Torque Resource Manager

Administrator Guide 6.1.0

November 2016

Scan to open online help

Welcome

Welcome to the *Torque 6.1.0 Administrator Guide*.

This guide is intended as a reference for system administrators.

For more information about this guide, see these topics:

- Torque Administrator Guide Overview
- Introduction

Torque Administrator Guide Overview

Chapter 1 Introduction on page 5 provides basic introduction information to help you get started using Torque.

Chapter 2 Overview on page 9 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 3 Submitting and Managing Jobs on page 69 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 4 Managing Nodes on page 111 covers administrator tasks relating to nodes, which include 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.

Chapter 5 Setting Server Policies on page 127 details server-side configurations of queue and high availability.

Chapter 6 Integrating Schedulers for Torque on page 149 offers information about using the native scheduler versus an advanced scheduler.

Chapter 7 Configuring Data Management on page 151 deals with issues of data management. For non-network file systems, SCP Setup details setting up SSH keys and nodes to automate transferring data. NFS and Other Networked Filesystems covers configuration for these file systems. This chapter also addresses the use of file staging using the **stagein** and **stageout** directives of the qsub command.

Chapter 8 MPI (Message Passing Interface) Support on page 157 offers details supporting MPI.

Chapter 9 Resources on page 161 covers configuration, utilization, and states of resources.

Chapter 10 Accounting Records on page 165 explains how jobs are tracked by Torque for accounting purposes.

Chapter 11 Job Logging on page 167 explains how to enable job logs that contain information for completed jobs.

Chapter 12 NUMA and Torque on page 169 provides a centralized location for information on configuring Torque for NUMA systems.

Chapter 13 Troubleshooting on page 199 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.

Related Topics

Introduction

Chapter 1 Introduction

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

- What is a Resource Manager?
- What are Batch Systems?
- Basic Job Flow

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.

Component	Description
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: ```\n#PBS -N localBlast\n#PBS -S /bin/sh\n#PBS -l nodes=1:ppn=2,walltime=240:00:00\n#PBS -M user@my.organization.com\n#PBS -m ea\nsource ~/.bashrc\ncd $HOME/work/dir\nsh myBlast.sh -i -v\n``` 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.

Stage	Description
Finalilzation	When a job completes, by default, the `stdout` and `stderr` files are copied to the directory where the job was submitted.

Chapter 2 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
- Initializing/Configuring Torque on the Server (pbs_server)
- Advanced Configuration
- Manual Setup of Initial Server Configuration
- Server Node File Configuration
- Testing Server Configuration
- Configuring Torque for NUMA Systems
- Torque Multi-MOM

Torque Installation Overview

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

For details, see these topics:

- Torque Architecture
- Installing Torque Resource Manager
- Compute Nodes
- Enabling Torque as a Service

Related Topics

Troubleshooting

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

Installing Torque Resource Manager

> ⚠️ If you intend to use Torque Resource Manager 6.1.0 with Moab Workload Manager, you must run Moab version 8.0 or later. However, some Torque functionality may not be available. See Compatibility Requirements in the Moab HPC Suite Release Notes for more information.

This topic contains instructions on how to install and start Torque Resource Manager (Torque).

> ℹ️ For Cray systems, Adaptive Computing recommends that you install Moab and Torque Servers (head nodes) on commodity hardware (*not* on Cray compute/service/login nodes).
>
> *However*, you *must* install the Torque pbs_mom daemon and Torque client commands on Cray login and "mom" service nodes since the pbs_mom *must* run on a Cray service node within the Cray system so it has access to the Cray ALPS subsystem.
>
> See Installation Notes for Moab and Torque for Cray in the *Moab Workload Manager Administrator Guide* for instructions on installing Moab and Torque on a non-Cray server.

In this topic:

- Requirements on page 11
- Open Necessary Ports on page 12
- Install Dependencies, Packages, or Clients on page 12
- Install Torque Server on page 14
- Install Torque MOMs on page 16

Requirements

In this section:

Supported Operating Systems

- CentOS 6.x, 7.x
- RHEL 6.x, 7.x
- Scientific Linux 6.x, 7.x
- SUSE Linux Enterprise Server 11, 12

Software Requirements

- libxml2-devel package (package name may vary)
- openssl-devel package (package name may vary)
- Tcl/Tk version 8 or later if you plan to build the GUI portion of Torque, or use a Tcl-based scheduler
- cpusets and cgroups

  cgroups are supported and cpusets are handled by the cgroup cpuset subsystem.

  > ⓘ It is recommended that you use --enable-cgroups instead of --enable-cpuset. --enable-cpuset is deprecated and no new features will be added to it.

  - boost version: 1.41 or later
  - libcgroup version: Red Hat-based systems must use libcgroup version 0.40.rc1-16.el6 or later; SUSE-based systems need to use a comparative libcgroup version.
  - libhwloc version: 1.9.1 is the minimum supported, however NVIDIA K80 requires libhwloc 1.11.0. Instructions for installing hwloc are provided as part of the Torque Resource Manager install or upgrade instructions.
- if you build Torque from source (i.e. clone from github), the following additional software is required:

○ gcc

○ gcc-c++

○ posix-compatible version of make

○ libtool 1.5.22 or later

○ boost-devel 1.36.0 or later

> ⓘ Red Hat 6-based systems come packaged with 1.41.0 and Red Hat 7-based systems come packaged with 1.53.0. If needed, use the --with-boost-path=DIR option to change the packaged boost version. See Customizing the Install on page 35 for more information.

Open Necessary Ports

Torque requires certain ports to be open for essential communication.

If your site is running firewall software on its hosts, you will need to configure the firewall to allow connections to the necessary ports.

Location	Ports	Functions	When Needed
Torque Server Host	15001	Torque Client and MOM communication to Torque Server	Always
Torque MOM Host (Compute Nodes)	15002	Torque Server communication to Torque MOMs	Always
Torque MOM Host (Compute Nodes)	15003	Torque MOM communication to other Torque MOMs	Always

See also:

- Opening Ports in a Firewall on page 50 for general instructions and an example of how to open ports in the firewall.

- Configuring Ports on page 26 for more information on how to configure the ports that Torque uses for communication.

Install Dependencies, Packages, or Clients

In this section:

Install Packages

On the Torque Server Host, use the following commands to install the `libxml2-devel`, `openssl-devel`, and `boost-devel` packages.

- Red Hat 6-based or Red Hat 7-based systems

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

- SUSE 11-based or SUSE 12-based systems

```
[root]# zypper install libopenssl-devel libtool libxml2-devel boost-devel gcc gcc-c++
make gmake
```

Install hwloc

> ⚠ Using "yum install hwloc" for Red Hat-based systems or "zypper install hwloc" for SUSE-based systems may install an older, non-supported version.

When cgroups are enabled (recommended), hwloc version 1.9.1 or later is required. NVIDIA K80 requires libhwloc 1.11.0.

The following instructions are for installing version 1.9.1.

Do the following:

1. On the Torque Server Host, *each* Torque MOM Host, and *each* Torque Client Host, do the following:

   a. Download hwloc-1.9.1.tar.gz from https://www.open-mpi.org/software/hwloc/v1.9.

   b. Run each of the following commands in order.

      - Red Hat 6-based or Red Hat 7-based systems

```
[root]# yum install gcc make
[root]# tar -xzvf hwloc-1.9.1.tar.gz
[root]# cd hwloc-1.9.1.tar.gz
[root]# ./configure
[root]# make
[root]# make install
```

      - SUSE 11-based or SUSE 12-based systems

```
[root]# zypper install gcc make
[root]# tar -xzvf hwloc-1.9.1.tar.gz
[root]# cd hwloc-1.9.1.tar.gz
[root]# ./configure
[root]# make
[root]# make install
```

2. For SUSE 11- based systems, run the following commands on the Torque Server Host, *each* Torque MOM Host, and *each* Torque Client Host.

```
[root]# echo /usr/local/lib >/etc/ld.so.conf.d/hwloc.conf
[root]# ldconfig
```

3. For SUSE 12- based systems, run the following commands on the Torque Server Host *only*.

```
[root]# echo /usr/local/lib >/etc/ld.so.conf.d/hwloc.conf
[root]# ldconfig
```

Install Torque Server

> ℹ️ You *must* complete the prerequisite tasks and the tasks to install the dependencies, packages, or clients before installing Torque Server. See Installing Torque Resource Manager on page 10 and Install Dependencies, Packages, or Clients on page 12.

On the Torque Server Host, do the following:

1. Download the latest 6.1.0 build from the Adaptive Computing website. It can also be downloaded via command line (github method or the tarball distribution).

   - Clone the source from github.

     > ℹ️ If git is not installed:
     >
     > ```
     > # Red Hat 6-based or Red Hat 7-based systems
     > [root]# yum install git
     >
     > # SUSE 11-based or SUSE 12-based systems
     > [root]# zypper install git
     > ```
     >
     > For SUSE 11-based systems, you will *also* need to install automake:
     >
     > ```
     > [root]# zypper install autotools automake pkg-config
     > ```

     ```
     [root]# git clone https://github.com/adaptivecomputing/torque.git -b 6.1.0 6.1.0
     [root]# cd 6.1.0
     [root]# ./autogen.sh
     ```

- Get the tarball source distribution.

    - Red Hat 6-based or Red Hat 7-based systems

```
[root]# yum install wget
[root]# wget http://www.adaptivecomputing.com/download/torque/torque-
6.1.0.tar.gz -O torque-6.1.0.tar.gz
[root]# tar -xzvf torque-6.1.0.tar.gz
[root]# cd torque-6.1.0/
```

    - SUSE 11-based or SUSE 12-based systems

```
[root]# zypper install wget
[root]# wget http://www.adaptivecomputing.com/download/torque/torque-
6.1.0.tar.gz -O torque-6.1.0.tar.gz
[root]# tar -xzvf torque-6.1.0.tar.gz
[root]# cd torque-6.1.0/
```

2. Depending on your system configuration, you will need to add ./configure command options.

    At a minimum, you add:

    - --enable-cgroups

    - --with-hwloc-path=/usr/local See for more information.

    > 🛈 These instructions assume you are using cgroups. When cgroups are supported, cpusets are handled by the cgroup cpuset subsystem. If you are not using cgroups, use --enable-cpusets instead.

    > 🛈 For SUSE 12-based systems only. If --enable-gui is part of your configuration, do the following:
    >
    > ```
    > $ cd /usr/lib64
    > $ ln -s libXext.so.6.4.0 libXext.so
    > $ ln -s libXss.so.1 libXss.so
    > ```
    >
    > When finished, cd back to your install directory.

    See for more information on which options are available to customize the ./configure command.

3. Run each of the following commands in order.

```
[root]# ./configure --enable-cgroups --with-hwloc-path=/usr/local # add any other
specified options
[root]# make
[root]# make install
```

4. Source the appropriate profile file to add `/usr/local/bin` and `/usr/local/sbin` to your path.

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

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

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

6. Add nodes to the `/var/spool/torque/server_priv/nodes` file. See Specifying Compute Nodes on page 25 for information on syntax and options for specifying compute nodes.

7. Configure pbs_server to start automatically at system boot, and then start the daemon.

    - Red Hat 6-based systems

```
[root]# chkconfig --add pbs_server
[root]# service pbs_server restart
```

    - Red Hat 7-based systems

```
[root]# qterm
[root]# systemctl enable pbs_server.service
[root]# systemctl start pbs_server.service
```

    - SUSE 11-based systems

```
[root]# chkconfig --add pbs_server
[root]# service pbs_server restart
```

    - SUSE 12-based systems

```
[root]# qterm
[root]# systemctl enable pbs_server.service
[root]# systemctl start pbs_server.service
```

Install Torque MOMs

In most installations, you will install a Torque MOM on each of your compute nodes.

> ⓘ See Specifying Compute Nodes or Configuring Torque on Compute Nodes for more information.

Do the following:

1. On the Torque Server Host, do the following:

    a. Create the self-extracting packages that are copied and executed on your nodes.

```
[root]# 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.
```

b. Copy the self-extracting MOM packages to each Torque MOM Host.

Adaptive Computing recommends that you use a remote shell, such as SSH, to install packages on remote systems. Set up shared SSH keys if you do not want to supply a password for each Torque MOM Host.

```
[root]# scp torque-package-mom-linux-x86_64.sh <mom-node>:
```

c. Copy the pbs_mom startup script to each Torque MOM Host.

- Red Hat 6-based systems

```
[root]# scp contrib/init.d/pbs_mom <mom-node>:/etc/init.d
```

- Red Hat 7-based systems

```
[root]# scp contrib/systemd/pbs_mom.service <mom-node>:/usr/lib/systemd/system/
```

- SUSE 11-based systems

```
[root]# scp contrib/init.d/suse.pbs_mom <mom-node>:/etc/init.d/pbs_mom
```

- SUSE 12-based systems

```
[root]# scp contrib/systemd/pbs_mom.service <mom-node>:/usr/lib/systemd/system/
```

> ⓘ Not all sites see an inherited ulimit but those that do can change the ulimit in the pbs_mom init script. The pbs_mom init script is responsible for starting and stopping the pbs_mom process.

2. For a Red Hat 6-based system or a SUSE 11-based system, on *each* Torque MOM Host, confirm that cgroups have been mounted; if not, mount them.

a. Run lssubsys -am.

b. If the command is not found, or you do not see something similar to the following, then cgroups are *not* mounted, continue with these instructions.

```
ns
perf_event
net_prio
cpuset /cgroup/cpuset
cpu /cgroup/cpu
cpuacct /cgroup/cpuacct
memory /cgroup/memory
devices /cgroup/devices
freezer /cgroup/freezer
net_cls /cgroup/net_cls
blkio /cgroup/blkio
```

   c. For Red Hat 6-based systems, install the cgroup library package and mount cgroups.

```
[root]# yum install libcgroup
[root]# service cgconfig start
```

   d. For SUSE 11-based systems, do the following:

      i. Install the cgroup library package.

```
[root]# zypper install libcgroup1
```

      ii. Edit /etc/cgconfig.conf and add the following:

```
mount {
        devices = /mnt/cgroups/devices;
        cpuset = /mnt/cgroups/cpuset;
        cpu = /mnt/cgroups/cpu;
        cpuacct = /mnt/cgroups/cpuacct;
        memory = /mnt/cgroups/memory;
}
```

      iii. Mount cgroups.

```
[root]# service cgconfig start
```

   e. Run lssubsys -am again and confirm cgroups are mounted.

2. 3. On *each* Torque MOM Host, do the following:

   a. For Red Hat 7-based or SUSE 12-based systems, install cgroup-tools.

      • Red Hat 7-based systems

```
[root]# yum install libcgroup-tools
```

      • SUSE 12-based systems

```
[root]# zypper install libcgroup-tools
```

   b. Install the self-extracting MOM package.

```
[root]# ./torque-package-mom-linux-x86_64.sh --install
```

c. Configure pbs_mom to start at system boot, and then start the daemon.

- Red Hat 6-based systems

```
[root]# chkconfig --add pbs_mom
[root]# service pbs_mom start
```

- Red Hat 7-based systems

```
[root]# systemctl enable pbs_mom.service
[root]# systemctl start pbs_mom.service
```

- SUSE 11-based systems

```
[root]# chkconfig --add pbs_mom
[root]# service pbs_mom start
```

- SUSE 12-based systems

```
[root]# systemctl enable pbs_mom.service
[root]# systemctl start pbs_mom.service
```

Install Torque Clients

If you want to have the Torque client commands installed on hosts other than the Torque Server Host (such as the compute nodes or separate login nodes), do the following:

1. On the Torque Server Host, do the following:

a. Copy the self-extracting client package to *each* Torque Client Host.

