specifies the minimum duration (in seconds) that a node can be unresponsive to server queries before being marked down by the pbs_server daemon.

node_pack	
Format	<BOOLEAN>
Default	---

node_pack	
Description	Controls how multiple processor nodes are allocated to jobs. If this attribute is set to true, jobs will be assigned to the multiple processor nodes with the fewest free processors. This packs jobs into the fewest possible nodes leaving multiple processor nodes free for jobs which need many processors on a node. If set to false, jobs will be scattered across nodes reducing conflicts over memory between jobs. If unset, the jobs are packed on nodes in the order that the nodes are declared to the server (in the nodes file). Default value: unset - assigned to nodes as nodes in order that were declared.

node_ping_rate	
Format	<INTEGER>
Default	300
Description	Specifies the maximum interval (in seconds) between successive "pings" sent from the pbs_server daemon to the pbs_mom daemon to determine node/daemon health.

no_mail_force	
Format	<BOOLEAN>
Default	**FALSE**
Description	When set to TRUE, eliminates all e-mails when mail_options (see **qsub** on page 189) is set to "n". The job owner won't receive e-mails when a job is deleted by a different user or a job failure occurs. If no_mail_force is unset or is FALSE, then the job owner receives e-mails when a job is deleted by a different user or a job failure occurs.

np_default	
Format	<INTEGER>
Default	---
Description	Allows the administrator to unify the number of processors (np) on all nodes. The value can be dynamically changed. A value of 0 tells pbs_server to use the value of np found in the nodes file. The maximum value is 32767.

operators	
Format	<user>@<host.sub.domain>[,<user>@<host.sub.domain>...]
Default	root@localhost
Description	List of users granted batch operator privileges. Requires full manager privilege to set or alter.

poll_jobs	
Format	<BOOLEAN>
Default	**TRUE** (FALSE in TORQUE 1.2.0p5 and earlier)
Description	If set to TRUE, pbs_server will poll job info from MOMs over time and will not block on handling requests which require this job information. If set to FALSE, no polling will occur and if requested job information is stale, pbs_server may block while it attempts to update this information. For large systems, this value should be set to TRUE.

query_other_jobs	
Format	<BOOLEAN>
Default	**FALSE**
Description	Specifies whether or not non-admin users may view jobs they do not own.

record_job_info	
Format	<BOOLEAN>
Default	**FALSE**
Description	This must be set to true in order for job logging to be enabled.

record_job_script	
Format	<BOOLEAN>

record_job_script	
Default	**FALSE**
Description	If set to TRUE, this adds the contents of the script executed by a job to the log.

resources_available	
Format	<STRING>
Default	---
Description	Allows overriding of detected resource quantity limits (see **Assigning queue resource limits** on page 84). pbs_server must be restarted for changes to take effect. Also, resources_available is constrained by the smallest of queue.resources_available and the server.resources_available.

submit_hosts	
Format	"<HOSTNAME>[,<HOSTNAME>]..."
Default	---
Description	Indicates which hosts included in the server nodes file located at $TORQUE/server_priv/nodes (see **Server node file configuration** on page 24) can submit batch or interactive jobs (see **Configuring job submission hosts** on page 19). For more information on adding hosts that are not included in the first nodes file, see the **acl_hosts** parameter.

tcp_timeout	
Format	<INTEGER>
Default	300
Description	Specifies the timeout for idle TCP connections. If no communication is received by the server on the connection after the timeout, the server closes the connection. There is an exception for connections made to the server on port 15001 (default); timeout events are ignored on the server for such connections established by a client utility or scheduler.Responsibility rests with the client to close the connection first. (See **Large cluster considerations** on page 251 for additional information.)

thread_idle_seconds	
Format	<INTEGER>
Default	-1
Description	This is the number of seconds a thread can be idle in the threadpool before it is deleted. If threads should not be deleted, set to `-1` (which is the default). TORQUE will always maintain at least **min_threads** number of threads, even if all are idle.

C.1 Node manager (MOM) configuration

Under TORQUE, MOM configuration is accomplished using the `mom_priv/config` file located in the PBS directory on each execution server.

For details, see these topics:

- **Parameters** on page 225
- **Node features and generic consumable resource specification** on page 239
- **Command-line arguments** on page 240

Related topics

- **Commands overview** on page 135
- **Prologue and epilogue scripts** on page 257

C.1.1 Parameters

arch	
Format	<STRING>
Description	Specifies the architecture of the local machine. This information is used by the scheduler only.
Example	`arch ia64`

$alias_server_name	
Format	<STRING>

$alias_server_name

Description	(*Applicable in version 2.5.0 and later.*) Allows the MOM to accept an additional pbs_server host name as a trusted address. This feature was added to overcome a problem with UDP and RPP where alias IP addresses are used on a server. With alias IP addresses a UDP packet can be sent to the alias address but the UDP reply packet will come back on the primary IP address. RPP matches addresses from its connection table to incoming packets. If the addresses do not match an entry in the RPP table, the packet is dropped. This feature allows an additional address for the server to be added to the table so legitimate packets are not dropped.
Example	`$alias_server_name node01`

$reject_job_submission

Format	<BOOLEAN>
Description	If set to **TRUE**, jobs will be rejected and the user will receive the message, "Jobs cannot be run on mom %s." Default is FALSE.
Example	`$reject_job_submission job01`

$attempt_to_make_dir

Format	<BOOLEAN>
Description	When set to **TRUE**, specifies that you want TORQUE to attempt to create the output directories for jobs if they do not already exist. Default is FALSE. TORQUE uses this parameter to make the directory as the *user* and not as *root*. TORQUE will create the directory (or directories) ONLY if the user has permissions to do so.
Example	`$attempt_to_make_dir true`

$clienthost

Format	<STRING>

$clienthost	
Description	Specifies the machine running pbs_server. This parameter is deprecated. Use **$pbsserver**.
Example	`$clienthost node01.teracluster.org`

$check_poll_time	
Format	<STRING>
Description	Amount of time between checking running jobs, polling jobs, and trying to resend obituaries for jobs that haven't sent successfully. Default is 45 seconds.
Example	`$check_poll_time 90`

$configversion	
Format	<STRING>
Description	Specifies the version of the config file data.
Example	`$configversion 113`

$cputmult	
Format	<FLOAT>
Description	CPU time multiplier. If set to 0.0, MOM level cputime enforcement is disabled.
Example	`$cputmult 2.2`

$ext_pwd_retry	
Format	<INTEGER>

$ext_pwd_retry	
Description	(*Available in TORQUE 2.5.10, 3.0.4, and later.*) Specifies the number of times to retry checking the password. Useful in cases where external password validation is used,such as with LDAP. The default value is 3 retries.
Example	`$ext_pwd_retry = 5`

$ideal_load	
Format	<FLOAT>
Description	Ideal processor load.
Example	`$ideal_load 4.0`

$igncput	
Format	<BOOLEAN>
Description	Ignores limit violation pertaining to CPU time. Default is FALSE.
Example	`$igncput true`

$ignmem	
Format	<BOOLEAN>
Description	Ignores limit violations pertaining to physical memory. Default is FALSE.
Example	`$ignmem true`

$ignvmem	
Format	<BOOLEAN>
Description	Ignores limit violations pertaining to virtual memory. Default is FALSE.
Example	`$ignvmem true`

$ignwalltime	
Format	<BOOLEAN>
Description	Ignore walltime (do not enable mom based walltime limit enforcement).
Example	`$ignwalltime true`

$job_output_file_unmask	
Format	<STRING>
Description	Uses the specified umask when creating job output and error files. Values can be specified in base 8, 10, or 16; leading 0 implies octal and leading 0x or 0X hexadecimal. A value of "userdefault" will use the user's default umask. This parameter is in version 2.3.0 and later.
Example	`$job_output_file_umask 027`

$job_starter	
Format	<STRING>
Description	Specifies the fully qualified pathname of the job starter. If this parameter is specified, instead of executing the job command and job arguments directly, the MOM will execute the job starter, passing the job command and job arguments to it as its arguments. The job starter can be used to launch jobs within a desired environment.

Example	`$job_starter /var/torque/mom_priv/job_starter.sh`
	`> cat /var/torque/mom_priv/job_starter.sh`
	`#!/bin/bash`
	`export FOOHOME=/home/foo`
	`ulimit -n 314`
	`$*`

$log_directory	
Format	<STRING>
Description	Changes the log directory. Default is `TORQUE_HOME/mom_logs/`. TORQUE_HOME default is `/var/spool/torque/` but can be changed in the `./configure` script. The value is a string and should be the full path to the desired mom log directory.
Example	`$log_directory /opt/torque/mom_logs/`

$log_file_suffix	
Format	<STRING>
Description	Optional suffix to append to log file names. If %h is the suffix, pbs_mom appends the hostname for where the log files are stored if it knows it, otherwise it will append the hostname where the mom is running.
Example	`$log_file_suffix %h = 20100223.mybox` `$log_file_suffix foo = 20100223.foo`

$logevent	
Format	<STRING>
Description	Specifies a bitmap for event types to log.
Example	`$logevent 255`

$loglevel	
Format	<INTEGER>
Description	Specifies the verbosity of logging with higher numbers specifying more verbose logging. Values may range between 0 and 7.
Example	`$loglevel 4`

$log_file_max_size	
Format	<INTEGER>
Description	Soft limit for log file size in kilobytes. Checked every 5 minutes. If the log file is found to be greater than or equal to log_file_max_size the current log file will be moved from X to X.1 and a new empty file will be opened.
Example	`$log_file_max_size = 100`

$log_file_roll_depth	
Format	<INTEGER>
Description	Specifies how many times a log fill will be rolled before it is deleted.
Example	`$log_file_roll_depth = 7`

$log_keep_days	
Format	<INTEGER>
Description	Specifies how many days to keep log files. pbs_mom deletes log files older than the specified number of days. If not specified, pbs_mom won't delete log files based on their age.
Example	`$log_keep_days 10`

$max_conn_timeout_micro_sec	
Format	<INTEGER>
Description	Specifies how long pbs_mom should wait for a connection to be made. Default value is 10000 (.1 sec).
Example	`$max_conn_timeout_micro_sec 30000` This sets the connection timeout on the MOM to .3 seconds..

$max_load	
Format	<FLOAT>
Description	Maximum processor load.
Example	`$max_load 4.0`

$memory_pressure_duration	
Format	<INTEGER>

$memory_pressure_duration	
Description	(*Applicable in version 3.0 and later*.) Memory pressure duration sets a limit to the number of times the value of memory_pressure_threshold can be exceeded before a process is terminated. This can only be used with **$memory_pressure_threshold**.
Example	`$memory_pressure_duration 5`

$memory_pressure_threshold	
Format	<INTEGER>
Description	(*Applicable in version 3.0 and later*.) The memory_pressure of a cpuset provides a simple per-cpuset running average of the rate that the processes in a cpuset are attempting to free up in-use memory on the nodes of the cpuset to satisfy additional memory requests. The memory_pressure_threshold is an integer number used to compare against the reclaim rate provided by the memory_pressure file. If the threshold is exceeded and memory_pressure_duration is set, then the process terminates after exceeding the threshold by the number of times set in memory_pressure_duration. If memory_ pressure duration is not set, then a warning is logged and the process continues. Memory_pressure_ threshold is only valid with memory_pressure enabled in the root cpuset. To enable, log in as the super user and execute the command `echo 1 >> /dev/cpuset/memory_ pressure_enabled`. See the cpuset man page for more information concerning memory pressure.
Example	`$memory_pressure_threshold 1000`

$node_check_script	
Format	<STRING>
Description	Specifies the fully qualified pathname of the health check script to run (see **Compute node health check** on page 124 for more information).
Example	`$node_check_script /opt/batch_tools/nodecheck.pl`

$node_check_interval	
Format	<STRING>

$node_check_interval	
Description	Specifies the number of MOM intervals between subsequent executions of the specified health check. This value default to 1 indicating the check is run every mom interval (see **Compute node health check** on page 124 for more information). $node_check_interval has two special strings that can be set: • **jobstart** – makes the node health script run when a job is started. • **jobend** – makes the node health script run after each job has completed on a node.
Example	`$node_check_interval 5`

$nodefile_suffix	
Format	<STRING>
Description	Specifies the suffix to append to a host names to denote the data channel network adapter in a multihomed compute node.
Example	`$nodefile_suffix i` with the suffix of "i" and the control channel adapter with the name **node01**, the data channel would have a hostname of **node01i**.

$nospool_dir_list	
Format	<STRING>
Description	If this is configured, the job's output is spooled in the working directory of the job or the specified output directory. Specify the list in full paths, delimited by commas. If the job's working directory (or specified output directory) is in one of the paths in the list (or a subdirectory of one of the paths in the list), the job is spooled directly to the output location. $nospool_dir_list * is accepted. The user that submits the job must have write permission on the folder where the job is written, and read permission on the folder where the file is spooled. Alternatively, you can use the $spool_as_final_name parameter to force the job to spool directly to the final output. This should generally be used only when the job can run on the same machine as where the output file goes, or if there is a shared filesystem. If not, this parameter can slow down the system or fail to create the output file.
Example	`$nospool_dir_list /home/mike/jobs/,/var/tmp/spool/`

opsys	
Format	<STRING>
Description	Specifies the operating system of the local machine. This information is used by the scheduler only.
Example	`opsys RHEL3`

$pbsclient	
Format	<STRING>
Description	Specifies machines which the mom daemon will trust to run resource manager commands via **momctl**. This may include machines where monitors, schedulers, or admins require the use of this command.
Example	`$pbsclient node01.teracluster.org`

$pbsserver	
Format	<STRING>
Description	Specifies the machine running pbs_server. This parameter replaces the deprecated parameter **$clienthost**.
Example	`$pbsserver node01.teracluster.org`

$prologalarm	
Format	<INTEGER>
Description	Specifies maximum duration (in seconds) which the mom will wait for the job prologue or job epilogue to complete. This parameter defaults to 300 seconds (5 minutes).
Example	`$prologalarm 60`

$rcpcmd	
Format	<STRING>
Description	Specifies the full path and optional additional command line args to use to perform remote copies.
Example	`mom_priv/config:` `$rcpcmd /usr/local/bin/scp -i /etc/sshauth.dat`

$remote_reconfig	
Format	<STRING>
Description	Enables the ability to remotely reconfigure pbs_mom with a new config file. Default is disabled. This parameter accepts various forms of true, yes, and 1. For more information on how to reconfigure MOMs, see **momctl** `-r`.
Example	`$remote_reconfig true`

$reduce_prolog_checks	
Format	<STRING>
Description	If enabled, TORQUE will only check if the file is a regular file and is executable, instead of the normal checks listed on the prologue and epilogue page. Default is FALSE.
Example	`$reduce_prolog_checks true`

$restricted	
Format	<STRING>
Description	Specifies hosts which can be trusted to access mom services as non-root. By default, no hosts are trusted to access mom services as non-root.
Example	`$restricted *.teracluster.org`

$rpp_throttle	
Format	<INTEGER>

$rpp_throttle	
Description	This integer is in microseconds and causes a sleep after every RPP packet is sent. It is for systems that experience job failures because of incomplete data.
Example	`$rpp_throttle 100` (will cause a 100 microsecond sleep)

size[fs=<FS>]	
Format	N/A
Description	Specifies that the available and configured disk space in the <FS> filesystem is to be reported to the pbs_server and scheduler. To request disk space on a per job basis, specify the file resource as in `qsub -l nodes=1, file=1000kb`. Unlike most mom config options, the **size** parameter is not preceded by a "**$**" character.
Example	`size[fs=/localscratch]` The available and configured disk space in the `/localscratch` filesystem will be reported.

$source_login_batch	
Format	<STRING>
Description	Specifies whether or not mom will source the `/etc/profile`, etc. type files for *batch* jobs. Parameter accepts various forms of true, false, yes, no, 1 and 0. Default is TRUE. This parameter is in version 2.3.1 and later.
Example	`$source_login_batch False` MOM will bypass the sourcing of `/etc/profile`, etc. type files.

$source_login_interactive	
Format	<STRING>
Description	Specifies whether or not mom will source the `/etc/profile`, etc. type files for *interactive* jobs. Parameter accepts various forms of true, false, yes, no, 1 and 0. Default is TRUE. This parameter is in version 2.3.1 and later.

$source_login_interactive	
Example	`$source_login_interactive False` MOM will bypass the sourcing of `/etc/profile`, etc. type files.