> ⓘ Adaptive Computing recommends that you use a remote shell, such as SSH, to install packages on remote systems. Set up shared SSH keys if you do not want to supply a password for each Torque Client Host.

```
[root]# scp torque-package-clients-linux-x86_64.sh <torque-client-host>:
```

b. Copy the trqauthd startup script to *each* Torque Client Host.

- Red Hat 6-based systems

```
[root]# scp contrib/init.d/trqauthd <torque-client-host>:/etc/init.d
```

- Red Hat 7-based systems

```
[root]# scp contrib/systemd/trqauthd.service <torque-client-host>:/usr/lib/systemd/system/
```

- SUSE 11-based systems

```
[root]# scp contrib/init.d/suse.trqauthd <torque-client-
host>:/etc/init.d/trqauthd
```

- SUSE 12-based systems

```
[root]# scp contrib/systemd/trqauthd.service <torque-client-
host>:/usr/lib/systemd/system/
```

2. On *each* Torque Client Host, install the self-extracting client package:

```
[root]# ./torque-package-clients-linux-x86_64.sh --install
```

Configure Data Management

When a batch job completes, stdout and stderr files are generated and placed in the spool directory on the master Torque MOM Host for the job instead of the submit host. You can configure the Torque batch environment to copy the stdout and stderr files back to the submit host. See Configuring Data Management for more information.

Compute Nodes

Use the Adaptive Computing Torque package system to create self-extracting tarballs which can be distributed and installed on compute nodes. The Torque packages are customizable. See the `INSTALL` file for additional options and features.

> ℹ If you installed Torque using the RPMs, you must install and configure your nodes manually by modifying the `/var/spool/torque/mom_priv/config` file of each one. This file is identical for all compute nodes and can be created on the head node and distributed in parallel to all systems.
>
> ```
> [root]# vi /var/spool/torque/mom_priv/config
>
> $pbsserver      headnode     # hostname running pbs server
> $logevent       225          # bitmap of which events to log
>
> [root]# service pbs_mom restart
> ```

To create Torque packages

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 Torque packages on the compute nodes.

   Adaptive Computing recommends that you use a remote shell, such as SSH, to install Torque packages 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
> for i in node01 node02 node03 node04 ; do ssh ${i} ldconfig ; done
```

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

To use a tool like xCAT

1. Copy the Torque package to the nodes.

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

2. Install the Torque package.

```
> 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

Enabling Torque as a Service

> ℹ️ Enabling Torque as a service is optional. In order to run Torque as a service, you must enable `trqauthd`. (see Configuring trqauthd for Client Commands).

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/` or `contrib/systemd/` directory of the source package. To enable Torque as a service, run the following *as root* on the host for the appropriate Torque daemon:

- Red Hat 6-based systems

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

- SUSE 11-based systems

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

- Red Hat 7-based or SUSE 12-based systems

```
> cp contrib/systemd/pbs_mom.service /user/lib/systemd/pbs_server.service
> systemctl enable pbs_mom.service
> cp contrib/systemd/pbs_server.service /user/lib/systemd/pbs_server.service
> systemctl enable pbs_server.service
```

> ℹ️ 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

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). It also maintains the status of each MOM node through updates from the MOMs in the cluster (see pbsnodes). All jobs are submitted via qsub to the server, which maintains a master database of all jobs and their states.

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

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

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

- As root, execute `./torque.setup` from the build directory (see ./torque.setup).
- Use `pbs_server` -t create (see pbs_server -t create).

Restart `pbs_server` after initializing `serverdb`.

- Red Hat 6-based or SUSE 11-based systems

  ```
  > qterm
  > service pbs_server start
  ```

- Red Hat 7-based or SUSE 12-based systems

  ```
  > qterm
  > systemctl start pbs_server.service
  ```

./torque.setup

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

`/torque.setup` *username*

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

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

pbs_server -t create

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

```
> pbs_server -t create
```

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

```
> qmgr -c 'p s'

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

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

This section contains these topics:

- Specifying Compute Nodes
- Configuring Torque on Compute Nodes
- Finalizing Configurations

Related Topics

Specifying Compute Nodes

The environment variable *TORQUE_HOME* holds the directory path where configuration files are stored. If you used the default locations during installation, you do not need to specify the *TORQUE_HOME* environment variable.

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

> **ⓘ** Only a root user can access the `server_priv` directory.

Syntax of nodes file:

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

- The **node-name** must match the hostname on the node itself, including whether it is fully qualified or shortened.

- The **[:ts]** option marks the node as timeshared. Timeshared nodes are listed by the server in the node status report, but the server does not allocate jobs to them.

- The **[np=]** option specifies the number of virtual processors for a given node. The value can be less than, equal to, or greater than the number of physical processors on any given node.

- The **[gpus=]** option specifies the number of GPUs for a given node. The value can be less than, equal to, or greater than the number of physical GPUs on any given node.

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

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

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

- The **[properties]** option allows you to specify arbitrary strings to identify the node. Property strings are alphanumeric characters only and must begin with an alphabetic character.

- Comment lines are allowed in the nodes file if the first non-white space character is the pound sign (**#**).

The following example shows a possible node file listing.

`TORQUE_HOME/server_priv/nodes`:

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

Related Topics

Initializing/Configuring Torque on the Server (pbs_server)

Configuring Torque on Compute Nodes

If you are 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`:

```
$logevent      1039           # 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)

Configuring Ports

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

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

> ℹ️ If you are running PBS Professional 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.
  - Red Hat 6-based or SUSE 11-based systems
    - Edit `/etc/sysconfig/pbs_server`.

      ```
      PBS_ARGS="-p"
      ```

    - Start the service.

      ```
      service pbs_server start
      ```

  - Red Hat 7-based or SUSE 12-based systems
    - Edit `/etc/systemconfig/pbs_server`.

      ```
      PBS_ARGS="-p"
      ```

    - Start the service.

      ```
      systemctl start pbs_server.service
      ```

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

- Start `pbs_server` with the `-H` option.
  - Red Hat 6-based or SUSE 11-based systems
    - Edit `/etc/sysconfig/pbs_server`.
      ```
      PBS_ARGS="-H"
      ```
    - Start the service.
      ```
      service pbs_server start
      ```
  - Red Hat 7-based or SUSE 12-based systems
    - Edit `/etc/systemconfig/pbs_server`.
      ```
      PBS_ARGS="-H"
      ```
    - Start the service.
      ```
      systemctl start pbs_server.service
      ```

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.
  - Red Hat 6-based or SUSE 11-based systems
    - Edit `/etc/sysconfig/pbs_mom`.
      ```
      PBS_ARGS="-M"
      ```
    - Start the service.
      ```
      service pbs_mom start
      ```
  - Red Hat 7-based or SUSE 12-based systems
    - Edit `/etc/systemconfig/pbs_mom`.
      ```
      PBS_ARGS="-M"
      ```
    - Start the service.
      ```
      systemctl start pbs_mom.service
      ```
- Edit the pbs_server `nodes` file to 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.
  - Red Hat 6-based or SUSE 11-based systems
    - Edit `/etc/sysconfig/pbs_mom`.
      ```
      PBS_ARGS="-M"
      ```
    - Start the service.
      ```
      service pbs_mom start
      ```
  - Red Hat 7-based or SUSE 12-based systems
    - Edit `/etc/systemconfig/pbs_mom`.
      ```
      PBS_ARGS="-M"
      ```
    - Start the service.
      ```
      systemctl start pbs_mom.service
      ```

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.
  - Red Hat 6-based or SUSE 11-based systems
    - Edit `/etc/sysconfig/pbs_mom`.
      ```
      PBS_ARGS="-S"
      ```

○ Start the service.

```
service pbs_mom start
```

○ Red Hat 7-based or SUSE 12-based systems

○ Edit `/etc/systemconfig/pbs_mom`.

```
PBS_ARGS="-S"
```

○ Start the service.

```
systemctl start pbs_mom.service
```

- 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`, do the following:

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

Changing Default Ports

This section provides examples of changing the default ports (using non-standard ports).

In this section:

MOM Service Port

The MOM service port is the port number on which MOMs are listening. This example shows how to change the default MOM service port (15002) to port

30001.

Do the following:

1. On the server, for the server_priv/nodes file, change the node entry.

```
nodename np=4 mom_service_port=30001
```

2. On the MOM, start `pbs_mom` with the `-M` option.

- Red Hat 6-based or SUSE 11-based systems

  - Edit `/etc/sysconfig/pbs_mom`.

    ```
    PBS_ARGS="-M"
    ```

  - Start the service.

    ```
    service pbs_mom start
    ```

- Red Hat 7-based or SUSE 12-based systems

  - Edit `/etc/systemconfig/pbs_mom`.

    ```
    PBS_ARGS="-M"
    ```

  - Start the service.

    ```
    systemctl start pbs_mom.service
    ```

Default Port on the Server

Do the following:

1. Set the $(TORQUE_HOME)/server_name file.

```
hostname:newport
numa3.ac:45001
```

2. On the MOM, start `pbs_mom` with the `-S` option.

- Red Hat 6-based or SUSE 11-based systems

  - Edit `/etc/sysconfig/pbs_mom`.

    ```
    PBS_ARGS="-S"
    ```

  - Start the service.

    ```
    service pbs_mom start
    ```

- Red Hat 7-based or SUSE 12-based systems

○ Edit `/etc/systemconfig/pbs_mom.`

```
PBS_ARGS="-S"
```

○ Start the service.

```
systemctl start pbs_mom.service
```

MOM Manager Port

The MOM manager port tell MOMs which ports on which other MOMs are listening for MOM-to-MOM communication. This example shows how to change the default MOM manager port (15003) to port 30002.

Do the following:

1. On the server nodes file.

```
nodename np=4 mom_manager_port=30002
```

2. On the MOM, start `pbs_mom` with the `-R` option.

- Red Hat 6-based or SUSE 11-based systems

   ○ Edit `/etc/sysconfig/pbs_mom.`

```
PBS_ARGS="-R"
```

   ○ Start the service.

```
service pbs_mom start
```

- Red Hat 7-based or SUSE 12-based systems

   ○ Edit `/etc/systemconfig/pbs_mom.`

```
PBS_ARGS="-R"
```

   ○ Start the service.

```
systemctl start pbs_mom.service
```

Related Topics

Initializing/Configuring Torque on the Server (pbs_server)
pbs_server
pbs_mom
trqauthd
client commands

Configuring trqauthd for Client Commands

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

Running trqauthd

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

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

trqauthd can be invoked directly from the command line or by the use of scripts which are located in the Torque source tree.

- For Red Hat 6-based or SUSE 11-based systems, the init.d scripts are located in the contrib/init.d directory
- For Red Hat 7-based or SUSE 12-based systems, the systemd scripts are located in the contrib/systemd directory

There are two scripts for trqauthd:

Script	Description
suse.trqauthd	Used only 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.

Do the following:

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

2. Restart the service.

   - Red Hat 6-based or SUSE 11-based systems

     ```
     service trqauthd restart
     ```

- Red Hat 7-based or SUSE 12-based systems

```
systemctl restart trqauthd.service
```

> ℹ️ If you receive an error that says "Could not open socket in trq_simple_
> connect. error 97" and you use a CentOS, RedHat, or Scientific Linux 6+
> operating system, check your `/etc/hosts` file for multiple entries of a
> single host name pointing to the same IP address. Delete the duplicate(s),
> save the file, and launch trqauthd again.

Related Topics

[Initializing/Configuring Torque on the Server (pbs_server)](#)

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:
  - Red Hat 6-based or SUSE 11-based sysytems

    ```
    > service pbs_mom start
    ```

  - Red Hat 7-based or SUSE 12-based systems

    ```
    > systemctl start pbs_mom.service
    ```

- Server Node:
  - Red Hat 6-based or SUSE 11-based sysytems

    ```
    > service pbs_server restart
    ```

  - Red Hat 7-based or SUSE 12-based systems

    ```
    > systemctl restart pbs_server.service
    ```

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)](#)

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
- Server Configuration

Related Topics

Server Parameters

Customizing the Install

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

- By default, Torque does not install the admin manuals. To enable this, use --enable-docs.
- By default, only children MOM processes use syslog. To enable syslog for all of Torque, use --enable-syslog.

Table 2-1: Optional Features

Option	Description
--disable-clients	Directs Torque not to build and install the Torque client utilities such as qsub, qstat, qdel, etc.
--disable-FEATURE	Do not include FEATURE (same as --enable-FEATURE=no).
--disable-libtool-lock	Avoid locking (might break parallel builds).
--disable-mom	Do not include the MOM daemon.
--disable-mom-check-spool	Don't check free space on spool directory and set an error.
--disable-posixmemlock	Disable the MOM's use of mlockall. Some versions of OSs seem to have buggy POSIX MEMLOCK.

Option	Description
--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-net-working	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.
--enable-autorun	Turn on the AUTORUN_JOBS flag. When enabled, Torque runs the jobs as soon as they are submitted (destroys Moab compatibly). This option is not supported.
--enable-blcr	Enable BLCR support.
--enable-cgroups	Enable cgroups. When cgroups are enabled, cpusets are handled by the cgroup cpuset subsystem. ⓘ If you are building with cgroups enabled, you must have boost version 1.41 or later.
--enable-cpa	Enable Cray's CPA support.

Option	Description
--enable-cpu-set	Enable Linux 2.6 kernel cpusets. ℹ️ It is recommended that you use --enable-cgroups instead of --enable-cpuset. --enable-cpuset is deprecated and no new features will be added to it. --enable-cgroups and --enable-cpuset are mutually exclusive.
--enable-debug	Prints debug information to the console for pbs_server and pbs_mom while they are running. (This is different than **--with-debug** which will compile with debugging symbols.)
--enable-dependency-tracking	Do not reject slow dependency extractors.
--enable-fast-install[=PKGS]	Optimize for fast installation [default=yes].
--enable-FEATURE [=ARG]	Include FEATURE [ARG=yes].
--enable-file-sync	Open files with sync on each write operation. This has a negative impact on Torque performance. This is disabled by default.
--enable-force-nodefile	Forces creation of nodefile regardless of job submission parameters. Not on by default.
--enable-gcc-warnings	Enable gcc strictness and warnings. If using gcc, default is to error on any warning.
--enable-geo-metry-requests	Torque is compiled to use procs_bitmap during job submission. ℹ️ When using --enable-geometry-requests, do not disable cpusets. Torque looks at the cpuset when killing jobs.
--enable-gui	Include the GUI-clients.
--enable-main-tainer-mode	This is for the autoconf utility and tells autoconf to enable so called rebuild rules. See maintainer mode for more information.

Option	Description
--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-dae-mons[=ARG]	Enable daemons to lock themselves into memory: logical-or of 1 for pbs_server, 2 for pbs_scheduler, 4 for pbs_mom (no argument means 7 for all three).
--enable-quick-commit	Turn on the QUICKCOMMIT flag. When enabled, adds a check to make sure the job is in an expected state and does some bookkeeping for array jobs. This option is not supported.
--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.
--enable-static [=PKGS]	Build static libraries [default=yes].

Option	Description
--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 2-2: 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).

Option	Description

--with-boost-path=DIR	ⓘ Boost version 1.36.0 or later is supported. Red Hat 6-based systems come packaged with 1.41.0 and Red Hat 7-based systems come packaged with 1.53.0.
	
	Set the path to the Boost header files to be used during make. This option does not require Boost to be built or installed.
	
	The --with-boost-path value must be a directory containing a sub-directory called boost that contains the boost .hpp files.
	
	For example, if downloading the boost 1.55.0 source tarball to the adaptive user's home directory:
	
	`[adaptive]$ cd ~`
	`[adaptive]$ wget`
	`http://sourceforge.net/projects/boost/files/boost/1.55.0/boost_1_55_`
	`0.tar.gz/download`
	`[adaptive]$ tar xzf boost_1_55_0.tar.gz`
	`[adaptive]$ ls boost_1_55_0`
	`boost`
	`boost-build.jam`
	`...`
	
	In this case use --with-boost-path=/home/adaptive/boost_1_55_0 during configure.
	