$spool_as_final_name	
Format	<BOOLEAN>
Description	This will spool the job under the final name that the output and error files will receive, instead of having an intermediate file and then copying the result to the final file when the job has completed. This allows users easier access to the file if they want to watch the jobs output as it runs.
Example	`$spool_as_final_name true`

$status_update_time	
Format	<INTEGER>
Description	Specifies the number of seconds between subsequent mom-to-server update reports. Default is 45 seconds.
Example	status_update_time: `$status_update_time 120` MOM will send server update reports every 120 seconds.

$thread_unlink_calls	
Format	<BOOLEAN>
Description	Threads calls to unlink when deleting a job. Default is false. If it is set to TRUE, pbs_mom will use a thread to delete the job's files.
Example	thread_unlink_calls: `$thread_unlink_calls true`

$timeout	
Format	<INTEGER>

$timeout	
Description	Specifies the number of seconds before mom-to-mom messages will timeout if RPP is disabled. Default is 60 seconds.
Example	`$timeout 120` MOM-to-MOM communication will allow up to 120 seconds before timing out.

$tmpdir	
Format	<STRING>
Description	Specifies a directory to create job-specific scratch space (see Creating Per-Job Temporary Directories).
Example	`$tmpdir /localscratch`

$usecp	
Format	<HOST>:<SRCDIR> <DSTDIR>
Description	Specifies which directories should be staged (see **NFS and other networked filesystems** on page 98)
Example	`$usecp *.fte.com:/data /usr/local/data`

$use_smt	
Format	<BOOLEAN>
Description	Indicates that the user would like to use SMT. If set, each logical core inside of a physical core will be used as a normal core for cpusets. This parameter is on by default. If SMT is used, you will need to set the **np** attribute so that each logical processor is counted.
Example	`$use_smt false`

varattr	
Format	<INTEGER> <STRING>

varattr	
Description	Provides a way to keep track of dynamic attributes on nodes. <INTEGER> is how many seconds should go by between calls to the script to update the dynamic values. If set to -1, the script is read only one time. <STRING> is the script path. This script should check for whatever dynamic attributes are desired, and then output lines in this format: `name=value` Include any arguments after the script's full path. These features are visible in the output of **pbsnodes** `-a` varattr=Matlab=7.1;Octave=1.0.
Example	`varattr 25 /usr/local/scripts/nodeProperties.pl arg1 arg2 arg3`

$wallmult	
Format	<FLOAT>
Description	Sets a factor to adjust walltime usage by multiplying a default job time to a common reference system. It modifies real walltime on a per-MOM basis (MOM configuration parameters). The factor is used for walltime calculations and limits in the same way that cputmult is used for cpu time. If set to 0.0, MOM level walltime enforcement is disabled.
Example	`$wallmult 2.2`

Related topics

- **Node manager (MOM) configuration** on page 225

C.1.2 Node features and generic consumable resource specification

Node features (a.k.a. "node properties") are *opaque* labels which can be applied to a node. They are not consumable and cannot be associated with a value. (Use generic resources described below for these purposes). Node features are configured within the global nodes file on the **pbs_server** head node and are not specified on a per node basis. This file can be used to specify an arbitrary number of node features.

Additionally, per node *consumable* generic resources may be specified using the format "<ATTR> <VAL>" with no leading dollar ("**$**") character. When specified, this information is routed to the scheduler and can be used in scheduling decisions. For example, to indicate that a given host has two tape drives and one node-locked matlab license available for batch jobs, the following could be specified:

`mom_priv/config:`

```
$clienthost 241.13.153.7
tape 2
matlab 1
```

Dynamic consumable resource information can be routed in by specifying a value preceded by a exclamation point (**!**) as in the example below. If the resource value is configured in this manner, the specified file will be periodically executed to load the effective resource value. (For more information, see section 2.5.3 of the 'PBS Administrator Guide'.)

`mom_priv/config:`

```
$clienthost 241.13.153.7
tape !/opt/rm/gettapecount.pl
matlab !/opt/tools/getlicensecount.pl
```

Related topics

- **Node manager (MOM) configuration** on page 225

C.1.3 Command-line arguments

Below is a table of pbs_mom command-line startup flags.

Flag	Description
a <integer>	Alarm time in seconds.
c <file>	Config file path.
C <directory>	Checkpoint path.
d <directory>	Home directory.
L <file>	Logfile.
M <integer>	MOM port to listen on.
p	Perform 'poll' based job recovery on restart (jobs persist until associated processes terminate).
P	On restart, deletes all jobs that were running on MOM (Available in 2.4.X and later).
q	On restart, requeues all jobs that were running on MOM (Available in 2.4.X and later).

Flag	Description
r	On restart, kills all processes associated with jobs that were running on MOM, and then requeues the jobs.
R <integer>	MOM 'RM' port to listen on.
S <integer>	pbs_server port to connect to.
v	Display version information and exit.
x	Disable use of privileged port.
?	Show usage information and exit.

For more details on these command-line options, see **pbs_mom** on page 141.

Related topics

- **Node manager (MOM) configuration** on page 225

D.1 Diagnostics and error codes

TORQUE has a diagnostic script to assist you in giving TORQUE Support the files they need to support issues. It should be run by a user that has access to run all TORQUE commands and access to all TORQUE directories (this is usually root).

The script (`contrib/diag/tdiag.sh`) is available in TORQUE 2.3.8, TORQUE 2.4.3, and later. The script grabs the nodefile, server and MOM logfiles, and captures the output of `qmgr -c 'p s'`. These are put in a tarfile.

The script also has the following options (this can be shown in the command line by entering `./tdiag.sh -h`):

```
USAGE: ./torque_diag [-d DATE] [-h] [-o OUTPUT_FILE] [-t TORQUE_HOME]
```

- *DATE* should be in the format YYYYmmdd. For example, " 20091130" would be the date for November 30th, 2009. If no date is specified, today's date is used.
- *OUTPUT_FILE* is the optional name of the output file. The default output file is `torque_diag<today's_date>.tar.gz`. TORQUE_HOME should be the path to your TORQUE directory. If no directory is specified, `/var/spool/torque` is the default.

Table D-1: TORQUE error codes

Error code name	Number	Description
PBSE_NONE	15000	No error
PBSE_UNKJOBID	15001	Unknown job identifier
PBSE_NOATTR	15002	Undefined attribute
PBSE_ATTRRO	15003	Attempt to set READ ONLY attribute
PBSE_IVALREQ	15004	Invalid request
PBSE_UNKREQ	15005	Unknown batch request
PBSE_TOOMANY	15006	Too many submit retries

Error code name	Number	Description
PBSE_PERM	15007	No permission
PBSE_BADHOST	15008	Access from host not allowed
PBSE_JOBEXIST	15009	Job already exists
PBSE_SYSTEM	15010	System error occurred
PBSE_INTERNAL	15011	Internal server error occurred
PBSE_REGROUTE	15012	Parent job of dependent in rte queue
PBSE_UNKSIG	15013	Unknown signal name
PBSE_BADATVAL	15014	Bad attribute value
PBSE_MODATRRUN	15015	Cannot modify attribute in run state
PBSE_BADSTATE	15016	Request invalid for job state
PBSE_UNKQUE	15018	Unknown queue name
PBSE_BADCRED	15019	Invalid credential in request
PBSE_EXPIRED	15020	Expired credential in request
PBSE_QUNOENB	15021	Queue not enabled
PBSE_QACESS	15022	No access permission for queue
PBSE_BADUSER	15023	Bad user - no password entry
PBSE_HOPCOUNT	15024	Max hop count exceeded
PBSE_QUEEXIST	15025	Queue already exists
PBSE_ATTRTYPE	15026	Incompatible queue attribute type
PBSE_QUEBUSY	15027	Queue busy (not empty)

Error code name	Number	Description
PBSE_QUENBIG	15028	Queue name too long
PBSE_NOSUP	15029	Feature/function not supported
PBSE_QUENOEN	15030	Cannot enable queue, needs add def
PBSE_PROTOCOL	15031	Protocol (ASN.1) error
PBSE_BADATLST	15032	Bad attribute list structure
PBSE_NOCONNECTS	15033	No free connections
PBSE_NOSERVER	15034	No server to connect to
PBSE_UNKRESC	15035	Unknown resource
PBSE_EXCQRESC	15036	Job exceeds queue resource limits
PBSE_QUENODFLT	15037	No default queue defined
PBSE_NORERUN	15038	Job not rerunnable
PBSE_ROUTEREJ	15039	Route rejected by all destinations
PBSE_ROUTEEXPD	15040	Time in route queue expired
PBSE_MOMREJECT	15041	Request to the MOM failed
PBSE_BADSCRIPT	15042	(qsub) cannot access script file
PBSE_STAGEIN	15043	Stage In of files failed
PBSE_RESCUNAV	15044	Resources temporarily unavailable
PBSE_BADGRP	15045	Bad group specified
PBSE_MAXQUED	15046	Max number of jobs in queue
PBSE_CKPBSY	15047	Checkpoint busy, may be retries

Error code name	Number	Description
PBSE_EXLIMIT	15048	Limit exceeds allowable
PBSE_BADACCT	15049	Bad account attribute value
PBSE_ALRDYEXIT	15050	Job already in exit state
PBSE_NOCOPYFILE	15051	Job files not copied
PBSE_CLEANEDOUT	15052	Unknown job id after clean init
PBSE_NOSYNCMSTR	15053	No master in Sync Set
PBSE_BADDEPEND	15054	Invalid dependency
PBSE_DUPLIST	15055	Duplicate entry in List
PBSE_DISPROTO	15056	Bad DIS based request protocol
PBSE_EXECTHERE	15057	Cannot execute there
PBSE_SISREJECT	15058	Sister rejected
PBSE_SISCOMM	15059	Sister could not communicate
PBSE_SVRDOWN	15060	Requirement rejected -server shutting down
PBSE_CKPSHORT	15061	Not all tasks could checkpoint
PBSE_UNKNODE	15062	Named node is not in the list
PBSE_UNKNODEATR	15063	Node-attribute not recognized
PBSE_NONODES	15064	Server has no node list
PBSE_NODENBIG	15065	Node name is too big
PBSE_NODEEXIST	15066	Node name already exists
PBSE_BADNDATVAL	15067	Bad node-attribute value

Error code name	Number	Description
PBSE_MUTUALEX	15068	State values are mutually exclusive
PBSE_GMODERR	15069	Error(s) during global modification of nodes
PBSE_NORELYMOM	15070	Could not contact the MOM
PBSE_NOTSNODE	15071	No time-shared nodes

E.1 Considerations before upgrading

TORQUE is flexible in regards to how it can be upgraded. In most cases, a TORQUE "shutdown" followed by a **configure**, **make**, **make install** procedure as documented in this guide is all that is required (see **Installing TORQUE** on page 2). This process will preserve existing configuration and in most cases, existing workload.

A few considerations are included below:

- If upgrading from OpenPBS, PBSPro, or TORQUE 1.0.3 or earlier, queued jobs whether active or idle will be lost. In such situations, job queues should be completely drained of all jobs.
- If not using the **pbs_mom** `-r` or `-p` flag (see **Command-line arguments** on page 240), running jobs may be lost. In such cases, running jobs should be allowed to completed or should be requeued before upgrading TORQUE.
- pbs_mom and pbs_server daemons of differing versions may be run together. However, not all combinations have been tested and unexpected failures may occur.

To upgrade

1. Build new release (do not install).
2. Stop all TORQUE daemons (see **qterm** and **momctl** `-s`).
3. Install new TORQUE (use **make install**).
4. Start all TORQUE daemons.

Rolling upgrade

The **enablemomrestart** option causes a MOM to check if its binary has been updated and will restart itself at a safe point when no jobs are running, making upgrades easier. This can be enabled in the MOM config file, but it is recommended to enable it with momctl.

1. Prepare the new version MOM package.
2. Install the MOM package on the compute nodes.
3. Run `momctl -q enablemomrestart=1 -h :ALL`.

F.1 Large cluster considerations

TORQUE has enhanced much of the communication found in the original OpenPBS project. This has resulted in a number of key advantages:

- Support for larger clusters
- Support for more jobs
- Support for larger jobs
- Support for larger messages

In most cases, enhancements made apply to all systems and no tuning is required. However, some changes have been made configurable to allow site specific modification. The configurable communication parameters are: **node_check_rate**, **node_ping_rate**, and **tcp_timeout**.

For details, see these topics:

- **Scalability guidelines** on page 251
- **End user command caching** on page 252
- **Other considerations** on page 254

F.1.1 Scalability guidelines

In very large clusters (in excess of 1,000 nodes), it may be advisable to additionally tune a number of communication layer timeouts. By default, PBS MOM daemons will timeout on inter-MOM messages after 60 seconds. In TORQUE 1.1.0p5 and higher, this can be adjusted by setting the timeout parameter in the `mom_priv/config` file (see, **Node manager (MOM) configuration** on page 225). If 15059 errors (cannot receive message from sisters) are seen in the MOM logs, it may be necessary to increase this value.

Client-to-PBS server and MOM-to-PBS server communication timeouts are specified via the **tcp_timeout** server option using the **qmgr** command.

On some systems, **ulimit** values may prevent large jobs from running. In particular, the open file descriptor limit (i.e., `ulimit -n`) should be set to at least the maximum job size in procs + 20. Further, there may be value in setting the `fs.file-max` in `sysctl.conf` to a high value, such as:

```
/etc/sysctl.conf:
fs.file-max = 65536
```

Related topics

- **Large cluster considerations** on page 251

F.1.2 End user command caching

qstat

In a large system, users may tend to place excessive load on the system by manual or automated use of resource manager end user client commands. A simple way of reducing this load is through the use of client command wrappers which cache data. The example script below will cache the output of the command '**qstat** `-f`' for 60 seconds and report this info to end users.

```
#!/bin/sh

# USAGE: qstat $@

CMDPATH=/usr/local/bin/qstat
CACHETIME=60
TMPFILE=/tmp/qstat.f.tmp

if [ "$1" != "-f" ] ; then
  #echo "direct check (arg1=$1) "
  $CMDPATH $1 $2 $3 $4
  exit $?
fi

if [ -n "$2" ] ; then
  #echo "direct check (arg2=$2)"
  $CMDPATH $1 $2 $3 $4
  exit $?
fi

if [ -f $TMPFILE ] ; then
  TMPFILEMTIME=`stat -c %Z $TMPFILE`
else
  TMPFILEMTIME=0
fi

NOW=`date +%s`

AGE=$(($NOW - $TMPFILEMTIME))

#echo AGE=$AGE

for i in 1 2 3;do
  if [ "$AGE" -gt $CACHETIME ] ; then
    #echo "cache is stale "

    if [ -f $TMPFILE.1 ] ; then
      #echo someone else is updating cache

      sleep 5

      NOW=`date +%s`

      TMPFILEMTIME=`stat -c %Z $TMPFILE`

AGE=$(($NOW - $TMPFILEMTIME))
    else
      break;
    fi
  fi
done

if [ -f $TMPFILE.1 ] ; then
  #echo someone else is hung

  rm $TMPFILE.1
fi

if [ "$AGE" -gt $CACHETIME ] ; then
  #echo updating cache

  $CMDPATH -f > $TMPFILE.1

mv $TMPFILE.1 $TMPFILE

fi

#echo "using cache"
```

```
cat $TMPFILE

exit 0
```

The above script can easily be modified to cache any command and any combination of arguments by changing one or more of the following attributes:

- script name
- value of $CMDPATH
- value of $CACHETIME
- value of $TMPFILE

For example, to cache the command **pbsnodes** -a, make the following changes:

- Move original pbsnodes command to `pbsnodes.orig`.
- Save the script as '`pbsnodes`'.
- Change $CMDPATH to `pbsnodes.orig`.
- Change $TMPFILE to `/tmp/pbsnodes.a.tmp`.

Related topics

- **Large cluster considerations** on page 251

F.1.3 Other considerations

job_stat_rate

In a large system, there may be many users, many jobs, and many requests for information. To speed up response time for users and for programs using the API the **job_stat_rate** can be used to tweak when the pbs_server daemon will query MOMs for job information. By increasing this number, a system will not be constantly querying job information and causing other commands to block.

poll_jobs

The **poll_jobs** parameter allows a site to configure how the pbs_server daemon will poll for job information. When set to TRUE, the pbs_server will poll job information in the background and not block on user requests. When set to FALSE, the pbs_server may block on user requests when it has stale job information data. Large clusters should set this parameter to TRUE.

Internal settings

On large, slow, and/or heavily loaded systems, it may be desirable to increase the `pbs_tcp_timeout` setting used by the pbs_mom daemon in MOM-to-MOM communication. This setting defaults to 20 seconds and requires rebuilding code to adjust. For client-server based communication, this attribute can be set using the **qmgr** command. For MOM-to-MOM communication, a source code modification is required. To make this change, edit the `$TORQUEBUILDDIR/src/lib/Libifl/tcp_dis.c` file and set `pbs_tcp_timeout` to the desired maximum number of seconds allowed for a MOM-to-MOM request to be serviced.