	Another example would be to use an installed version of Boost. If the installed Boost header files exist in /usr/include/boost/*.hpp, use --with-boost-path=/usr/include.

--with-cpa-include=DIR	Include path for cpalib.h.
--with-cpa-lib=DIR	Lib path for libcpalib.
--with-debug=no	Do not compile with debugging symbols.
--with-default-server-r=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].

Advanced Configuration

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 full path of a remote file copy program. scp is the default if found, otherwise mom_rcp is used. Some rcp programs don't always exit with valid error codes in case of failure. mom_rcp is a copy of BSD rcp included with this source that has correct error codes, but it is also old, unmaintained, and doesn't have large file support.
--with-sched=TYPE	Sets the scheduler type. If TYPE is "c", the scheduler will be written in C. If TYPE is "tcl" the server will use a Tcl based scheduler. If TYPE is "basl", Torque will use the rule based scheduler. If TYPE is "no", then no scheduling is done. "c" is the default.

Option	Description
--with-sched-code=PATH	Sets the name of the scheduler to use. This only applies to BASL schedulers and those written in the C language. For C schedulers this should be a directory name and for BASL schedulers a filename ending in ".basl". It will be interpreted relative to srctree/src/schedulers.SCHD_TYPE/samples. As an example, an appropriate BASL scheduler relative path would be "nas.basl". The default scheduler code for "C" schedulers is "fifo".
--with-scp	In Torque 2.1 and later, SCP is the default remote copy protocol. See **--with-rcp** if a different protocol is desired.
--with-sendmail[=FILE]	Sendmail executable to use.
--with-server-home=DIR	Set the server home/spool directory for PBS use. Defaults to /var/spool/torque.
--with-server-name-file-e=FILE	Set the file that will contain the name of the default server for clients to use. If this is not an absolute pathname, it will be evaluated relative to the server home directory that either defaults to /usr/spool/PBS or is set using the --with-server-home option to configure. If this option is not specified, the default name for this file will be set to "server_name".
--with-tcl	Directory containing tcl configuration (tclConfig.sh).
--with-tclatrsep=CHAR	Set the Tcl attribute separator character this will default to "." if unspecified.
--with-tclinclude	Directory containing the public Tcl header files.
--with-tclx	Directory containing tclx configuration (tclxConfig.sh).
--with-tk	Directory containing tk configuration (tkConfig.sh).
--with-tkinclude	Directory containing the public Tk header files.
--with-tkx	Directory containing tkx configuration (tkxConfig.sh).
--with-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

Server Configuration

Server Configuration

This topic contains information and instructions to configure your server.

In this topic:

- Server Configuration Overview
- Name Service Configuration
- Configuring Job Submission Hosts
- Configuring Torque on a Multi-Homed Server
- Architecture Specific Notes
- Specifying Non-Root Administrators
- Setting Up Email
- Using MUNGE Authentication

Also see Setting Up the MOM Hierarchy (Optional)

Server Configuration Overview

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

Name Service Configuration

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

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

Configuring Job Submission Hosts

Using RCmd authentication

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

/etc/hosts.equiv:

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

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

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

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

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

These lines will work:

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

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

```
node01 -tsmith
node01 +
```

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

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'
```

> ℹ️ 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.

> ℹ️ 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")).

Using MUNGE Authentication

> ℹ The same version on MUNGE must be installed on all of your Torque Hosts (Server, Client, MOM).

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

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

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

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

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

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

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

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

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

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

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

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

Related Topics

Setting Up the MOM Hierarchy (Optional)
Advanced Configuration

Setting Up the MOM Hierarchy (Optional)

> ⓘ Mom hierarchy is designed for large systems to configure how information is passed directly to the pbs_server.

The MOM hierarchy allows you to override the compute nodes' default behavior of reporting status updates directly to the pbs_server. Instead, you configure compute nodes so that each node sends its status update information to another compute node. The compute nodes pass the information up a tree or hierarchy until eventually the information reaches a node that will pass the information directly to pbs_server. This can significantly reduce network traffic and ease the load on the pbs_server in a large system.

> ℹ Adaptive Computing recommends approximately 25 nodes per path.
> Numbers larger than this may reduce the system performance.

MOM Hierarchy Example

The following example illustrates how information is passed to the pbs_server without and with mom_hierarchy.

Without mom_hierarchy
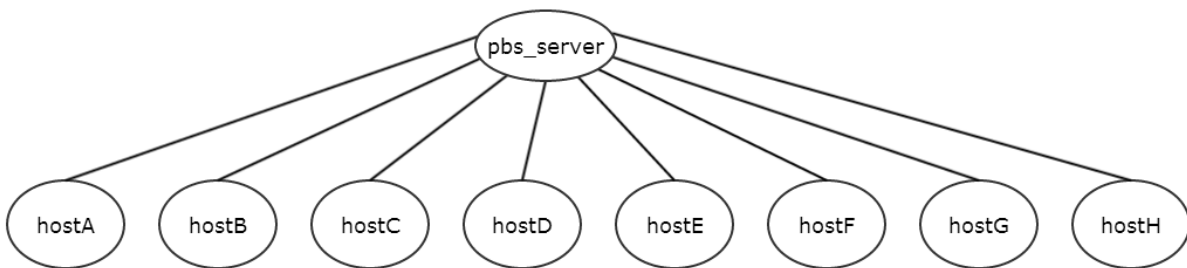
With mom_hierarchy
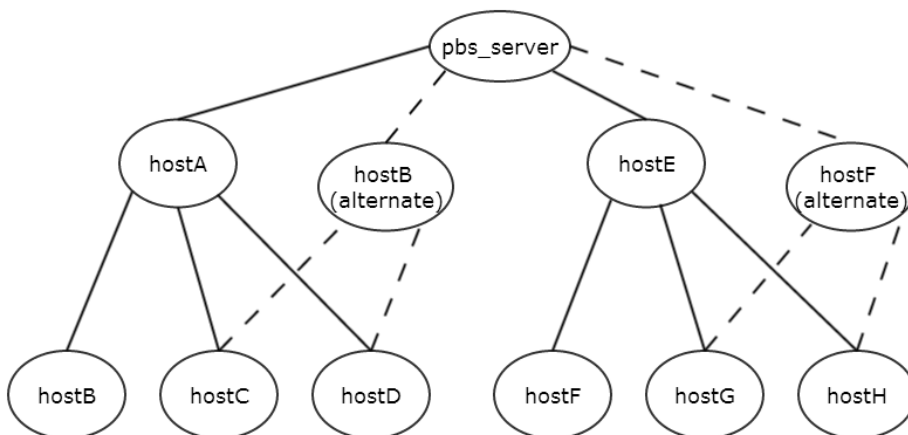
> ℹ The dotted lines indicates an alternate path if the hierarchy-designated node goes down.

The following is the mom_hierachy_file for the with mom_hierarchy example:

```
<path>
  <level>hostA,hostB</level>
  <level>hostB,hostC,hostD</level>
</path>
<path>
  <level>hostE,hostF</level>
  <level>hostE,hostF,hostG</level>
</path>
```

Setting Up the MOM Hierarchy

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

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

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

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

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

> ℹ️ If you want to specify MOMs on a different port than the default, you must list the node in the form: `hostname:mom_manager_port`.
>
> For example:
> ```
> <path>
>   <level>hostname:mom_manager_port,... </level>
>   ...
> </path>
> ...
> ```

Putting the MOM Hierarchy on the MOMs

You can put the MOM hierarchy file directly on the MOMs. The default location is `/var/spool/torque/mom_priv/mom_hierarchy`. This way, the pbs_server

doesn't have to send the hierarchy to all the MOMs during each pbs_server startup. The hierarchy file still has to exist on the pbs_server and if the file versions conflict, the pbs_server version overwrites the local MOM file. When using a global file system accessible from both the MOMs and the pbs_server, it is recommended that the hierarchy file be symbolically linked to the MOMs.

Once the hierarchy file exists on the MOMs, start pbs_server with the -n option which tells pbs_server to not send the hierarchy file on startup. Instead, pbs_server waits until a MOM requests it.

Opening Ports in a Firewall

If your site is running firewall software on its hosts, you will need to configure the firewall to allow connections to the products in your installation.

This topic provides an example and general instructions for how to open ports in your firewall. The actual port numbers for the various products will be provided in the installation instructions for that product.

In this topic:

Red Hat 6-Based Systems

Red Hat 6-based systems use iptables as the default firewall software. For the ip6tables service, replace all occurrences of iptables with ip6tables in the example. If you use different firewall software, refer to your firewall documentation for opening ports in your firewall.

The following is an example of adding port 1234 when using iptables.

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

[root]# vi /tmp/iptables.mod

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

-A INPUT -p tcp --dport 1234 -j ACCEPT

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

Red Hat 7-Based Systems

Red Hat 7-based systems use firewalld as the default firewall software. If you use different firewall software, refer to your firewall documentation for opening ports in your firewall.

The following is an example of adding port 1234 when using firewalld.

```
[root]# firewall-cmd --add-port=1234/tcp --permanent
[root]# firewall-cmd --reload
```

SUSE 11-Based Systems

SUSE 11-based systems use SuSEfirewall2 as the default firewall software. If you use different firewall software, refer to your firewall documentation for opening ports in your firewall.

The following is an example of adding port 1234 when using SuSEfirewall2.

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

FW_SERVICES_EXT_TCP="1234"

[root]# service SuSEfirewall2_setup restart
```

SUSE 12-Based Systems

SUSE 12-based systems use SuSEfirewall2 as the default firewall software. If you use different firewall software, refer to your firewall documentation for opening ports in your firewall.

The following is an example of adding port 1234 when using SuSEfirewall2.

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

FW_SERVICES_EXT_TCP="1234"

[root]# service SuSEfirewall2 restart
```

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 `$TORQUE_HOME`/`server_priv` named `serverdb` which contains the server configuration information.

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

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

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

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

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

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

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

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

Related Topics

Torque Installation Overview

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. See these topics for details:

- Basic Node Specification
- Specifying Virtual Processor Count for a Node
- Specifying GPU Count for a Node
- Specifying Node Features (Node Properties)

Related Topics

Torque Installation Overview
Server Parameters
Node Features/Node Properties

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
```

> ℹ The server `nodes` file also displays the parameters applied to the node. See Adding nodes for more information on the parameters.

Related Topics

Server Node File Configuration

Specifying Virtual Processor Count for a Node

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

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

`server_priv/nodes:`

```
node001 np=4
node002 np=12
 ...
```

Related Topics

[Server Node File Configuration](#)

Specifying GPU Count for a Node

Administrators can manually set the number of GPUs on a node or if they are using NVIDIA GPUs and drivers, they can have them detected automatically. For more information about how to set up Torque with GPUS, see Accelerators in the *Moab Workload Manager Administrator Guide*.

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](#)

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

Testing Server Configuration

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

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

server:kmn

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

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

# verify all nodes are correctly reporting
> pbsnodes -a
node001
  state=free
  np=2
  properties=bigmem,fast,ia64,smp
  ntype=cluster
  status=rectime=1328810402,varattr=,jobs=,state=free,netload=6814326158,gres=,loadave
=0.21,ncpus=6,physmem=8193724kb,
availmem=13922548kb,totmem=16581304kb,idletime=3,nusers=3,nsessions=18,sessions=1876
1120 1912 1926 1937 1951 2019 2057 28399 2126 2140 2323 5419 17948 19356 27726 22254
29569,uname=Linux kmn 2.6.38-11-generic #48-Ubuntu SMP Fri Jul 29 19:02:55 UTC 2011
x86_64,opsys=linux
  mom_service_port = 15002
  mom_manager_port = 15003
  gpus = 0
# submit a basic job - DO NOT RUN AS ROOT
> su - testuser
> echo "sleep 30" | qsub

# verify jobs display
> qstat

Job id    Name    User       Time Use  S  Queue
------    -----   ----       --------  -- -----
```

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

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

Related Topics

Torque Installation Overview

Configuring Torque for NUMA Systems

Torque supports these two types of Non-Uniform Memory Architecture (NUMA) systems:

- NUMA-Aware – For Torque 6.0 and later, supports multi-req jobs and jobs that span hosts. Requires the `--enable-cgroups` configuration command to support cgroups. See Torque NUMA-Aware Configuration on page 57 for instructions and additional information.
- NUMA-Support – For Torque version 3.0 and later; *only* for large-scale SLES systems (SGI Altix and UV hardware). Requires the `--enable-numa-support` configuration command. See Torque NUMA-Support Configuration on page 60 for instructions and additional information.

⚠ Torque cannot be configured for both systems at the same.

Related Topics

- Torque NUMA-Aware Configuration on page 57
- Torque NUMA-Support Configuration on page 60

Torque NUMA-Aware Configuration

This topic provides instructions for enabling NUMA-aware, including cgroups, and requires Torque 6.0 or later. For instructions on NUMA-support configurations, see Torque NUMA-Support Configuration on page 60. This topic assumes you have a basic understanding of cgroups. See RedHat Resource Management Guide (https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/ch01.html) or cgroups on kernel.org (https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt) for basic information on cgroups.

Torque uses cgroups to better manage cpu and memory accounting, memory enforcement, cpuset management, and binding jobs to devices such as MICs and GPUs.

> ℹ **Be aware of the following:**
>
> - If you are building with cgroups enabled, you must have boost version 1.41 or later.
>
> - The pbs_mom daemon is the binary that interacts cgroups, but both the server and the MOM must be built with `--enable-cgroups` to understand all of the new structures.
>
> - Beginning with Torque 6.0.2, Cray-enabled Torque may be configured with cgroups. On the login node, each job will have all of the cpus and all of the memory controllers in it's cgroup.

Prerequisites

1. Install the prerequisites found in [Installing Torque Resource Manager](#).

2. hwloc version 1.9.1 or later is required. Version 1.11.0 is needed if installing with NVIDIA K80 or newer GPU hardware

   - download hwloc-1.9.1.tar.gz from: https://www.open-mpi.org/software/hwloc/v1.9

   - perform the following command line actions:

     ```
     $ tar -xzvf hwloc-1.9.1.tar.gz
     $ cd hwloc-1.9.1.tar.gz
     $ sudo ./configure
     ```

   - You do not need to overwrite the default installation of hwloc. By default hwloc will install to the /usr/local directory. You can also configure hwloc with the --prefix option to have it install to a location of your choosing. If you do not install hwloc to /usr directory you can tell Torque where to find the version you want it to use at configure time using the --with-hwloc-path option. For example:

     ```
     ./configure --enable-cgroups --with-hwloc-path=/usr/local
     ```

   - Run make

   - sudo make install

Installation Instructions

Do the following:

1. Install the libcgroup package.

    > ℹ Red Hat-based Systems must use libcgroup version 0.40.rc1-16.el6 or later; SUSE-based systems need to use a comparative libcgroup version.

    - Red Hat 6-based systems

    ```
    yum install libcgroup
    ```

    - Red Hat 7-based systems

    ```
    yum install libcgroup-tools
    ```

    - SUSE 11-based systems

    ```
    zypper install libcgroup1
    ```

    - SUSE 12-based systems

    ```
    zypper install libcgroup-tools
    ```

2. Enable Torque to access cgroups.

    ```
    $ ./configure --enable-cgroups --with-hwloc-path=/usr/local
    ```

3. For a Red Hat 6-based system or a SUSE 11-based system, on *each* Torque MOM Host, confirm that cgroups have been mounted; if not, mount them.

    a. Run lssubsys -am.

    b. If the command is not found, or you do not see something similar to the following, then cgroups are *not* mounted, continue with these instructions.

    ```
    ns
    perf_event
    net_prio
    cpuset /cgroup/cpuset
    cpu /cgroup/cpu
    cpuacct /cgroup/cpuacct
    memory /cgroup/memory
    devices /cgroup/devices
    freezer /cgroup/freezer
    net_cls /cgroup/net_cls
    blkio /cgroup/blkio
    ```

    c. For Red Hat 6-based systems, install the cgroup library package and mount cgroups.

    ```
    [root]# yum install libcgroup
    [root]# service cgconfig start
    ```

d. For SUSE 11-based systems, do the following:

   i. Install the cgroup library package.

```
[root]# zypper install libcgroup1
```

   ii. Edit /etc/cgconfig.conf and add the following:

```
mount {
        devices = /mnt/cgroups/devices;
        cpuset = /mnt/cgroups/cpuset;
        cpu = /mnt/cgroups/cpu;
        cpuacct = /mnt/cgroups/cpuacct;
        memory = /mnt/cgroups/memory;
}
```

   iii. Mount cgroups.

```
[root]# service cgconfig start
```

e. Run lssubsys -am again and confirm cgroups are mounted.

Multiple cgroup Directory Configuration

If your system has more than one cgroup directory configured, you must create the `trq-cgroup-paths` file in the $TORQUE_HOME directory. This file has a list of the cgroup subsystems and the mount points for each subsystem in the syntax of *<subsystem> <mount point>*.

All five subsystems used by pbs_mom must be in the `trq-cgroup-paths` file. In the example that follows, a directory exists at /cgroup with subdirectories for each subsystem. Torque uses this file first to configure where it will look for cgroups.

```
cpuset   /cgroup/cpuset
cpuacct /cgroup/cpuacct
cpu      /cgroup/cpu
memory   /cgroup/memory
devices /cgroup/devices
```

Torque NUMA-Support Configuration

> ⓘ This topic provides instructions for enabling NUMA-support on large-scale SLES systems using SGI Altix and UV hardware and requires Torque 3.0 or later. For instructions on enabling NUMA-aware, see Torque NUMA-Aware Configuration on page 57.

Do the following in order:

- Configure Torque for NUMA-Support
- Create the mom.layout File

- Configure the server_priv/nodes File
- Limit Memory Resources (Optional)

Configure Torque for 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.

Create the mom.layout File

This section provides instructions to create the mom.layout file.

Do *one* of the following:

Automatically Create 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, do the following:

1. Verify `hwloc` library and corresponding hwloc-devel package are installed. See Installing Torque Resource Manager for more information.

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.

```
$ ./mom_gencfg
```

Manually Create 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.

Configure the server_priv/nodes File

The pbs_server requires awareness of how the MOM is reporting nodes since there is only one MOM daemon and multiple MOM nodes.

You need to 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.

The following is an example of how to configure the nodes file with num_node_boards.

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

In this example, 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.

The previous example showed that the NUMA system is uniform in its configuration of CPUs per node board. However, 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 example, pbs_server knows it has 3 MOM nodes and the nodes have 6, 8, and 12 CPUs respectively. Notice that the attribute np is not used. The np attribute is ignored because the number of CPUs per node is expressly given.

Limit Memory Resources (Optional)

Torque can better enforce memory limits with the use of the memacctd utility. The memacctd utility is a daemon that caches memory footprints when it is queried. When configured to use the memory monitor, Torque queries memacctd.

> ℹ The memacctd utility is provided by SGI for SLES systems only. 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, do the following:

1. Start **memacctd** as instructed by SGI.

2. Reconfigure Torque with `--enable-memacct`. This will link in the necessary library when Torque is recompiled.

3. Recompile and reinstall Torque.

4. Restart all MOM nodes.

> ℹ You use the qsub filter to include a default memory limit for all jobs that are not submitted with memory limit.

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
- Stopping pbs_mom in Multi-MOM Mode

Multi-MOM Configuration

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

1. Configure server_priv/nodes
2. /etc/hosts file
3. Starting pbs_mom with Multi-MOM Options

Configure server_priv/nodes

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

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

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

/etc/hosts file

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

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

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

Starting pbs_mom with Multi-MOM Options

To start multiple instances of `pbs_mom` on the same node, use the following syntax (see pbs_mom 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

Stopping pbs_mom in Multi-MOM Mode

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

Multi-MOM Configuration

Chapter 3 Submitting and Managing Jobs

This section contains information about how you can submit and manage jobs with Torque. See the following topics for details:

- Job Submission
- Monitoring Jobs
- Canceling Jobs
- Job Preemption
- Keeping Completed Jobs
- Job Checkpoint and Restart
- Job Exit Status
- Service Jobs

Job Submission

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

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

```
#PBS -S /bin/bash
#PBS -m abe
#PBS -M <yourEmail@company.com>
echo sleep 300
```

This is an example of properly declared PBS directives.

```
#PBS -S /bin/bash
SOMEVARIABLE=42
#PBS -m abe
#PBS -M <yourEmail@company.com>
echo sleep 300
```

This is an example of improperly declared PBS directives. PBS directives below "SOMEVARIABLE=42" are ignored.

ⓘ By default, job submission is allowed only on the Torque server host (host on which **pbs_server** is running). Enablement of job submission from other hosts is documented in Server Configuration.

ⓘ 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
- Requesting Resources
- Requesting Generic Resources
- Requesting Floating Resources
- Requesting Other Resources
- Exported Batch Environment Variables
- Enabling Trusted Submit Hosts
- Example Submit Scripts

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") – Allow local checking and modification of submitted job

Multiple Job Submission

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

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

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

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

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

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

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

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

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

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

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

Slot Limit

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

Related Topics

Job Submission

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 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 3-1: Job radix example
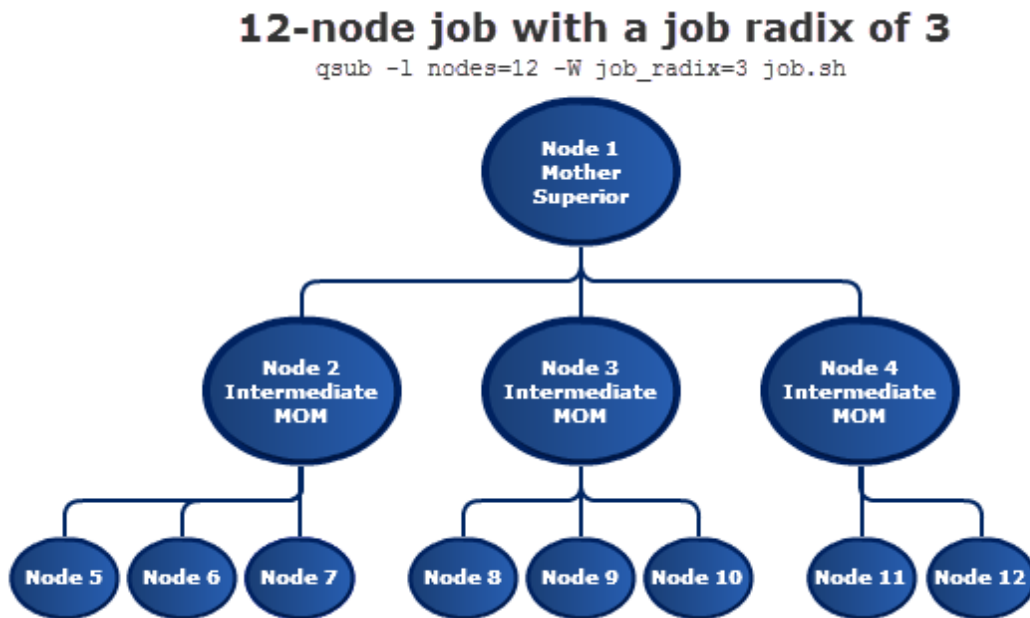
The Mother Superior picks three sister nodes with which to communicate the job information. Each of those nodes (intermediate MOMs) receives a list of all sister nodes that will be subordinate to it. They each contact up to three nodes and pass the job information on to those nodes. This pattern continues until the bottom level is reached. All communication is now passed across this new hierarchy. The stdout and stderr data is aggregated and sent up the tree until it reaches the Mother Superior, where it is saved and copied to the .o and .e files.

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

Requesting Resources

Various resources can be requested at the time of job submission. A job can request a particular node, a particular node attribute, or even a number of nodes with particular attributes. Either native Torque resources or external scheduler resource extensions may be specified.

qsub -l supports:

- All the native Torque resources. See Native Torque Resources on page 74 for a list of resources.

- Some Moab scheduler job extensions (for legacy support). See Moab Job Extensions on page 82 for a list of resources.

> ℹ️ For Moab resource extensions, "qsub -W x=" is recommended instead of "qsub -l". See Resource Manager Extensions in the *Moab Workload ManagerAdministrator Guide* for a complete list of scheduler-only job extensions.

Native Torque Resources

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.

Resource	Format	Description
cpuclock	string	Specify the CPU clock frequency for each node requested for this job. A **cpuclock** request applies to every processor on every node in the request. Specifying varying CPU frequencies for different nodes or different processors on nodes in a single job request is not supported.
		Not all processors support all possible frequencies or ACPI states. If the requested frequency is not supported by the CPU, the nearest frequency is used.
		ALPS 1.4 or later is required when using **cpuclock** on Cray.
		The clock frequency can be specified via:
		• a number that indicates the clock frequency (with or without the SI unit suffix).

```
qsub -l cpuclock=1800,nodes=2 script.sh
qsub -l cpuclock=1800mhz,nodes=2 script.sh
```

This job requests 2 nodes and specifies their CPU frequencies should be set to 1800 MHz.

• a Linux power governor policy name. The governor names are:

  ○ *performance*: This governor instructs Linux to operate each logical processor at its maximum clock frequency.

    This setting consumes the most power and workload executes at the fastest possible speed.

  ○ *powersave*: This governor instructs Linux to operate each logical processor at its minimum clock frequency.

    This setting executes workload at the slowest possible speed. This setting does not necessarily consume the least amount of power since applications execute slower, and may actually consume more energy because of the additional time needed to complete the workload's execution.

  ○ *ondemand*: This governor dynamically switches the logical processor's clock frequency to the maximum value when system load is high and to the minimum value when the system load is low.

    This setting causes workload to execute at the fastest possible speed or the slowest possible speed, depending on OS load. The system switches between consuming the most power and the least power.

Resource	Format	Description

> ℹ️ The power saving benefits of *ondemand* might be non-existent due to frequency switching latency if the system load causes clock frequency changes too often.
>
> This has been true for older processors since changing the clock frequency required putting the processor into the C3 "sleep" state, changing its clock frequency, and then waking it up, all of which required a significant amount of time.
>
> Newer processors, such as the Intel Xeon E5-2600 Sandy Bridge processors, can change clock frequency dynamically and much faster.

- *conservative*: This governor operates like the *ondemand* governor but is more conservative in switching between frequencies. It switches more gradually and uses all possible clock frequencies.

  This governor can switch to an intermediate clock frequency if it seems appropriate to the system load and usage, which the *ondemand* governor does not do.

```
qsub -l cpuclock=performance,nodes=2 script.sh
```

This job requests 2 nodes and specifies their CPU frequencies should be set to the performance power governor policy.

- an ACPI performance state (or P-state) with or without the P prefix. P-states are a special range of values (0-15) that map to specific frequencies. Not all processors support all 16 states, however, they all start at P0. P0 sets the CPU clock frequency to the highest performance state which runs at the maximum frequency. P15 sets the CPU clock frequency to the lowest performance state which runs at the lowest frequency.

```
qsub -l cpuclock=3,nodes=2 script.sh
qsub -l cpuclock=p3,nodes=2 script.sh
```

This job requests 2 nodes and specifies their CPU frequencies should be set to a performance state of 3.

When reviewing job or node properties when **cpuclock** was used, be mindful of unit conversion. The OS reports frequency in Hz, not MHz or GHz.

Resource	Format	Description
epilogue	string	Specifies a user owned epilogue script which will be run before the system epilogue and epilogue.user scripts at the completion of a job. The syntax is `epilogue=<file>`. The file can be designated with an absolute or relative path. For more information, see Prologue and Epilogue Scripts.
feature	string	Specifies a property or feature for the job. Feature corresponds to Torque node properties and Moab features. ```qsub script.sh -l procs=10,feature=bigmem```
file	size	Sets RLIMIT_FSIZE for each process launched through the TM interface. See FILEREQUESTISJOBCENTRIC for information on how Moab schedules.
host	string	Name of the host on which the job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent.
mem	size	Maximum amount of physical memory used by the job. Ignored on Darwin, Digital Unix, Free BSD, HPUX 11, IRIX, NetBSD, and SunOS. Not implemented on AIX and HPUX 10. The `mem` resource will only work for single-node jobs. If your job requires multiple nodes, use `pmem` instead.
ncpus	integer	The number of processors in one task where a task cannot span nodes. ⓘ You cannot request both ncpus and nodes in the same job.
nice	integer	Number between -20 (highest priority) and 19 (lowest priority). Adjust the process execution priority.

Resource	Format	Description
nodes	{<node_count> | <hostname>} [:ppn=<ppn>] [:gpus=<gpu>] [:<property> [:<property>]...] [+ ...]	Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more node_specs joined with the + (plus) character: node_spec [+node_spec...]. Each node_spec is a number of nodes required of the type declared in the node_spec and a name of one or more properties desired for the nodes. The number, the name, and each property in the node_spec are separated by a : (colon). If no number is specified, one (1) is assumed. The name of a node is its hostname. The properties of nodes are:

- **ppn=#** - Specify the number of virtual processors per node requested for this job.

  The number of virtual processors available on a node by default is 1, but it can be configured in the TORQUE_HOME/server_priv/nodes file using the **np** attribute (see Server Node File Configuration). 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). 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.

> ℹ Torque does not have a TPN (tasks per node) property. You can specify TPN in Moab Workload Manager with Torque as your resource manager, but Torque does not recognize the property when it is submitted directly to it via qsub.

See qsub -l nodes for examples.

> ℹ 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 Moab Parameters in the *Moab Workload Manager Administrator Guide* for more information.

> ℹ All nodes in Torque have their own name as a property. You may request a specific node by using it's name in the nodes request. Multiple nodes can be requested this way by using '+' as a delimiter. For example:
>
> ```
> qsub -l nodes=node01:ppn=3+node02:ppn=6
> ```

Resource	Format	Description
		See the HOSTLIST RM extension in the *Moab Workload Manager Administrator Guide* for more information.
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 -1 procs=2`
procs_bit-map	string	A string made up of 1's and 0's in reverse order of the processor cores requested. A **procs_bitmap=1110** means the job requests a node that has four available cores, but the job runs exclusively on cores two, three, and four. With this bitmap, core one is not used. For more information, see Scheduling Cores.
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.
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 License Management 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

Interpreting Resource Requests

The table below shows how various requests are interpreted in the qsub -l syntax and corresponding cgroup usage.

Memory parameters (mem, pmem, vmem, pvmem) may specify units (examples: mem=1024mb, mem=320kb, mem=1gb). Recognized units are kb (kilobytes), mb (megabytes), gb (gigabytes), tb (terabyte), pb (petabytes), and eb (exabyte). If units are not specified, mb (megabytes) is assumed.

Example 3-1: Interpreting qsub -l requests

Usage	Description
`node=X:ppn=Y`	Creates *X* tasks that will use *Y* lprocs per task.
`procs=X`	Creates *X* tasks that will use 1 lproc each.
`ncpus=X`	Creates 1 task that will use *X* lprocs.
`mem=X`	The entire job will use *X* memory, divided evenly among the tasks.*
`pmem=X`	Each task will use *X* memory. In translation, -l nodes=1:ppn=4,pmem=1gb will use 4 GB of memory.*
`vmem=X`	The entire job will use *X* swap, divided evenly among the tasks. If legacy_vmem is set to true in the server, then the entire specified value will be given per host.**
`pvmem=X`	Each task will use *X* swap. In translation, -l nodes=1:ppn=4,pvmem=1gb will use 4 GB of swap.**

*If both mem and pmem are specified, the less restrictive of the two will be used as the limit for the job. For example, `qsub job.sh -l nodes=2:ppn=2,mem=4gb,pmem=1gb` will apply the mem requested instead of pmem, because it will allow 2 GB per task (4 GB/2 tasks) instead of 1 GB per task.

**If both vmem and pvmem are specified, the less restrictive of the two will be used as the limit for the job. For example, `qsub job.sh -l nodes=2:ppn=2,vmem=4gb,pvmem=1gb` will apply pvmem instead of vmem, because it will allow 2 GB swap per task (1 GB * 2 ppn) instead of .5 GB per task (1 GB/2 tasks).

Interpreting Node Requests

The table below shows how various qsub -l nodes requests are interpreted.

Example 3-2: qsub -l nodes

Usage	Description
`> qsub -l nodes=12`	Request 12 nodes of any type

Usage	Description
`> qsub -l nodes=2:server+14`	Request 2 "server" nodes and 14 other nodes (a total of 16) - this specifies two node_specs, "2:server" and "14"
`> qsub -l nodes-s=server:hippi+10:noserver+3:bigmem:hippi`	Request (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not 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 3-3:

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

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

Example 3-4:

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

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

Moab Job Extensions

qsub -l supports these Moab scheduler job extensions:

advres	hostlist	pref	spriority
cpuclock	image	procs	subnode
deadline	jgroup	procs_bitmap	subnode_list
depend	jobflags	prologue	taskdistpolicy
ddisk	latency	qos	template, termsig
dmem	loglevel	queuejob	termtime
energy_used	minprocspeed	reqattr	tid
epilogue	minpreempttime	retrycount	tpn
feature	minwclimit	retrycc	trig
flags	naccesspolicy	rmtype	trl
gattr	nallocpolicy	select	var
geometry	nodeset	sid	vcores
gmetric	opsys	signal	wcrequeue
gres	os	stagein	
	partition		

Related Topics

Job Submission

Requesting NUMA-Aware Resources

ℹ This topic only applies for NUMA-aware systems and requires Torque Resource Manager 6.0 and later.

Various NUMA resources can be requested at the time of job submission.

The qsub -L option allows administrators the ability to place jobs at the "task" or "OS process" level to get maximum efficiency out of the available hardware. In addition, multiple, non-symmetric resource requests can be made for the same job using the -L job submission syntax. See for a complete list of -L values.

For example:

```
qsub -L tasks=4:lprocs=2:usecores:memory=500mb -L tasks=8:lprocs=4:memory=2gb
```

Creates two requests. The first will create 4 tasks with two logical processors per task and 500 mb of memory per task. The logical processors will be placed on cores. The second request calls for 8 tasks with 4 logical processors per task and 2 gb of memory per task. The logical processors may be placed on cores or threads since the default placement is allowthreads.

> ⓘ The queue attribute "resources_default" has several options which are not
> compatible with the qsub -L syntax. If a queue has any of the following
> "resources_default" options set, the job will be rejected from the queue.
> "resources_default" options: nodes, size, mppwidth, mem, hostlist, ncpus,
> procs, pvmem, pmem, vmem, reqattr, software, geometry, opsys, tpn,
> and trl.

Requesting Generic Resources

When **generic** resources have been assigned to nodes using the server's nodes
file, these resources can be requested at the time of job submission using the
other field. See **Managing Consumable Generic Resources** on page 1 in the
Moab Workload Manager Administrator Guide for details on configuration
within Moab.

Example 3-5: 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
 Job Submission

Requesting Floating Resources

When **floating** resources have been set up inside Moab, they can be requested
in the same way as **generic** resources. Moab will automatically understand that
these resources are floating and will schedule the job accordingly. See
Managing Shared Cluster Resources (Floating Resources) on page 1 in
the *Moab Workload Manager Administrator Guide* for details on configuration
within Moab.

Example 3-6: 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
Job Submission

Requesting Other Resources

Many other resources can be requested at the time of job submission using the
Moab Workload Manager. See **Resource Manager Extensions** on page 1 in
the *Moab Workload Manager Administrator Guide* for a list of these supported
requests and correct syntax.

Related Topics

Requesting Resources
Job Submission

Exported Batch Environment Variables

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

Variable	Description
PBS_JOBNAME	User specified jobname
PBS_ARRAYID	Zero-based value of job array index for this job (in version 2.2.0 and later)
PBS_GPUFILE	Line-delimited list of GPUs allocated to the job located in `TORQUE_HOME/aux/jobidgpu`. Each line follows the following format: *`<host>`*`-gpu`*`<number>`* For example, `myhost-gpu1`.
PBS_O_ WORKDIR	Job's submission directory
PBS_ ENVIRONMENT	N/A

Variable	Description
PBS_TASKNUM	Number of tasks requested
PBS_O_HOME	Home directory of submitting user
PBS_MOMPORT	Active port for MOM daemon
PBS_O_LOGNAME	Name of submitting user
PBS_O_LANG	Language variable for job
PBS_JOBCOOKIE	Job cookie
PBS_JOBID	Unique pbs job id
PBS_NODENUM	Node offset number
PBS_NUM_NODES	Number of nodes allocated to the job
PBS_NUM_PPN	Number of procs per node allocated to the job
PBS_O_SHELL	Script shell
PBS_O_HOST	Host on which job script is currently running
PBS_QUEUE	Job queue
PBS_NODEFILE	File containing line delimited list of nodes allocated to the job
PBS_NP	Number of execution slots (cores) for the job
PBS_O_PATH	Path variable used to locate executables within job script

Related Topics

Requesting Resources
Job Submission

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).

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

- Set the **submit_hosts** server parameter (see Using the "submit_hosts" service parameter).

  Allows specified hosts to be trusted as a submit host.

- Use **.rhosts** to enable ruserok() based authentication (see Using RCmd authentication).

See Configuring Job Submission Hosts for more information.

> 🛈 When you enable allow_node_submit, you must also enable the allow_proxy_user parameter to allow user proxying when submitting and running jobs.

Related Topics

Job Submission

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 Files

Torque 4.5.0 was updated to accept XML-based job files in addition to the binary job files. The change allows job files to be more human-readable and easier to parse. Below is a sample job file in the new XML format:

```
<?xml version="1.0"?>
<job>
  <version>131842</version>
  <state>1</state>
  <substate>10</substate>
  <server_flags>33</server_flags>
  <start_time>0</start_time>
  <jobid>340</jobid>
  <fileprefix>340</fileprefix>
  <queue>batch</queue>
  <destination_queue></destination_queue>
  <record_type>1</record_type>
  <mom_address>0</mom_address>
  <mom_port>11</mom_port>
  <mom_rmport>0</mom_rmport>
  <attributes>
    <Job_Name flags="1">job2.sh</Job_Name>
    <Job_Owner flags="1">echan@moabServer.cn</Job_Owner>
    <job_state flags="1">Q</job_state>
    <queue flags="3">batch</queue>
    <server flags="1">company.com</server>
    <Checkpoint flags="1">u</Checkpoint>
    <ctime flags="1">1384292754</ctime>
    <Error_Path flags="1">moabServer.cn:/home/echan/work/job2.sh.e340</Error_Path>
    <Hold_Types flags="1">n</Hold_Types>
    <Join_Path flags="1">n</Join_Path>
    <Keep_Files flags="1">n</Keep_Files>
    <Mail_Points flags="1">a</Mail_Points>
    <mtime flags="1">1384292754</mtime>
    <Output_Path flags="1">moabServer.cn:/home/echan/work/job2.sh.o340</Output_Path>
    <Priority flags="1">0</Priority>
    <qtime flags="1">1384292754</qtime>
    <Rerunable flags="1">True</Rerunable>
    <Resource_List>
      <epilogue flags="1">/tmp/epilogue.sh</epilogue>
      <neednodes flags="1">moabServer:ppn=1</neednodes>
      <nodect flags="1">1</nodect>
      <nodes flags="1">moabServer:ppn=1</nodes>
    </Resource_List>
    <substate flags="1">10</substate>
    <Variable_List flags="1">PBS_O_QUEUE=batch
PBS_O_HOME=/home/echan
PBS_O_LOGNAME=echan
PBS_O_
PATH=/home/echan/eclipse:/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/
sbin:/usr/bin:/sbin:/bin:/usr/games:/opt/moab/bin:/opt/moab/sbin
PBS_O_SHELL=/bin/bash
PBS_O_LANG=en_US
PBS_O_WORKDIR=/home/echan/work
PBS_O_HOST=moabServer.cn
PBS_O_SERVER=moabServer
</Variable_List>
    <euser flags="1">echan</euser>
    <egroup flags="5">company</egroup>
    <hop_count flags="1">1</hop_count>
    <queue_rank flags="1">2</queue_rank>
    <queue_type flags="1">E</queue_type>
    <etime flags="1">1384292754</etime>
    <submit_args flags="1">-l nodes=lei:ppn=1 -l epilogue=/tmp/epilogue.sh
./job2.sh</submit_args>
    <fault_tolerant flags="1">False</fault_tolerant>
    <job_radix flags="1">0</job_radix>
```

```
      <submit_host flags="1">lei.ac</submit_host>
   </attributes>
</job>
```

The above job was submitted with this submit command:

```
qsub -l nodes=moabServer:ppn=1 -l epilogue=/tmp/epilogue.sh ./job2.sh
```

Related Topics

Job Submission

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
...
```

Monitoring NUMA Job Task Placement

> ℹ NUMA-aware job task placement is available with Torque Resource Manager 6.0 and later.

When using NUMA, job resources are tracked per task. To support this `qstat -f` produces a new category of information that begins with the " req_ information" keyword. Following each "req_information keyword" is another keyword giving information about how the job was allocated. See -L NUMA Resource Request on page 185 for available allocation keywords.

When the job has completed, the output will also include the per task resident memory used and per task cpu time used. The following is a sample qstat -f completed job output.

You will see that req_information.task_usage.0.task.0.cpu_list gives the cores to which the job is bound for the cpuset. The same for mem_list. The keywords memory_used and cput_used report the per task resident memory used and cpu time used respectively.

```
Job Id: 832.pv-knielson-dt
Job_Name = bigmem.sh
Job_Owner = knielson@pv-knielson-dt
resources_used.cput = 00:00:00
resources_used.energy_used = 0
resources_used.mem = 3628kb
resources_used.vmem = 31688kb
resources_used.walltime = 00:00:00
job_state = C
queue = second
server = pv-knielson-dt
Checkpoint = u
ctime = Tue Jul 28 23:23:15 2015
Error_Path = pv-knielson-dt:/home/knielson/jobs/bigmem.sh.e832
exec_host = pv-knielson-dt/0-3
Hold_Types = n
Join_Path = n
Keep_Files = n
Mail_Points = a
mtime = Tue Jul 28 23:23:18 2015
Output_Path = pv-knielson-dt:/home/knielson/jobs/bigmem.sh.o832
Priority = 0
qtime = Tue Jul 28 23:23:15 2015
Rerunable = True
Resource_List.walltime = 00:05:00
session_id = 2708
substate = 59
Variable_List = PBS_O_QUEUE=routeme,PBS_O_HOME=/home/knielson,
PBS_O_LOGNAME=knielson,
PBS_O_PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
in:/usr/games:/usr/local/games,PBS_O_SHELL=/bin/bash,PBS_O_LANG=en_US,
PBS_O_WORKDIR=/home/knielson/jobs,PBS_O_HOST=pv-knielson-dt,
PBS_O_SERVER=pv-knielson-dt
euser = knielson
egroup = company
hashname = 832.pv-knielson-dt
queue_rank = 391
queue_type = E
etime = Tue Jul 28 23:23:15 2015
exit_status = 0
submit_args = -L tasks=2:lprocs=2 ../scripts/bigmem.sh
start_time = Tue Jul 28 23:23:18 2015
start_count = 1
fault_tolerant = False
comp_time = Tue Jul 28 23:23:18 2015
job_radix = 0
total_runtime = 0.093262
submit_host = pv-knielson-dt
req_information.task_count.0 = 2
req_information.lprocs.0 = 2
req_information.thread_usage_policy.0 = allowthreads
req_information.hostlist.0 = pv-knielson-dt:ppn=4
req_information.task_usage.0.task.0.cpu_list = 2,6
req_information.task_usage.0.task.0.mem_list = 0
req_information.task_usage.0.task.0.memory_used = 258048
req_information.task_usage.0.task.0.cput_used = 18
req_information.task_usage.0.task.0.cores = 0
req_information.task_usage.0.task.0.threads = 0
req_information.task_usage.0.task.0.host =
req_information.task_usage.0.task.1.cpu_list = 3,7
req_information.task_usage.0.task.1.mem_list = 0
req_information.task_usage.0.task.1.memory_used = 258048
req_information.task_usage.0.task.1.cput_used = 18
req_information.task_usage.0.task.1.cores = 0
req_information.task_usage.0.task.1.threads = 2
req_information.task_usage.0.task.1.host = pv-knielson-dt
```

Related Topics

[Submitting and Managing Jobs](#)

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](#)

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](#)

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 or the [keep_completed](#) parameter on the server. This should be set to the number of seconds that jobs should be held in the queue. If you set keep_completed on the job execution queue, completed jobs will be reported in the **C** state and the exit status is seen in the `exit_status` job attribute.

> ⓘ If the Mother Superior and Torque server are on the same server, expect the following behavior:
>
> - When keep_completed is set, the job spool files will be deleted when the specified time arrives and Torque purges the job from memory.
>
> - When keep_completed is not set, Torque deletes the job spool files upon job completion.
>
> - If you manually purge a job (qdel -p) before the job completes or time runs out, Torque will never delete the spool files.

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

Related Topics

[Submitting and Managing Jobs](#)

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
- Configuration Files and Scripts
- Starting a Checkpointable Job
- Checkpointing a Job
- Restarting a Job
- Acceptance Tests

Related Topics

Submitting and Managing Jobs

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

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 3-7: 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 3-8: 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
```

Job Checkpoint and Restart

Example 3-9: `mom_priv/blcr_checkpoint_script`

```perl
#! /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 3-10: `mom_priv/blcr_restart_script`

```perl
#! /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](#)

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 3-11: 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 3-12: Instructions for building test program

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

Example 3-13: Sample test script

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

Example 3-14: 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

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

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

Acceptance Tests

A number of tests were made to verify the functioning of the BLCR implementation. See BLCR Acceptance Tests for a description of the testing.

Related Topics

Job Checkpoint and Restart

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 3-15: qstat -f (job failure)

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

> ⓘ The value of `Resource_List.*` is the amount of resources requested.

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

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

Example 3-16: 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

Name	Value	Description
JOB_EXEC_INITRST	-5	Job aborted on MOM init, chkpt, no migrate
JOB_EXEC_INITRMG	-6	Job aborted on MOM init, chkpt, ok migrate
JOB_EXEC_BADRESRT	-7	Job restart failed
JOB_EXEC_CMDFAIL	-8	Exec() of user command failed
JOB_EXEC_STDOUTFAIL	-9	Could not create/open stdout stderr files
JOB_EXEC_OVERLIMIT_MEM	-10	Job exceeded a memory limit
JOB_EXEC_OVERLIMIT_WT	-11	Job exceeded a walltime limit
JOB_EXEC_OVERLIMIT_CPUT	-12	Job exceeded a CPU time limit

Example 3-17: 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
Submitting and Managing Jobs

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
- Submitting Service Jobs in MCM
- Managing Service Jobs

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

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

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

Chapter 4 Managing Nodes

This chapter 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
- Node Properties
- Changing Node State
- Host Security
- Linux Cpuset Support
- Scheduling Cores
- Scheduling Accelerator Hardware
- Node Resource Plug-in

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).

> **ⓘ** Be aware of the following:
>
> - Nodes cannot be added or deleted dynamically if there is a mom_hierarchy file in the server_priv directory.
> - When you make changes to nodes by directly editing the nodes file, you must restart pbs_server for those changes to take effect. Changes made using qmgr do not require a restart.
> - When you make changes to a node's ip address, you must clear the pbs_server cache. Either restart pbs_server or delete the changed node and then re-add it.
> - Before a newly added node is set to a free state, the cluster must be informed that the new node is valid and they can trust it for running jobs. Once this is done, the node will automatically transition to free.
> - Adding or changing a hostname on a node requires a pbs_server restart in order to add the new hostname as a node.

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[,node004,node005...] [np=n,][TTL=yyyy-mm-ddThh:mm:ssZ,]
[acl="user==user1:user2:user3",][requestid=n]'
```

The optional parameters are used as follows:

- np – Number of virtual processors.