A system may be heavily loaded if it reports multiple 'End of File from addr' or 'Premature end of message' failures in the pbs_mom or pbs_server logs.

Scheduler settings

If using Moab, there are a number of parameters which can be set on the scheduler which may improve TORQUE performance. In an environment containing a large number of short-running jobs, the JOBAGGREGATIONTIME parameter (see the "Parameters" section of the Moab Workload Manager Administrator Guide) can be set to reduce the number of workload and resource queries performed by the scheduler when an event based interface is enabled. If the pbs_server daemon is heavily loaded and PBS API timeout errors (ie. "Premature end of message") are reported within the scheduler, the "TIMEOUT" attribute of the RMCFG parameter may be set with a value of between 30 and 90 seconds.

File system

TORQUE can be configured to disable file system blocking until data is physically written to the disk by using the `--disable-filesync` argument with **configure**. While having filesync enabled is more reliable, it may lead to server delays for sites with either a larger number of nodes, or a large number of jobs. Filesync is enabled by default.

Network ARP cache

For networks with more than 512 nodes it is mandatory to increase the kernel's internal ARP cache size. For a network of ~1000 nodes, we use these values in `/etc/sysctl.conf` on all nodes and servers:

```
/etc/sysctl.conf

# Don't allow the arp table to become bigger than this
net.ipv4.neigh.default.gc_thresh3 = 4096
# Tell the gc when to become aggressive with arp table cleaning.
# Adjust this based on size of the LAN.
net.ipv4.neigh.default.gc_thresh2 = 2048
# Adjust where the gc will leave arp table alone
net.ipv4.neigh.default.gc_thresh1 = 1024
# Adjust to arp table gc to clean-up more often
net.ipv4.neigh.default.gc_interval = 3600
# ARP cache entry timeout
net.ipv4.neigh.default.gc_stale_time = 3600
```

Use `sysctl -p` to reload this file.

The ARP cache size on other Unixes® can presumably be modified in a similar way.

An alternative approach is to have a static `/etc/ethers` file with all hostnames and MAC addresses and load this by `arp -f /etc/ethers`. However, maintaining this approach is quite cumbersome when nodes get new MAC addresses (due to repairs, for example).

Related topics

- **Large cluster considerations** on page 251

G.1 Prologue and epilogue scripts

TORQUE provides administrators the ability to run scripts before and/or after each job executes. With such a script, a site can prepare systems, perform node health checks, prepend and append text to output and error log files, cleanup systems, and so forth.

The following table shows which MOM runs which script. All scripts must be in the `TORQUE_HOME/mom_priv/` directory and be available on every compute node. "Mother Superior," as referenced in the following table, is the pbs_mom on the first node allocated, and the term "Sisters" refers to pbs_moms, although note that a Mother Superior is also a sister node.

The execution directory for each script is `TORQUE_HOME/mom_priv/`.

Script	Execution location	Execute as	File permissions
prologue	Mother Superior	root	Readable and executable by root and NOT writable by anyone but root (e.g., `-r-x------`)
epilogue		root	
prologue.user		user	Readable and executable by root and other (e.g., `-r-x---r-x`)
epilogue.user		user	
prologue.parallel	Sister	user	Readable and executable by user and NOT writable by anyone but user (e.g., `-r-x--r-x`)
epilogue.parallel		user	
epilogue.precancel	Mother Superior This script runs after a job cancel request is received from pbs_server and before a kill signal is sent to the job process.	user	

epilogue.parallel is available in version 2.1 and later.

This section contains these topics:

- **Script order of execution** on page 258
- **Script environment** on page 258
- **Per job prologue and epilogue scripts** on page 260
- **Prologue and epilogue scripts time out** on page 260
- **Prologue error processing** on page 261

G.1.1 Script order of execution

When jobs start, the order of script execution is `prologue` followed by `prologue.user`. On job exit, the order of execution is `epilogue.user` followed by `epilogue` unless a job is canceled. In that case, `epilogue.precancel` is executed first. `epilogue.parallel` is executed only on the Sister nodes when the job is completed.

The `epilogue` and `prologue` scripts are controlled by the system administrator. However, beginning in TORQUE version 2.4 a user `epilogue` and `prologue` script can be used on a per job basis. (See **Per job prologue and epilogue scripts** on page 260 for more information.)

Root squashing is now supported for `epilogue` and `prologue` scripts.

Related topics

- **Prologue and epilogue scripts** on page 257

G.1.2 Script environment

The `prologue` and `epilogue` scripts can be very simple. On most systems, the script must declare the execution shell using the `#!<SHELL>` syntax (for example, "`#!/bin/sh`"). In addition, the script may want to process context sensitive arguments passed by TORQUE to the script.

Prolog Environment

The following arguments are passed to the `prologue`, `prologue.user`, and `prologue.parallel` scripts:

Argument	Description
argv[1]	job id
argv[2]	job execution user name
argv[3]	job execution group name

Argument	Description
argv[4]	job name (TORQUE 1.2.0p4 and higher only)
argv[5]	list of requested resource limits (TORQUE 1.2.0p4 and higher only)
argv[6]	job execution queue (TORQUE 1.2.0p4 and higher only)
argv[7]	job account (TORQUE 1.2.0p4 and higher only)

Epilog Environment

TORQUE supplies the following arguments to the `epilogue`, `epilogue.user`, `epilogue.precancel`, and `epilogue.parallel` scripts:

Argument	Description
argv[1]	job id
argv[2]	job execution user name
argv[3]	job execution group name
argv[4]	job name
argv[5]	session id
argv[6]	list of requested resource limits
argv[7]	list of resources used by job
argv[8]	job execution queue
argv[9]	job account
argv[10]	job exit code

The `epilogue.precancel` script is run after a job cancel request is received by the MOM and before any signals are sent to job processes. If this script exists, it is run whether the canceled job was active or idle.

The cancel job command (**qdel**) will take as long to return as the `epilogue.precancel` script takes to run. For example, if the script runs for 5 minutes, it takes 5 minutes for qdel to return.

For all scripts, the environment passed to the script is empty. Also, standard input for both scripts is connected to a system dependent file. Currently, for all systems this is `/dev/null`. Except for epilogue scripts of an interactive job, `prologue.parallel`, `epilogue.precancel`, and `epilogue.parallel`, the standard output and error are connected to output and error files associated with the job. For an interactive job, since the pseudo terminal connection is released after the job completes, the standard input and error point to `/dev/null`. For `prologue.parallel` and `epilogue.parallel`, the user will need to redirect `stdout` and `stderr` manually.

Related topics

- **Prologue and epilogue scripts** on page 257

G.1.3 Per job prologue and epilogue scripts

TORQUE supports per job prologue and epilogue scripts when using the **qsub** `-l` option. The syntax is:

`qsub -l prologue=<prologue_script_path> epilogue=<epilogue_script_path> <script>`.

The path can be either relative (from the directory where the job is submitted) or absolute. The files must be owned by the user with at least execute and read privileges, and the permissions must not be writeable by group or other.

`/home/usertom/dev/`

```
-r-x------ 1 usertom usertom 24 2009-11-09 16:11 prologue_script.sh
-r-x------ 1 usertom usertom 24 2009-11-09 16:11 epilogue_script.sh
```

Example G-1:

```
$ qsub -l prologue=/home/usertom/dev/prologue_script.sh,
epilogue=/home/usertom/dev/epilogue_script.sh job14.pl
```

This job submission executes the `prologue` script first. When the `prologue` script is complete, `job14.pl` runs. When `job14.pl` completes, the `epilogue` script is executed.

Related topics

- **Prologue and epilogue scripts** on page 257

G.1.4 Prologue and epilogue scripts time out

TORQUE takes preventative measures against prologue and epilogue scripts by placing an alarm around the scripts execution. By default, TORQUE sets the alarm to go off after 5 minutes of execution. If the script exceeds this time, it will be terminated and the node will be marked down. This timeout can be adjusted by setting the **$prologalarm** parameter in the `mom_priv/config` file.

While TORQUE is executing the `epilogue`, `epilogue.user`, or `epilogue.precancel` scripts, the job will be in the **E** (exiting) state.

Related topics

- **Prologue and epilogue scripts** on page 257

G.1.5 Prologue error processing

If the `prologue` script executes successfully, it should exit with a zero status. Otherwise, the script should return the appropriate error code as defined in the table below. The pbs_mom will report the script's exit status to pbs_server which will in turn take the associated action. The following table describes each exit code for the prologue scripts and the action taken.