- TTL – (Time to Live) Specifies the time in UTC format that the node is supposed to be retired by Moab. Moab will not schedule any jobs on a node after its time to live has passed.

- acl – (Access control list) Can be used to control which users have access to the node in Moab.

- requestid – An ID that can be used to track the request that created the node.

You can alter the parameters of a node using a set command as follows:

```
qmgr -c 'set node node003 np=y'
qmgr -c 'set node node003 TTL=yyyy-mm-ddThh:mm:ssZ'
qmgr -c 'set node node003 requestid=23234'
qmgr -c 'set node node003 acl="user10,user11,user12"'
qmgr -c 'set node node003 acl+="user5,user6"'
qmgr -c 'set node node003 acl-=user1'
```

> ⓘ Torque does not use the TTL, acl, and requestid parameters. Information for those parameters are simply passed to Moab.

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

```
node003 np=3 TTL=2014-08-06T14:30:00Z acl=user1,user2,user3 requestid=3210
node004 ...
```

Nodes can also be removed with a similar command:

```
> qmgr -c 'delete node node003[,node004,node005...]'
```

Related Topics

Changing Node State
Managing Nodes

Node Properties

Torque can associate properties with nodes to aid in identifying groups of nodes. It's typical for a site to conglomerate a heterogeneous set of resources.

To identify the different sets, properties can be given to each node in a set. For example, a group of nodes that has a higher speed network connection could have the property "ib". Torque can set, update, or remove properties either dynamically with **qmgr** or by manually editing the `nodes` file.

Run-time Node Changes

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

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

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

Manual Node Changes

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

`server_priv/nodes`:

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

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

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

Related Topics

    Job Submission

    Managing Nodes

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.)

Node Recovery

When a mom gets behind on processing requests, pbs_server has a failsafe to allow for node recovery in processing the request backlog. After three failures without having two consecutive successes in servicing a request, pbs_server will mark that mom as offline for five minutes to allow the mom extra time to process the backlog before it resumes its normal activity. If the mom has two consecutive successes in responding to network requests before the timeout, then it will come back earlier.

Related Topics

Managing Nodes

Changing Node Power States

Beginning with Torque 5.0.0, the pbsnodes -m command can modify the power state of nodes. Node cannot go from one low-power state to another low-power state. They must be brought up to the Running state and then moved to the new low-power state. The supported power states are:

State	Description
Running	- Physical machine is actively working - Power conservation is on a per-device basis - Processor power consumption controlled by P-states
Standby	- System appears off - Processor halted (OS executes a "halt" instruction) - Processor maintains CPU and system cache state - RAM refreshed to maintain memory state - Machine in low-power mode - Requires interrupt to exit state - Lowest-latency sleep state - has no effect on software

State	Description
Suspend	- System appears off - Processor and support chipset have no power - OS maintains CPU, system cache, and support chipset state in memory - RAM in slow refresh - Machine in lowest-power state - Usually requires specific interrupt (keyboard, mouse) to exit state - Third lowest-latency sleep state - system must restore power to processor and support chipset
Hibernate	- System is off - Physical machine state and memory saved to disk - Requires restoration of power and machine state to exit state - Second highest-latency sleep state - system performs faster boot using saved machine state and copy of memory
Shutdown	Equivalent to shutdown now command as root

In order to wake nodes and bring them up to a running state:

- the nodes must support, and be configured to use, Wake-on-LAN (WOL).
- the pbsnodes command must report the node's MAC address correctly.

To configure nodes to use Wake-on-LAN

1. Enable WOL in the BIOS for each node. If needed, contact your hardware manufacturer for details.
2. Use the ethtool command to determine what types of WOL packets your hardware supports. Torque uses the g packet. If the g packet is not listed, you cannot use WOL with Torque.

```
[root]# ethtool eth0
Settings for eth0:
        Supported ports: [ TP ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Speed: 100Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 2
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: off
        Supports Wake-on: pumbg
        Wake-on: p
         Current message level: 0x00000007 (7)
                                drv probe link
        Link detected: yes
```

This Ethernet interface supports the g *WOL packet, but is currently set to use the* p *packet.*

3. If your Ethernet interface supports the g packet, but is configured for a different packet, use ethtool -s *<interface>* wol g to configure it to use g.

```
[root]# ethtool -s eth0 wol g
[root]# ethtool eth0
Settings for eth0:
        Supported ports: [ TP ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Speed: 100Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 2
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: off
        Supports Wake-on: pumbg
        Wake-on: g
        Current message level: 0x00000007 (7)
                                drv probe link
        Link detected: yes
```

Now the power state of your nodes can be modified and they can be woken up from power-saving states.

Related Topics

pbsnodes

Host Security

Enabling PAM with Torque

Torque is able to take advantage of the authentication services provided through Pluggable Authentication Modules (PAM) to help administrators manage access to compute nodes by users. The PAM module available in Torque is located in the PAM security directory. This module, when used in conjunction with other PAM modules, restricts access to the compute node unless the user has a job currently running on the node. The following configurations are examples only. For more information about PAM, see the PAM (Pluggable Authentication Modules) documentation from LinuxDocs.

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

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

In this example, after installing Torque with PAM enabled, you would add the following two lines to `/etc/pam.d/sshd`:

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

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

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

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

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

Legacy Torque PAM Configuration

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

For systems requiring dedicated access to compute nodes (for example, users with sensitive data), Torque prologue and epilogue scripts provide a vehicle to leverage the authentication provided by linux-PAM modules. (See Prologue and Epilogue Scripts 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

Linux Cpuset Support

- Cpuset Overview
- Cpuset Support
- Configuring Cpuset
- Cpuset Advantages/Disadvantages

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 and corresponding hwloc-devel package installed. See Installing Torque Resource Manager for more information.

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.

Configuring Cpuset

To configure cpuset

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

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

> ⓘ Do this for each MOM that is to use cpusets.

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

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

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

Cpuset Advantages/Disadvantages

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

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

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

Related Topics

Managing Nodes
Geometry Request Configuration

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.

See these topics for details:

- Geometry Request Configuration
- Geometry Request Usage
- Geometry Request Considerations

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. If the operating environment is suitable for geometry requests, configure Torque with the `--enable-geometry-requests` option.

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

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

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

Related Topics

[Scheduling Cores](#)

Geometry Request Usage

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

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

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

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

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

Related Topics

[Scheduling Cores](#)

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](#)

Scheduling Accelerator Hardware

Torque works with accelerators (such as NVIDIA GPUs and Intel MICs) and can collect and report metrics from them or submit workload to them. This feature requires the use of the Moab scheduler. See **Accelerators** on page 1 in the *Moab Workload Manager Administrator Guide* for information on configuring accelerators in Torque.

Node Resource Plug-in

There is now an API for creating a resource plug-in to allow the reporting of custom varattrs, generic resources, generic metrics, and node features. Additionally, jobs can be made to report custom resources through the same plug-in. The purpose of this plug-in is to allow some resource integration to happen outside of the normal code release cycle and without having to be part of the main codebase for Torque This should allow specific sites to implement things that are not of general interest, as well as provide a tight integration option for resources that vary widely based on hardware.

Torque's resource plug-in capability provides an API through which a Torque plug-in can add arbitrary generic resources, generic metrics, varattrs, and features to a node. Additionally, Torque plug-in can add arbitrary resource usage per job.

The API can be found in `trq_plugin_api.h`. To implement a plug-in, you must implement all of the API functions, even if the function does nothing. An implementation that does nothing may be found in `contrib/resource_plugin.cpp`. If you wish, you may simply add the desired functionality to this file, build the library, and link it to the MOM at build time.

Plug-in Implementation Recommendations

Your plug-in must execute very quickly in order to avoid causing problems for the pbs_mom daemon. The node resource portion of the plug-in has a 5 second time limit, and the job resource usage portion has a 3 second time limit. The node resource portion executes each time the MOM sends a status to pbs_ server, and the job resource usage portion executes once per job at the same time interval. The node resource and job resource portions block pbs_mom while they are executing, so they should execute in a short, deterministic amount of time.

Remember, you are responsible for plug-ins, so please design well and test thoroughly.

Building the Plug-in

If you do not change the name of the .cpp file and wish to build it, execute the following:

```
export TRQ_HEADER_LOCATION=/usr/local/include/
g++ -fPIC -I $TRQ_HEADER_LOCATION resource_plugin.cpp -shared -o libresource_plugin.so
```

NOTE: Change TRQ_HEADER_LOCATION if you configured torque with the --prefix option.

Testing the Plug-in

NOTE: You assume all responsibility for any plug-ins. This document is intended to assist you in testing the plug-ins, but this list of suggested tests may not be comprehensive. We do not assume responsibility if these suggested tests do not cover everything.

Testing Basic Functionality

Once you've implemented and built your library, you can begin testing. For your convenience, a simple test driver can be found in `plugin_driver.cpp`. You can build this executable and link it against your library as shown below in order to manually verify the output:

```
export PLUGIN_LIB_PATH=/usr/local/lib
g++ plugin_driver.cpp -I $TRQ_HEADER_LOCATION -L $PLUGIN_LIB_PATH -lresource_plugin -o driver
```

You can then execute the driver and manually inspect the output:

```
./driver
```

NOTE: Change PLUGIN_LIB_PATH if you have installed the plug-in somewhere other than `/usr/local/lib`.

To illustrate output, a simple plug-in that reports:

- 2 broams of stormlight used, ignoring the random process ID found by the driver
- 1024 hbmem for GRES
- temperature of 75.2 for GMETRICS
- octave = 3.2.4 for VARATTRS
- haswell for features

will have the output:

```
$ ./driver
Your plugin reported the following for the random pid 7976:
stormlight = 2broams
Your plugin reports the following for this host:
        GRES:
                hbmem = 1024

        GMETRICS:
                temperature = 75.20

        VARATTRS:
                octave = 3.2.4

        FEATURES: haswell
```

Testing for Memory Leaks

In order to prevent your compute nodes from being compromised for speed or even going down due to out-of-memory conditions, you should run your plug-in under valigrind to test that it is correctly managing memory.

Assuming you are executing the driver from the "Testing Basic Functionality" section, you can run:

```
valgrind --tool=memcheck --leak-check=full --log-file=plugin_valgrind_output.txt
./driver
```

If you are not familiar with valgrind, a good primer can be found at The Valgrind Quick Start Guide.

We recommend that you fix all errors reported by valgrind.

Enabling the Plug-in

Once you've implemented, built, and thoroughly tested your plug-in (remember that our suggestions may not address everything), you will want to enable it in Torque. Your plug-in can be linked in at build time:

```
./configure <your other options> --with-resource-plugin=<path to your resource
plugin>
```

NOTE: You will want to make sure that the path you specify is in $LD_LIBRARY_ PATH, or can otherwise be found by pbs_mom when you start the daemon.

Once you build, you can then start the new MOM and be able to observe the plug-in's output using pbsnodes, qstat -f, and in the accounting file.

Sample pbsnodes output:

```
<normal output>
gres:hbmem = 20
gmetric:temperature = 76.20
varattr:octave = 3.2.4
features = haswell
```

The keywords at the front let Moab know what each line means, so it can use them accordingly.

Sample accounting file entry:

```
<normal entry until resources used> resources_used.cput=0
resources_used.energy_used=0 resources_used.mem=452kb
resources_used.vmem=22372kb resources_used.walltime=00:05:00
resources_used.stormlight=2broams
```

Your plug-in resources reported will appear in the form:

```
resources_used.<name you supplied>=<value you supplied>
```

The above example includes the arbitrary resource stormlight and the value of 2broams.

Sample qstat -f output:

```
<normal qstat -f output>
resources_used.stormlight = 2broams
```

The resources used reported bythe plug-in will appear at the end of the qstat -f output in the same format as in the accounting file.

Chapter 5 Setting Server Policies

This section explains how to set up and configure your queue. This section also explains how to set up Torque to run in high availability mode. For details, see these topics:

- Queue Configuration
- Server High Availability

Queue Configuration

To initially define a queue, use the create subcommand of **qmgr**:

```
> 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**.

```
> qmgr -c "set queue batch started=true"
> qmgr -c "set queue batch enabled=true"
```

Further configuration is possible using any combination of the following attributes.

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:

Related Topics

qalter - command which can move jobs from one queue to another

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

Setting Queue Resource Controls with Resource Request Syntax 2.0

Using the -L syntax, you can set default, minimum, and maximum values for lproces, memory, swap, disk, sockets, numanode, cores and threads with resource request 2.0.

These can be set in the general format:

```
qmgr -c "set queue <queue_name> req_information_[default|min|max].
[lprocs|memory|swap|disk|sockets|numanode|core|thread]
```

Example 5-1: Jobs using -L syntax

```
qmgr -c "set queue q1 req_information_default.lprocs=2"
qmgr -c "set queue q1 req_information_minimum.memory=2gb"
qmgr -c "set queue q1 req_information_maximum.core=10
```

> ⚠️ When req_information_default.* settings are set, only jobs with the -L syntax are accepted. Jobs submitted using the -l (resources_default.*) setting will be rejected with a message citing conflicting resource requests. If you want a queue to be able to accept *both* kinds of jobs and still be able to enforce defaults, then simply set defaults for both resource request types. See Example 5-2 for more information.

Example 5-2: Jobs using -L or -l syntax

This example shows how to enable a queue to be able to accept *both* kinds of jobs and still be able to enforce defaults.

```
qmgr -c "create queue batch"
qmgr -c "set queue batch queue_type = Execution"
qmgr -c "set queue batch resources_default.mem = 3gb"
qmgr -c "set queue batch enabled = True"
qmgr -c "set queue batch started = True"
qmgr -c "set queue batch req_information_default.memory = 3gb
```

In this example, jobs submitted that explicitly use the -L syntax will have the req_information_default.memory setting applied. If the job does *not* explicitly use this syntax, then the resources_default.mem setting will be applied.

Related Topics

Queue Configuration

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

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
```

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

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

Related Topics

Queue Configuration

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 = route
set server resources_default.ncpus = 1
set server resources_default.walltime = 24:00:00
   ...
```

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

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

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

```
qmgr

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

Torque waits to apply the server and queue defaults until the job is assigned to its final execution queue. Queue defaults override the server defaults. 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

Queue Attributes

Server High Availability

Torque can run 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 primary server fails.

> ℹ The high availability feature is available in the 2.3 and later versions of Torque.

> ℹ The "native" high availability implementation, as described here, is only suitable for Moab Basic Edition. Contact Adaptive Computing for information on high availability for Enterprise Edition.

For more details, see these sections:

- Redundant server host machines
- Enabling High Availability
- Server High Availability
- Enhanced High Availability with Moab
- How Commands Select the Correct Server Host
- Job Names
- Persistence of the pbs_server Process
- High Availability of the NFS Server
- Installing Torque in High Availability Mode
- Installing Torque in High Availability Mode on Headless Nodes
- Example Setup of High Availability

Redundant server host machines

High availability enables Moab HPC Suite to continue running even if pbs_server is brought down. This is done by running multiple copies of pbs_server which have their `torque/server_priv` directory mounted on a shared file system.

> ℹ Do not use symlinks when sharing the Torque home directory or server_priv directories. A workaround for this is to use `mount --rbind /path/to/share /var/spool/torque`. Also, it is highly recommended that you only share the server_priv and not the entire *TORQUE_HOMEDIR*.

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 in the pbs_mom configuration to specify the server host names. You must use the *TORQUE_HOMEDIR*/`server_name` file.

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

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

Enabling High Availability

To use high availability, you must start each instance of pbs_server with the `--ha` option.

Three server options help manage high availability. The server parameters are **lock_file**, **lock_file_update_time**, and **lock_file_check_time**.

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

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

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

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

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

> ℹ️ Do not use anything but a simple NFS fileshare that is not used by anything
> else (i.e., only Moab can use the fileshare).

> ℹ️ Do not use a general-purpose NAS, parallel file system, or company-wide
> shared infrastructure to set up Moab high availability using "native" high
> availability.

Enhanced High Availability with Moab

When Torque is run with an external scheduler such as Moab, and the pbs_
server is not running on the same host as Moab, pbs_server needs to know
where to find the scheduler. To do this, use the `-l` option as demonstrated in
the example below (the port is required and the default is 15004).

- Red Hat 6 or SUSE 11-based systems

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

- Red Hat 7 or SUSE 12-based systems

  Set the PBS_ARGS environment variable in the `/etc/sysconfig/pbs_
  server` file. Set `PBS_ARGS=-l <moabhost:port>` where `moabhost` is the
  name of the alternate server node and `port` is the port on which Moab on
  the alternate server node is listening (default 15004).

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

- Red Hat 6 or SUSE 11-based systems

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

- Red Hat 7 or SUSE 12-based systems

  Set the PBS_ARGS environment variable in the `/etc/sysconfig/pbs_
  server` file to `PBS_ARGS=-l <moabhost:port> -l
  <moabhost2:port>`.

If pbs_server and Moab run on the same host, use the `--ha` option as
demonstrated in the example below.

- Red Hat 6 or SUSE 11-based systems

```
> pbs_server --ha
```

- Red Hat 7 or SUSE 12-based systems

  Set the PBS_ARGS environment variable in the `/etc/sysconfig/pbs_
  server` file to `PBS_ARGS=--ha`.

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

How Commands Select the Correct Server Host

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

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

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

Job Names

Job names normally contain the name of the host machine where pbs_server is running. When job names are constructed, only the server name in *$PBS_ DEFAULT* or the first name from the server specification list, `TORQUE_ HOME/server_name`, is used in building the job name.

Persistence of the pbs_server Process

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

High Availability of the NFS Server

> ⓘ Before installing a specific NFS HA solution please contact Adaptive Computing Support for a detailed discussion on NFS HA type and implementation path.

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

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

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

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

Installing Torque in High Availability Mode

The following procedure demonstrates a Torque installation in high availability (HA) mode.

Requirements

- gcc (GCC) 4.1.2
- BASH shell
- Servers configured the following way:
  - 2 main servers with identical architecture:
    - `server1` — Primary server running Torque with a shared file system (this example uses NFS)
    - `server2` — Secondary server running with Torque with a shared file system (this example uses NFS)
  - `fileServer` — Shared file system (this example uses NFS)
  - Compute nodes

To install Torque in HA mode

1. Stop all firewalls or update your firewall to allow traffic from Torque services.

   - Red Hat 6 or SUSE 11-based systems

     ```
     > service iptables stop
     > chkconfig iptables off
     ```

   - Red Hat 7 or SUSE 12-based systems

     ```
     > systemctl stop firewalld
     > systemctl disable firewalld
     ```

   If you are unable to stop the firewall due to infrastructure restriction, open the following ports:

- `15001[tcp,udp]`
- `15002[tcp,udp]`
- `15003[tcp,udp]`

2. Disable SELinux

```
> vi /etc/sysconfig/selinux
SELINUX=disabled
```

3. Update your main `~/.bashrc` profile to ensure you are always referencing the applications to be installed on all servers.

```
Torque
export TORQUE_HOME=/var/spool/torque

# Library Path

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${TORQUE_HOME}/lib

# Update system paths
export PATH=${TORQUE_HOME}/bin:${TORQUE_HOME}/sbin:${PATH}
```

4. Verify `server1` and `server2` are resolvable via either DNS or looking for an entry in the `/etc/hosts` file.

5. Configure the NFS Mounts by following these steps:

   a. Create mount point folders on `fileServer`.

   ```
   fileServer# mkdir -m 0755 /var/spool/torque
   fileServer# mkdir -m 0750 /var/spool/torque/server_priv
   ```

   b. Update `/etc/exports` on `fileServer`. The IP addresses should be that of `server2`.

   ```
   /var/spool/torque/server_priv 192.168.0.0/255.255.255.0(rw,sync,no_root_squash)
   ```

   c. Update the list of NFS exported file systems.

   ```
   fileServer# exportfs -r
   ```

6. If the NFS daemons are not already running on `fileServer`, start them.

   - Red Hat 6 or SUSE 11-based systems

   ```
   > service rpcbind restart
   > service nfs-server start
   > service nfs-lock start
   > service nfs-idmap start
   ```

- Red Hat 7 or SUSE 12-based systems

```
> systemctl restart rpcbind.service
> systemctl start nfs-server.service
> systemctl start nfs-lock.service
> systemctl start nfs-idmap.service
```

7. Mount the exported file systems on `server1` by following these steps:

   a. Create the directory reference and mount them.

   ```
   server1# mkdir /var/spool/torque/server_priv
   ```

   Repeat this process for `server2`.

   b. Update `/etc/fstab` on `server1` to ensure that NFS mount is performed on startup.

   ```
   fileServer:/var/spool/torque/server_priv /var/spool/torque/server_priv nfs
   rsize= 8192,wsize=8192,timeo=14,intr
   ```

   Repeat this step for `server2`.

8. Install Torque by following these steps:

   a. Download and extract Torque6.1.0 on `server1`.

   ```
   server1# wget http://github.com/adaptivecomputing/torque/branches/6.1.0/torque-
   6.1.0.tar.gz
   server1# tar -xvzf torque-6.1.0.tar.gz
   ```

   b. Navigate to the Torque directory and compile Torque on server1.

   ```
   server1# configure
   server1# make
   server1# make install
   server1# make packages
   ```

   c. If the installation directory is shared on both head nodes, then run `make install` on `server1`.

   ```
   server1# make install
   ```

   If the installation directory is not shared, repeat step 8a-b (downloading and installing Torque) on `server2`.

9. Start trqauthd.

   - Red Hat 6 or SUSE 11-based systems

   ```
   server1# service trqauthd start
   ```

- Red Hat 7 or SUSE 12-based systems

```
server1# systemctl start trqauthd
```

10. Configure Torque for HA.

a. List the host names of all nodes that run pbs_server in the `torque/server_name` file. You must also include the host names of all nodes running pbs_server in the `torque/server_name` file of each MOM node. The syntax of `torque/server_name` is a comma-delimited list of host names.

```
server1,server2
```

b. Create a simple queue configuration for Torque job queues on server1.

```
server1# pbs_server -t create
server1# qmgr -c "set server scheduling=true"
server1# qmgr -c "create queue batch queue_type=execution"
server1# qmgr -c "set queue batch started=true"
server1# qmgr -c "set queue batch enabled=true"
server1# qmgr -c "set queue batch resources_default.nodes=1"
server1# qmgr -c "set queue batch resources_default.walltime=3600"
server1# qmgr -c "set server default_queue=batch"
```

> ⓘ Because `server_priv/*` is a shared drive, you do not need to repeat this step on `server2`.

c. Add the root users of Torque to the Torque configuration as an operator and manager.

```
server1# qmgr -c "set server managers += root@server1"
server1# qmgr -c "set server managers += root@server2"
server1# qmgr -c "set server operators += root@server1"
server1# qmgr -c "set server operators += root@server2"
```

> ⓘ Because `server_priv/*` is a shared drive, you do not need to repeat this step on Server 2.

d. You must update the lock file mechanism for Torque in order to determine which server is the primary. To do so, use the `lock_file_update_time` and `lock_file_check_time` parameters. The primary pbs_server will update the lock file based on the specified `lock_file_update_time` (default value of 3 seconds). All backup pbs_servers will check the lock file as indicated by the `lock_file_check_time` parameter (default value of 9 seconds). The `lock_file_update_time` must be less than the `lock_file_check_time`. When a failure occurs, the backup pbs_server takes up to the `lock_file_check_time` value to take over.

```
server1# qmgr -c "set server lock_file_check_time=5"
server1# qmgr -c "set server lock_file_update_time=3"
```

> ⓘ Because `server_priv/*` is a shared drive, you do not need to repeat this step on `server2`.

e. List the servers running pbs_server in the Torque`acl_hosts` file.

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

> ⓘ Because `server_priv/*` is a shared drive, you do not need to repeat this step on `server2`.

f. Restart the running pbs_server in HA mode.

- Red Hat 6 or SUSE 11-based systems

```
service pbs_server stop
```

- Red Hat 7 or SUSE 12-based systems

```
systemctl stop pbs_server
```

g. Start the pbs_server on the secondary server.

- Red Hat 6 or SUSE 11-based systems

```
service pbs_server start
```

- Red Hat 7 or SUSE 12-based systems

```
systemctl start pbs_server
```

11. Check the status of Torque in HA mode.

```
server1# qmgr -c "p s"
server2# qmgr -c "p s"
```
> The commands above returns all settings from the active Torque server from either node.

a. Drop one of the pbs_servers to verify that the secondary server picks up the request.

- Red Hat 6 or SUSE 11-based systems

```
service pbs_server stop
```

- Red Hat 7 or SUSE 12-based systems

```
systemctl stop pbs_server
```

b. Stop the pbs_server on `server2` and restart pbs_server on `server1` to verify that both nodes can handle a request from the other.

12. Install a pbs_mom on the compute nodes.

   a. Copy the install scripts to the compute nodes and install.

   b. Navigate to the shared source directory of Torque and run the following:

```
node1 torque-package-mom-linux-x86_64.sh --install
node2 torque-package-clients-linux-x86_64.sh --install
```

Repeat this for each compute node. Verify that the `/var/spool/torque/server-name` file shows all your compute nodes.

   c. On `server1` or `server2`, configure the nodes file to identify all available MOMs. To do so, edit the `/var/spool/torque/server_priv/nodes` file.

```
node1 np=2
node2 np=2
```

> ⓘ Change the `np` flag to reflect number of available processors on that node.

   d. Recycle the pbs_servers to verify that they pick up the MOM configuration.

   - Red Hat 6 or SUSE 11-based systems

```
service pbs_server stop
```

   - Red Hat 7 or SUSE 12-based systems

```
systemctl stop pbs_server
```

   e. Start the pbs_mom on each execution node.

   - Red Hat 6 or SUSE 11-based systems

```
service pbs_mom start
```

   - Red Hat 7 or SUSE 12-based systems

```
systemctl start pbs_mom
```

Installing Torque in High Availability Mode on Headless Nodes

The following procedure demonstrates a Torque installation in high availability (HA) mode on nodes with no local hard drive.

Requirements

- gcc (GCC) 4.1.2
- BASH shell
- Servers (these cannot be two VMs on the same hypervisor) configured the following way:
  - 2 main servers with identical architecture
    - `server1` — Primary server running Torque with a file system share (this example uses NFS)
    - `server2` — Secondary server running with Torque with a file system share (this example uses NFS)
  - Compute nodes
  - `fileServer` — A shared file system server (this example uses NFS)

To install Torque in HA mode on a node with no local hard drive

1. Stop all firewalls or update your firewall to allow traffic from Torque services.

   - Red Hat 6 or SUSE 11-based systems

     ```
     > service iptables stop
     > chkconfig iptables off
     ```

   - Red Hat 7 or SUSE 12-based systems

     ```
     > systemctl stop firewalld
     > systemctl disable firewalld
     ```

   If you are unable to stop the firewall due to infrastructure restriction, open the following ports:

   - `15001[tcp,udp]`
   - `15002[tcp,udp]`
   - `15003[tcp,udp]`

2. Disable SELinux

   ```
   > vi /etc/sysconfig/selinux

   SELINUX=disabled
   ```

3. Update your main `~/.bashrc` profile to ensure you are always referencing the applications to be installed on all servers.

```
# Torque
export TORQUE_HOME=/var/spool/torque

# Library Path

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${TORQUE_HOME}/lib

# Update system paths
export PATH=${TORQUE_HOME}/bin:${TORQUE_HOME}/sbin:${PATH}
```

4. Verify `server1` and `server2` are resolvable via either DNS or looking for an entry in the `/etc/hosts` file.

5. Configure the NFS Mounts by following these steps:

   a. Create mount point folders on `fileServer`.

   ```
   fileServer# mkdir -m 0755 /var/spool/torque
   ```

   b. Update `/etc/exports` on `fileServer`. The IP addresses should be that of `server2`.

   ```
   /var/spool/torque/ 192.168.0.0/255.255.255.0(rw,sync,no_root_squash)
   ```

   c. Update the list of NFS exported file systems.

   ```
   fileServer# exportfs -r
   ```

6. If the NFS daemons are not already running on `fileServer`, start them.

   - Red Hat 6 or SUSE 11-based systems

   ```
   > service rpcbind restart
   > service nfs-server start
   > service nfs-lock start
   > service nfs-idmap start
   ```

   - Red Hat 7 or SUSE 12-based systems

   ```
   > systemctl restart rpcbind.service
   > systemctl start nfs-server.service
   > systemctl start nfs-lock.service
   > systemctl start nfs-idmap.service
   ```

7. Mount the exported file systems on `server1` by following these steps:

   a. Create the directory reference and mount them.

   ```
   server1# mkdir /var/spool/torque
   ```

   Repeat this process for `server2`.

   b. Update `/etc/fstab` on `server1` to ensure that NFS mount is performed on startup.

```
fileServer:/var/spool/torque/server_priv /var/spool/torque/server_priv nfs
rsize= 8192,wsize=8192,timeo=14,intr
```

Repeat this step for `server2`.

8. Install Torque by following these steps:

   a. Download and extract Torque6.1.0 on `server1`.

```
server1# wget http://github.com/adaptivecomputing/torque/branches/6.1.0/torque-
6.1.0.tar.gz
server1# tar -xvzf torque-6.1.0.tar.gz
```

   b. Navigate to the Torque directory and compile Torque with the HA flag on server1.

```
server1# configure --prefix=/var/spool/torque
server1# make
server1# make install
server1# make packages
```

   c. If the installation directory is shared on both head nodes, then run `make install` on `server1`.

```
server1# make install
```

   If the installation directory is not shared, repeat step 8a-b (downloading and installing Torque) on `server2`.