Error	Description	Action
-4	The script timed out	Job will be requeued
-3	The wait(2) call returned an error	Job will be requeued
-2	Input file could not be opened	Job will be requeued
-1	Permission error (script is not owned by root, or is writable by others)	Job will be requeued
0	Successful completion	Job will run
1	Abort exit code	Job will be aborted
>1	other	Job will be requeued

Example G-1:

Following are example prologue and epilogue scripts that write the arguments passed to them in the job's standard out file:

prologue	

Script	`#!/bin/sh`
	`echo "Prologue Args:"`
	`echo "Job ID: $1"`
	`echo "User ID: $2"`
	`echo "Group ID: $3"`
	`echo ""`
	
	`exit 0`

prologue	
stdout	Prologue Args: Job ID: 13724.node01 User ID: user1 Group ID: user1

epilogue	
Script	#!/bin/sh echo "Epilogue Args:" echo "Job ID: $1" echo "User ID: $2" echo "Group ID: $3" echo "Job Name: $4" echo "Session ID: $5" echo "Resource List: $6" echo "Resources Used: $7" echo "Queue Name: $8" echo "Account String: $9" echo "" exit 0
stdout	Epilogue Args: Job ID: 13724.node01 User ID: user1 Group ID: user1 Job Name: script.sh Session ID: 28244 Resource List: neednodes=node01,nodes=1,walltime=00:01:00 Resources Used: cput=00:00:00,mem=0kb,vmem=0kb,walltime=00:00:07 Queue Name: batch Account String:

Example G-2:

The Ohio Supercomputer Center contributed the following scripts:

"prologue creates a unique temporary directory on each node assigned to a job before the job begins to run, and epilogue deletes that directory after the job completes.

Having a separate temporary directory on each node is probably not as good as having a good, high performance parallel filesystem.

prologue

```
#!/bin/sh
# Create TMPDIR on all the nodes
# Copyright 1999, 2000, 2001 Ohio Supercomputer Center
# prologue gets 3 arguments:
# 1 -- jobid
# 2 -- userid
# 3 -- grpid
#
jobid=$1
user=$2
group=$3
nodefile=/var/spool/pbs/aux/$jobid
if [ -r $nodefile ] ; then
    nodes=$(sort $nodefile | uniq)
else
    nodes=localhost
fi
tmp=/tmp/pbstmp.$jobid
for i in $nodes ; do
    ssh $i mkdir -m 700 $tmp \&\& chown $user.$group $tmp
done
exit 0
```

epilogue

```
#!/bin/sh
# Clear out TMPDIR
# Copyright 1999, 2000, 2001 Ohio Supercomputer Center
# epilogue gets 9 arguments:
# 1 -- jobid
# 2 -- userid
# 3 -- grpid
# 4 -- job name
# 5 -- sessionid
# 6 -- resource limits
# 7 -- resources used
# 8 -- queue
# 9 -- account
#
jobid=$1
nodefile=/var/spool/pbs/aux/$jobid
if [ -r $nodefile ] ; then
    nodes=$(sort $nodefile | uniq)
else
    nodes=localhost
fi
tmp=/tmp/pbstmp.$jobid
for i in $nodes ; do
    ssh $i rm -rf $tmp
done
exit 0
```

`prologue`, `prologue.user`, and `prologue.parallel` scripts can have dramatic effects on job scheduling if written improperly.

Related topics

- **Prologue and epilogue scripts** on page 257

H.1 Running multiple TORQUE servers and MOMs on the same node

TORQUE can be configured to allow multiple servers and MOMs to run on the same node. This example will show how to configure, compile and install two different TORQUE servers and moms on the same node. For details, see these topics:

- **Configuring the first TORQUE** on page 265
- **Configuring the second TORQUE** on page 265
- **Bringing the first TORQUE server online** on page 265
- **Bringing the second TORQUE server online** on page 266

Configuring the first TORQUE

```
./configure --with-server-home=/usr/spool/PBS1 --bindir=/usr/spool/PBS1/bin --
sbindir=/usr/spool/PBS1/sbin
```

Then make and make install will place the first TORQUE into `/usr/spool/PBS1` with the executables in their corresponding directories.

Configuring the second TORQUE

```
./configure --with-server-home=/usr/spool/PBS2 --bindir=/usr/spool/PBS2/bin --
sbindir=/usr/spool/PBS2/sbin
```

Then make and make install will place the second TORQUE into `/usr/spool/PBS2` with the executables in their corresponding directories.

Bringing the first TORQUE server online

Each command, including pbs_server and pbs_mom, takes parameters indicating which servers and ports to connect to or listen on (when appropriate). Each of these is documented in their corresponding man pages (configure with `--enable-docs`).

In this example the first TORQUE server will accept batch requests on port 35000, communicate with the MOMs on port 35001, and communicate via RPP on port 35002. The first TORQUE MOM will try to connect to the server on port 35000, it will listen for requests from the server on port 35001 and will communicate via RPP on port 35002. (Each of these command arguments is discussed in further details on the corresponding man page. In particular, `-t create` is only used the first time a server is run.)

```
> pbs_server -p 35000 -M 35001 -R 35002 -t create
> pbs_mom -S 35000 -M 35001 -R 35002
```

Afterwards, when using a client command to make a batch request it is necessary to specify the servername and serverport (35000):

```
> pbsnodes -a -s node01:35000
```

Submitting jobs can be accomplished using the `-q` option ([queue][@host[:port]]):

```
> qsub -q @node01:35000 /tmp/script.pbs
```

Bringing the second TORQUE server online

In this example the second TORQUE server will accept batch requests on port 36000, communicate with the MOMS on port 36002, and communicate via RPP on port 36002. The second TORQUE MOM will try to connect to the server on port 36000, it will listen for requests from the server on port 36001 and will communicate via RPP on port 36002.

```
> pbs_server -p 36000 -M 36001 -R 36002 -t create
> pbs_mom -S 36000 -M 36001 -R 36002
```

Afterward, when using a client command to make a batch request it is necessary to specify the servername and serverport (36002):

```
> pbsnodes -a -s node01:36000
> qsub -q @node01:36000 /tmp/script.pbs
```

I.1 Security overview

SUID ssage

TORQUE uses setuid (SUID) permissions in a single location so as to validate the identity of a user request. This is accomplished using the pbs_iff tool which is SUID root and performs the following actions:

- parse specified server hostname and port
- connect to specified server port using reserved/privileged port
- determine UID of executing user
- report UID and socket port info of caller to server
- verify response from server

/etc/hosts usage

In systems where security is a major concern, please be aware that some security experts consider adding the compute nodes to the `/etc/hosts` file to be more secure than using ACL lists.

J.1 Job submission filter ("qsub wrapper")

When a "submit filter" exists, TORQUE will send the command file (or contents of STDIN if piped to qsub) to that script/executable and allow it to evaluate the submitted request based on specific site policies. The resulting file is then handed back to qsub and processing continues. Submit filters can check user jobs for correctness based on site policies. They can also modify user jobs as they are submitted. Some examples of what a submit filter might evaluate and check for are:

- Memory Request - Verifiy that the job requests memory and rejects if it does not.
- Job event notifications - Check if the job does one of the following and rejects it if it:
  - explicitly requests no notification.
  - requests notifications but does not provide an email address.
- Walltime specified - Verify that the walltime is specified.
- Global Walltime Limit - Verify that the walltime is below the global max walltime.
- Test Walltime Limit - If the job is a test job, this check rejects the job it if it requests a walltime longer than the testing maximum.

The script below reads the original submission request from STDIN and shows how you could insert parameters into a job submit request:

```
#!/bin/sh
# add default memory constraints to all requests
# that did not specify it in user's script or on command line
echo "#PBS -l mem=16MB"
while read i
do
echo $i
done
```

Command line arguments passed to qsub are passed as arguments to the submit filter (filter won't see them in STDIN) in the same order and may be used as needed. It should be noted that as of TORQUE 2.2.0 extended attributes are not passed to the filter. Exit status of -1 will cause qsub to reject the submission with a message stating that it failed due to administrative policies.

The "submit filter" must be executable, must be available on each of the nodes where users may submit jobs, and by default, must be located at `${libexecdir}/qsub_filter` (for version 2.1 and older: `/usr/local/sbin/torque_submitfilter`). At run time, if the file does not exist at this new preferred path then qsub will fall back to the old hard-coded path. The submit filter location can be customized by setting the **SUBMITFILTER** parameter inside the file (see **"torque.cfg" configuration file** on page 271), as in the following example:

`torque.cfg:`

```
SUBMITFILTER /opt/torque/submit.pl
...
```

Initial development courtesy of Oak Ridge National Laboratories.

K.1 "torque.cfg" configuration file

CLIENTRETRY	
Format	<INT>
Default	0
Description	Seconds between retry attempts to talk to pbs_server.

DEFAULTCKPT	
Format	<STRING>
Default	None
Description	Default value for job's checkpoint attribute.

FAULT_TOLERANT_BY-DEFAULT	
Format	<BOOLEAN>
Default	FALSE
Description	Sets all jobs to fault tolerant by default. (See **qsub** `-f` for more information on fault tolerance.)

QSUBHOST	
Format	<HOSTNAME>
Default	None

QSUBHOST	
Description	The hostname given as the argument of this option will be used as the PBS_O_HOST variable for job submissions. By default, PBS_O_HOST is the hostname of the submission host. This option allows administrators to override the default hostname and substitute a new name.

QSUBSENDUID	
Format	N/A
Default	None
Description	Integer for job's PBS_OUID variable. Specifying the parameter name anywhere in the config file enables the feature. Removing the parameter name disables the feature.

QSUBSLEEP	
Format	<INT>
Default	0
Description	Specifies time to sleep when running qsub command. Used to prevent users from overwhelming the scheduler.

RERUNNABLEBYDEFAULT	
Format	<BOOLEAN>
Default	TRUE
Description	Specifies if a job is re-runnable by default. Setting this to false causes the re-runnable attribute value to be false unless the users specifies otherwise with the **qsub** `-r` option. (New in TORQUE 2.4.)

SERVERHOST	
Format	<STRING>
Default	localhost

SERVERHOST	
Description	If set, the server will open socket connections and communicate with client commands and other services using the specified network interface. (useful with multi-homed hosts, i.e., nodes with multiple network adapters)

SUBMITFILTER	
Format	<STRING>
Default	${libexecdir}/qsub_filter (for version 2.1 and older: /usr/local/sbin/torque_submitfilter)
Description	Specifies the location of the submit filter (see **Job submission filter ("qsub wrapper")** on page 269 used to pre-process job submission.

TRQ_IFNAME	
Format	<STRING>
Default	null
Description	Allows you to specify a specific network interface to use for outbound TORQUE requests. The string is the name of a network interface, such as **eth0** or **eth1**, depending on which interface you want to use.

VALIDATEGROUP	
Format	<BOOLEAN>
Default	FALSE
Description	Validate submit user's group on **qsub** commands. For TORQUE builds released after 2/8/2011, **VALIDATEGROUP** also checks any groups requested in group_list at the submit host. Set **VALIDATEGROUP** to "TRUE" if you set **disable_server_id_check** to TRUE.

VALIDATEPATH	
Format	<BOOLEAN>
Default	TRUE
Description	Validate local existence of '-d' working directory.

Example K-1:

`torque.cfg:`

```
QSUBSLEEP          2
SERVERHOST         orion15
```

L.1 TORQUE Quick Start Guide

Initial installation

Download the TORQUE distribution file from http://clusterresources.com/downloads/torque.

Extract and build the distribution on the machine that will act as the "TORQUE server" - the machine that will monitor and control all compute nodes by running the pbs_server daemon. See the example below:

```
> tar -xzvf torque.tar.gz
> cd torque
> ./configure
> make
> make install
```

OSX 10.4 users need to change the *#define __TDARWIN* in `src/include/pbs_config.h` to **#define __TDARWIN_8**.

After installation, verify you have PATH environment variables configured for `/usr/local/bin/` and `/usr/local/sbin/`. Client commands are installed to `/usr/local/bin` and server binaries are installed to `/usr/local/sbin`.

In this document, TORQUE_HOME corresponds to where TORQUE stores its configuration files. The default is `/var/spool/torque`.

Initialize/Configure TORQUE on the server (pbs_server)

- Once installation on the TORQUE server is complete, configure the pbs_server daemon by executing the command `torque.setup <USER>` found packaged with the distribution source code, where *<USER>* is a username that will act as the TORQUE admin. This script will set up a basic batch queue to get you started. If you experience problems, make sure that the most recent TORQUE executables are being executed, or that the executables are in your current PATH.
- If doing this step manually, be certain to run the command `pbs_server -t create` to create the new batch database. If this step is not taken, the pbs_server daemon will be unable to start.
- Proper server configuration can be verified by following the steps listed in Testing server configuration.

Install TORQUE on the compute nodes

To configure a compute node do the following on each machine (see page 19, Section 3.2.1 of PBS Administrator's Manual for full details):

- Create the self-extracting, distributable packages with make packages (See the `INSTALL` file for additional options and features of the distributable packages) and use the parallel shell command from your cluster management suite to copy and execute the package on all nodes (ie: xCAT users might do `prcp torque-package-linux-i686.sh main:/tmp/; psh main /tmp/torque-package-linux-i686.sh --install`). Optionally, distribute and install the clients package.

Configure TORQUE on the compute nodes

- For each compute host, the MOM daemon must be configured to trust the pbs_server daemon. In TORQUE 2.0.0p4 and earlier, this is done by creating the `TORQUE_HOME/mom_priv/config` file and setting the **$pbsserver** parameter. In TORQUE 2.0.0p5 and later, this can also be done by creating the `TORQUE_HOME/server_name` file and placing the server hostname inside.
- Additional config parameters may be added to `TORQUE_HOME/mom_priv/config` (see **Node manager (MOM) configuration** on page 225 for details).

Configure data management on the compute nodes

Data management allows jobs' data to be staged in/out or to and from the server and compute nodes.

- For shared filesystems (i.e., NFS, DFS, AFS, etc.) use the **$usecp** parameter in the `mom_priv/config` files to specify how to map a user's home directory.

  (Example: `$usecp gridmaster.tmx.com:/home /home`)
- For local, non-shared filesystems, rcp or scp must be configured to allow direct copy without prompting for passwords (key authentication, etc.)

Update TORQUE server configuration

On the TORQUE server, append the list of newly configured compute nodes to the `TORQUE_HOME/server_priv/nodes` file:

```
server_priv/nodes

computenode001.cluster.org
computenode002.cluster.org
computenode003.cluster.org
```

Start the pbs_mom daemons on compute nodes

- Next start the pbs_mom daemon on each compute node by running the pbs_mom executable.

Run the trqauthd daemon to run client commands (see **Configuring trqauthd for client commands** on page 10). This enables running client commands.

Verifying correct TORQUE installation

The pbs_server daemon was started on the TORQUE server when the `torque.setup` file was executed or when it was manually configured. It must now be restarted so it can reload the updated configuration changes.

```
# shutdown server
> qterm # shutdown server

# start server
> pbs_server

# verify all queues are properly configured
> qstat -q

# view additional server configuration
> qmgr -c 'p s'

# verify all nodes are correctly reporting
> pbsnodes -a

# submit a basic job
>echo "sleep 30" | qsub

# verify jobs display
> qstat
```

At this point, the job will not start because there is no scheduler running. The scheduler is enabled in the next step below.

Enabling the scheduler

Selecting the cluster scheduler is an important decision and significantly affects cluster utilization, responsiveness, availability, and intelligence. The default TORQUE scheduler, pbs_sched, is very basic and will provide poor utilization of your cluster's resources. Other options, such as Maui Scheduler or Moab Workload Manager are highly recommended. If using Maui/Moab, refer to the Moab-PBS Integration Guide. If using pbs_sched, start this daemon now.

If you are installing ClusterSuite, TORQUE and Moab were configured at installation for interoperability and no further action is required.

Startup/Shutdown service script for TORQUE/Moab (OPTIONAL)

Optional startup/shutdown service scripts are provided as an example of how to run TORQUE as an OS service that starts at bootup. The scripts are located in the `contrib/init.d/` directory of the TORQUE tarball you downloaded. In order to use the script you must:

- Determine which init.d script suits your platform the best.
- Modify the script to point to TORQUE's install location. This should only be necessary if you used a non-default install location for TORQUE (by using the `--prefix` option of `./configure`).
- Place the script in the `/etc/init.d/` directory.
- Use a tool like `chkconfig` to activate the start-up scripts or make symbolic links (S99moab and K15moab, for example) in desired runtimes (`/etc/rc.d/rc3.d/` on Redhat, etc.).

Related topics

- **Advanced configuration** on page 12

M.1 BLCR acceptance tests

This section contains a description of the testing done to verify the functionality of the BLCR implementation. For details, see these topics:

- **Test environment** on page 279
- **Test 1 - Basic operation** on page 280
- **Test 2 - Persistence of checkpoint images** on page 282
- **Test 3 - Restart after checkpoint** on page 283
- **Test 4 - Multiple checkpoint/restart** on page 284
- **Test 5 - Periodic checkpoint** on page 284
- **Test 6 - Restart from previous image** on page 285

M.1.1 Test environment

All these tests assume the following test program and shell script, `test.sh`.

```
#include
int main( int argc, char *argv[] )
{
int i;

    for (i=0; i<100; i++)
    {
        printf("i = %d\n", i);
        fflush(stdout);
        sleep(1);
    }
}
#!/bin/bash

/home/test/test
```

Related topics

- **BLCR acceptance tests** on page 279

M.1.2 Test 1 - Basic operation

Introduction

This test determines if the proper environment has been established.

Test steps

Submit a test job and the issue a hold on the job.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
```

Possible failures

Normally the result of qhold is nothing. If an error message is produced saying that qhold is not a supported feature then one of the following configuration errors might be present.

- The TORQUE images may have not be configured with `--enable-blcr`
- BLCR support may not be installed into the kernel with insmod.
- The config script in mom_priv may not exist with `$checkpoint_script defined`.
- The config script in mom_priv may not exist with `$restart_script defined`.
- The config script in mom_priv may not exist with `$checkpoint_run_exe defined`.
- The scripts referenced in the config file may not exist.
- The scripts referenced in the config file may not have the correct permissions.

Successful results

If no configuration was done to specify a specific directory location for the checkpoint file, the default location is off of the TORQUE directory, which in my case is `/var/spool/torque/checkpoint`.

Otherwise, go to the specified directory for the checkpoint image files. This was done by either specifying an option on job submission, i.e. `-c dir=/home/test` or by setting an attribute on the execution quere. This is done with the command `qmgr -c 'set queue batch checkpoint_dir=/home/test'`.

Doing a directory listing shows the following.

```
# find /var/spool/torque/checkpoint
/var/spool/torque/checkpoint
/var/spool/torque/checkpoint/999.xxx.yyy.CK
/var/spool/torque/checkpoint/999.xxx.yyy.CK/ckpt.999.xxx.yyy.1205266630
# find /var/spool/torque/checkpoint |xargs ls -l
-r-------- 1 root root 543779 2008-03-11 14:17
/var/spool/torque/checkpoint/999.xxx.yyy.CK/ckpt.999.xxx.yyy.1205266630

/var/spool/torque/checkpoint:
total 4
drwxr-xr-x 2 root root 4096 2008-03-11 14:17 999.xxx.yyy.CK

/var/spool/torque/checkpoint/999.xxx.yyy.CK:
total 536
-r-------- 1 root root 543779 2008-03-11 14:17 ckpt.999.xxx.yyy.1205266630
```

Doing a `qstat -f` command should show the job in a held state, **job_state = H**. Note that the attribute checkpoint_name is set to the name of the file seen above.

If a checkpoint directory has been specified, there will also be an attribute **checkpoint_dir** in the output of `qstat -f`.

```
$ qstat -f
Job Id: 999.xxx.yyy
    Job_Name = test.sh
    Job_Owner = test@xxx.yyy
    resources_used.cput = 00:00:00
    resources_used.mem = 0kb
    resources_used.vmem = 0kb
    resources_used.walltime = 00:00:06
    job_state = H
    queue = batch
    server = xxx.yyy
    Checkpoint = u
    ctime = Tue Mar 11 14:17:04 2008
    Error_Path = xxx.yyy:/home/test/test.sh.e999
    exec_host = test/0
    Hold_Types = u
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Tue Mar 11 14:17:10 2008
    Output_Path = xxx.yyy:/home/test/test.sh.o999
    Priority = 0
    qtime = Tue Mar 11 14:17:04 2008
    Rerunable = True
    Resource_List.neednodes = 1
    Resource_List.nodect = 1
    Resource_List.nodes = 1
    Resource_List.walltime = 01:00:00
    session_id = 9402 substate = 20
    Variable_List = PBS_O_HOME=/home/test,PBS_O_LANG=en_US.UTF-8,
        PBS_O_LOGNAME=test,
        PBS_O_PATH=/usr/local/perltests/bin:/home/test/bin:/usr/local/s
bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games,
        PBS_O_SHELL=/bin/bash,PBS_SERVER=xxx.yyy,
        PBS_O_HOST=xxx.yyy,PBS_O_WORKDIR=/home/test,
        PBS_O_QUEUE=batch
    euser = test
    egroup = test
    hashname = 999.xxx.yyy
    queue_rank = 3
    queue_type = E comment = Job started on Tue Mar 11 at 14:17
    exit_status = 271
    submit_args = test.sh
    start_time = Tue Mar 11 14:17:04 2008
    start_count = 1
    checkpoint_dir = /var/spool/torque/checkpoint/999.xxx.yyy.CK
    checkpoint_name = ckpt.999.xxx.yyy.1205266630
```

Related topics

- **BLCR acceptance tests** on page 279

M.1.3 Test 2 - Persistence of checkpoint images

Introduction

This test determines if the checkpoint files remain in the default directory after the job is removed from the TORQUE queue.

Note that this behavior was requested by a customer but in fact may not be the right thing to do as it leaves the checkpoint files on the execution node. These will gradually build up over time on the node

being limited only by disk space. The right thing would seem to be that the checkpoint files are copied to the users home directory after the job is purged from the execution node.

Test steps

Assuming the steps of Test 1 (see **Test 1 - Basic operation** on page 280), delete the job and then wait until the job leaves the queue after the completed job hold time. Then look at the contents of the default checkpoint directory to see if the files are still there.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
> qdel 999
> sleep 100
> qstat
>
> find /var/spool/torque/checkpoint
... files ...
```

Possible failures

The files are not there, did Test 1 actually pass?

Successful results

The files are there.

Related topics

- **BLCR acceptance tests** on page 279

M.1.4 Test 3 - Restart after checkpoint

Introduction

This test determines if the job can be restarted after a checkpoint hold.

Test steps

Assuming the steps of Test 1 (see **Test 1 - Basic operation** on page 280), issue a **qrls** command. Have another window open into the `/var/spool/torque/spool` directory and tail the job.

Successful results

After the qrls, the job's output should resume.

Related topics

- **BLCR acceptance tests** on page 279

M.1.5 Test 4 - Multiple checkpoint/restart

Introduction

This test determines if the checkpoint/restart cycle can be repeated multiple times.

Test steps

Start a job and then while tail'ing the job output, do multiple **qhold**/**qrls** operations.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
> qrls 999
> qhold 999
> qrls 999
> qhold 999
> qrls 999
```

Successful results

After each qrls, the job's output should resume. Also tried "while true; do qrls 999; qhold 999; done" and this seemed to work as well.

Related topics

- **BLCR acceptance tests** on page 279

M.1.6 Test 5 - Periodic checkpoint

Introduction

This test determines if automatic periodic checkpoint will work.

Test steps

Start the job with the option `-c enabled,periodic,interval=1` and look in the checkpoint directory for checkpoint images to be generated about every minute.

```
> qsub -c enabled,periodic,interval=1 test.sh
999.xxx.yyy
```

Successful results

After each qrls, the job's output should resume. Also tried "while true; do qrls 999; qhold 999; done" and this seemed to work as well.

Related topics

- **BLCR acceptance tests** on page 279

M.1.7 Test 6 - Restart from previous image

Introduction

This test determines if the job can be restarted from a previous checkpoint image.

Test steps

Start the job with the option `-c enabled,periodic,interval=1` and look in the checkpoint directory for checkpoint images to be generated about every minute. Do a **qhold** on the job to stop it. Change the attribute checkpoint_name with the **qalter** command. Then do a **qrls** to restart the job.

```
> qsub -c enabled,periodic,interval=1 test.sh
999.xxx.yyy
> qhold 999
> qalter -W checkpoint_name=ckpt.999.xxx.yyy.1234567
> qrls 999
```

Successful results

The job output file should be truncated back and the count should resume at an earlier number.

Related topics

- **BLCR acceptance tests** on page 279