9. Start trqauthd.

   • Red Hat 6 or SUSE 11-based systems

```
server1# service trqauthd start
```

   • Red Hat 7 or SUSE 12-based systems

```
server1# systemctl start trqauthd
```

10. Configure Torque for HA.

   a. List the host names of all nodes that run pbs_server in the `torque/server_name` file. You must also include the host names of all nodes running pbs_server in the `torque/server_name` file of each MOM node. The syntax of `torque/server_name` is a comma-delimited list of host names.

```
server1,server2
```

   b. Create a simple queue configuration for Torque job queues on server1.

```
server1# pbs_server -t create
server1# qmgr -c "set server scheduling=true"
server1# qmgr -c "create queue batch queue_type=execution"
server1# qmgr -c "set queue batch started=true"
server1# qmgr -c "set queue batch enabled=true"
server1# qmgr -c "set queue batch resources_default.nodes=1"
server1# qmgr -c "set queue batch resources_default.walltime=3600"
server1# qmgr -c "set server default_queue=batch"
```

> 🛈 Because `TORQUE_HOME` is a shared drive, you do not need to repeat this step on `server2`.

c. Add the root users of Torque to the Torque configuration as an operator and manager.

```
server1# qmgr -c "set server managers += root@server1"
server1# qmgr -c "set server managers += root@server2"
server1# qmgr -c "set server operators += root@server1"
server1# qmgr -c "set server operators += root@server2"
```

> 🛈 Because `TORQUE_HOME` is a shared drive, you do not need to repeat this step on `server2`.

d. You must update the lock file mechanism for Torque in order to determine which server is the primary. To do so, use the `lock_file_update_time` and `lock_file_check_time` parameters. The primary pbs_server will update the lock file based on the specified `lock_file_update_time` (default value of 3 seconds). All backup pbs_servers will check the lock file as indicated by the `lock_file_check_time` parameter (default value of 9 seconds). The `lock_file_update_time` must be less than the `lock_file_check_time`. When a failure occurs, the backup pbs_server takes up to the `lock_file_check_time` value to take over.

```
server1# qmgr -c "set server lock_file_check_time=5"
server1# qmgr -c "set server lock_file_update_time=3"
```

> 🛈 Because `TORQUE_HOME` is a shared drive, you do not need to repeat this step on `server2`.

e. List the servers running pbs_server in the Torque`acl_hosts` file.

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

> 🛈 Because `TORQUE_HOME` is a shared drive, you do not need to repeat this step on `server2`.

f. Restart the running pbs_server in HA mode.

- Red Hat 6 or SUSE 11-based systems

```
service pbs_server stop
```

- Red Hat 7 or SUSE 12-based systems

```
systemctl stop pbs_server
```

g. Start the pbs_server on the secondary server.

- Red Hat 6 or SUSE 11-based systems

```
service pbs_server start
```

- Red Hat 7 or SUSE 12-based systems

```
systemctl start pbs_server
```

11. Check the status of Torque in HA mode.

```
server1# qmgr -c "p s"
server2# qmgr -c "p s"
```

> *The commands above returns all settings from the active Torque server from either node.*

a. Drop one of the pbs_servers to verify that the secondary server picks up the request.

- Red Hat 6 or SUSE 11-based systems

```
service pbs_server stop
```

- Red Hat 7 or SUSE 12-based systems

```
systemctl stop pbs_server
```

b. Stop the pbs_server on `server2` and restart pbs_server on `server1` to verify that both nodes can handle a request from the other.

12. Install a pbs_mom on the compute nodes.

a. On `server1` or `server2`, configure the nodes file to identify all available MOMs. To do so, edit the `/ var/spool/torque/server_priv/nodes` file.

```
node1 np=2
node2 np=2
```

> ⓘ Change the `np` flag to reflect number of available processors on that node.

b. Recycle the pbs_servers to verify that they pick up the MOM configuration.

- Red Hat 6 or SUSE 11-based systems

```
service pbs_server stop
```

- Red Hat 7 or SUSE 12-based systems

```
systemctl stop pbs_server
```

> ⓘ You can specify command line arguments for pbs_server using the PBS_ARGS environment variable in the /etc/sysconfig/pbs_server file. Set PBS_ARGS=--ha -l <host>:<port> where <host> is the name of the alternate server node and <port> is the port on which pbs_server on the alternate server node is listening (default 15004).

c. Start the pbs_mom on each execution node.

- Red Hat 6 or SUSE 11-based systems

```
service pbs_mom start
```

- Red Hat 7 or SUSE 12-based systems

```
systemctl start pbs_mom
```

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.

- Red Hat 6 or SUSE 11-based systems

```
[root@server1]$ pbs_server --ha
[root@server2]$ pbs_server --ha
```

- Red Hat 7 or SUSE 12-based systems

```
[root@server1]$ systemctl start pbs_server
[root@server2]$ systemctl start pbs_server
```

Related Topics

Setting Server Policies
Queue Configuration

Setting min_threads and max_threads

There are two threadpools in Torque, one for background tasks and one for incoming requests from the MOMs and through the API (client commands, Moab, and so forth). The min_threads and max_threads parameters control the number of total threads used for both, not for each individually. The incoming requests' threadpool has three-quarters of **min_threads** for its minimum, and three-quarters of **max_threads** for its maximum, with the background pool receiving the other one-quarter.

Additionally, pbs_server no longer allows incoming requests to pile up indefinitely. When the threadpool is too busy for incoming requests, it indicates such, returning PBSE_SERVER_BUSY with the accompanying message that "Pbs Server is currently too busy to service this request. Please retry this request." The threshold for this message, if the request is from a manager, is that at least two threads be available in the threadpool. If the request comes from a non-manager, 5% of the threadpool must be available for the request to be serviced. Note that availability is calculated based on the maximum threads and not based on the current number of threads allocated.

If an undesirably large number of requests are given a busy response, one option is to increase the number of maximum threads for the threadpool. If the load on the server is already very high, then this is probably not going to help, but if the CPU load is lower, then it may help. Remember that by default the threadpool shrinks down once the extra threads are no longer needed. This is controlled via the thread_idle_seconds server parameter.

> ⓘ Any change in the **min_threads**, **max_threads**, or **thread_idle_seconds** parameters requires a restart of pbs_server to take effect.

Chapter 6 Integrating Schedulers for Torque

Selecting the cluster scheduler is an important decision and significantly affects cluster utilization, responsiveness, availability, and intelligence. The default Torque scheduler, pbs_sched, is very basic and will provide poor utilization of your cluster's resources. Other options, such as Maui Scheduler or Moab Workload Manager, are highly recommended. If you are using Maui or Moab, see **Moab-Torque Integration Guide** on page 1 in the *Moab Workload Manager Administrator Guide*. If using pbs_sched, simply start the pbs_sched daemon.

> **ℹ** If you are installing Moab Cluster Manager, Torque and Moab were configured at installation for interoperability and no further action is required.

Chapter 7 Configuring Data Management

This chapter provides instructions to configure Torque for data management purposes. For example, if you want to copy stdout and stderr files back to the submit host.

In this chapters:

- SCP Setup
- NFS and Other Networked Filesystems
- File stage-in/stage-out

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
- Copying Public SSH Key to Each Destination Host
- Configuring the SSH Daemon on Each Destination Host
- Validating Correct SSH Configuration
- Enabling Bi-Directional SCP Access
- Compiling Torque to Support SCP
- Troubleshooting

Related Topics

Configuring Data Management

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
>
> Copying Public SSH Key to Each Destination Host

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
>
> SCP Setup

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 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

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

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

Compiling Torque to Support SCP

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

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

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

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

Related Topics

SCP Setup

Troubleshooting

If, after following all of the instructions in this section (see SCP Setup), 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

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](#)

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

Chapter 8 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
- Open MPI

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](#)).

Related Topics

[MPI (Message Passing Interface) Support](#)

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](#).

TM Aware

To make use of Moab HPC Suite's TM interface, MPI must be configured to be TM aware.

Use these guidelines:

1. If you have installed from source, you need to use "./configure --with-tm" when you configure and make `openmpi`.

2. Run `mpirun` *without* the `-machinefile`. Moab HPC Suite will copy down the environment PATH and Library path down to each sister MOM. If `-machinefile` is used, `mpirun` will bypass the TM interface.

Example 8-1: Without TM aware

```
[jbooth@support-mpi1 ~]$ /usr/lib64/openmpi/bin/mpirun -np 4 -machinefile $PBS_
NODEFILE echo.sh
=============
support-mpi1
=============
/usr/lib64/openmpi/bin:/usr/lib64/openmpi/bin:/usr/lib64/qt-
3.3/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/opt/mo
ab/bin:/opt/moab/sbin:/home/jbooth/bin

/usr/lib64/openmpi/lib:/usr/lib64/openmpi/lib

=============
support-mpi1
=============
/usr/lib64/openmpi/bin:/usr/lib64/openmpi/bin:/usr/lib64/qt-
3.3/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/opt/mo
ab/bin:/opt/moab/sbin:/home/jbooth/bin

/usr/lib64/openmpi/lib:/usr/lib64/openmpi/lib

=============
support-mpi2
=============
/usr/lib64/openmpi/bin:/usr/lib64/openmpi/bin:/usr/lib64/qt-
3.3/bin:/usr/local/bin:/bin:/usr/bin

/usr/lib64/openmpi/lib:/usr/lib64/openmpi/lib:

=============
support-mpi2
=============
/usr/lib64/openmpi/bin:/usr/lib64/openmpi/bin:/usr/lib64/qt-
3.3/bin:/usr/local/bin:/bin:/usr/bin

/usr/lib64/openmpi/lib:/usr/lib64/openmpi/lib:
```

The paths, /opt/moab/bin and /opt/moab/sbin, were not passed down to the sister MOMs.

Example 8-2: With TM aware

```
[jbooth@support-mpi1 ~]$ /usr/local/bin/mpirun -np 4 echo.sh
=============
support-mpi1
=============
/usr/local/bin:/usr/local/bin:/usr/lib64/qt-
3.3/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/opt/mo
ab/bin:/opt/moab/sbin:/home/jbooth/bin

/usr/local/lib:/usr/local/lib


=============
support-mpi1
=============
/usr/local/bin:/usr/local/bin:/usr/lib64/qt-
3.3/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/opt/mo
ab/bin:/opt/moab/sbin:/home/jbooth/bin

/usr/local/lib:/usr/local/lib


=============
support-mpi2
=============
/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/qt-
3.3/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/opt/mo
ab/bin:/opt/moab/sbin:/home/jbooth/bin

/usr/local/lib:/usr/local/lib:/usr/local/lib


=============
support-mpi2
=============
/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/qt-
3.3/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/opt/mo
ab/bin:/opt/moab/sbin:/home/jbooth/bin

/usr/local/lib:/usr/local/lib:/usr/local/lib
```

The paths, /opt/moab/bin and /opt/moab/sbin, were passed down to the sister MOMs.

Related Topics

MPI (Message Passing Interface) Support

Chapter 9 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
- Utilization
- Node States

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 (tot-mem)	virtual memory/Swap	Virtual memory/Swap is monitored and reported in kilobytes.

Utilization

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

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

Node States

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

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

Chapter 10 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.

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

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

```
08/26/2014 17:07:44;Q;11923.napali;queue=batch
08/26/2014 17:07:50;S;11923.napali;user=dbeer group=company jobname=STDIN queue=batch
ctime=1409094464 qtime=1409094464 etime=1409094464 start=1409094470 owner=dbeer@napali
exec_host=napali/0+napali/1+napali/2+napali/3+napali/4+napali/5+torque-devtest-
03/0+torque-devtest-03/1+torque-devtest-03/2+torque-devtest-03/3+torque-devtest-
03/4+torque-devtest-03/5 Resource_List.neednodes=2:ppn=6 Resource_List.nodect=2
Resource_List.nodes=2:ppn=6
08/26/2014 17:08:04;E;11923.napali;user=dbeer group=company jobname=STDIN queue=batch
ctime=1409094464 qtime=1409094464 etime=1409094464 start=1409094470 owner=dbeer@napali
exec_host=napali/0+napali/1+napali/2+napali/3+napali/4+napali/5+torque-devtest-
03/0+torque-devtest-03/1+torque-devtest-03/2+torque-devtest-03/3+torque-devtest-
03/4+torque-devtest-03/5 Resource_List.neednodes=2:ppn=6 Resource_List.nodect=2
Resource_List.nodes=2:ppn=6 session=11352 total_execution_slots=12 unique_node_count=2
end=1409094484 Exit_status=265 resources_used.cput=00:00:00 resources_used.mem=82700kb
resources_used.vmem=208960kb resources_used.walltime=00:00:14 Error_Path=/dev/pts/11
Output_Path=/dev/pts/11
```

> ℹ The value of `Resource_List.*` is the amount of resources requested, and the value of `resources_used.*` is the amount of resources actually used.

> ℹ *total_execution_slots* and *unique_node_count* display additional information regarding the job resource usage.

Chapter 11 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
- Enabling Job Logs

Job Log Location and Name

When job logging is enabled (see Enabling Job Logs.), 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
Job Logging

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
Job Logging

Chapter 12 NUMA and Torque

Torque supports two types of Non-Uniform Memory Architecture (NUMA) systems. This chapter serves as a central information repository for the various configuration settings involved when using either NUMA system configuration.

> ⚠️ Torque cannot be configured for both systems at the same.

In this chapter:

- Supported NUMA Systems on page 169
- NUMA-Aware Systems on page 169
- NUMA-Support Systems on page 197

Supported NUMA Systems

Torque supports these two NUMA system configurations:

- NUMA-Aware – Introduced with Torque version 6.0, this configuration supports multi-req jobs and jobs that span hosts. Moab version 9.0 and later is also required.

- NUMA-Suppport – Introduced with Torque version 3.0, this configuration supports only a single instance for pbs_mom that as treated as if there where were multiple nodes running in the cluster. This configuration is only for large-scale SLES systems using SGI Altix and UV hardware.

NUMA-Aware Systems

This topic serves as a central information repository for NUMA-aware systems. This topic provides basic information and contains links to the various NUMA-aware topics found throughout the documentation.

> ℹ️ Support for NUMA-aware systems is available only with Torque Resource Manager 6.0 and later and Moab Workload Manager 9.0 and later.

In this topic:

- About NUMA-Aware Systems on page 170
- Installation and Configuration on page 171
- Job Resource Requests on page 171
- Job Monitoring on page 172
- Moab/Torque NUMA Configuration on page 172

About NUMA-Aware Systems

The NUMA-aware architecture is a hardware design which separates its cores into multiple clusters where each cluster has its own local memory region and still allows cores from one cluster to access all memory in the system. However, if a processor needs to use memory that is not its own memory region, it will take longer to access that (remote) memory. For applications where performance is crucial, preventing the need to access memory from other clusters is critical.

Torque uses cgroups to better manage cpu and memory accounting, memory enforcement, cpuset management, and binding jobs to devices such as MICs and GPUs. Torque will try to place jobs which request GPUs or MICs on NUMA nodes next to the GPU or MIC device to be used.

PCIe devices are similar to cores in that these devices will be closer to the memory of one NUMA node than another. Examples of PCIe devices are GPUs, NICs, disks, etc.

The resources of a processor chip have a hierarchy. The largest unit is a socket. A socket can contain one or more NUMA nodes with its cores and memory. A NUMA node will contain a set of cores and threads and memory which is local to the NUMA node. A core may have 0 or more threads.

- A socket refers to the physical location where a processor package plugs into a motherboard. The processor that plugs into the motherboard is also known as a socket. The socket can contain one or more NUMA nodes.

- A core is an individual execution unit within a processor that can independently execute a software execution thread and maintains its execution state separate from the execution state of any other cores within a processor.

- A thread refers to a hardware-based thread execution capability. For example, the Intel Xeon 7560 has eight cores, each of which has hardware that can effectively execute two software execution threads simultaneously, yielding 16 threads.

The following image is a simple depiction of a NUMA-aware architecture. In this example, the system has two NUMA nodes with four cores per NUMA node. The cores in each NUMA node have access to their own memory region but they can also access the memory region of the other NUMA node through the inter-connect.

> ℹ If the cores from NUMA chip 0 need to get memory from NUMA chip 1 there will be a greater latency to fetch the memory.
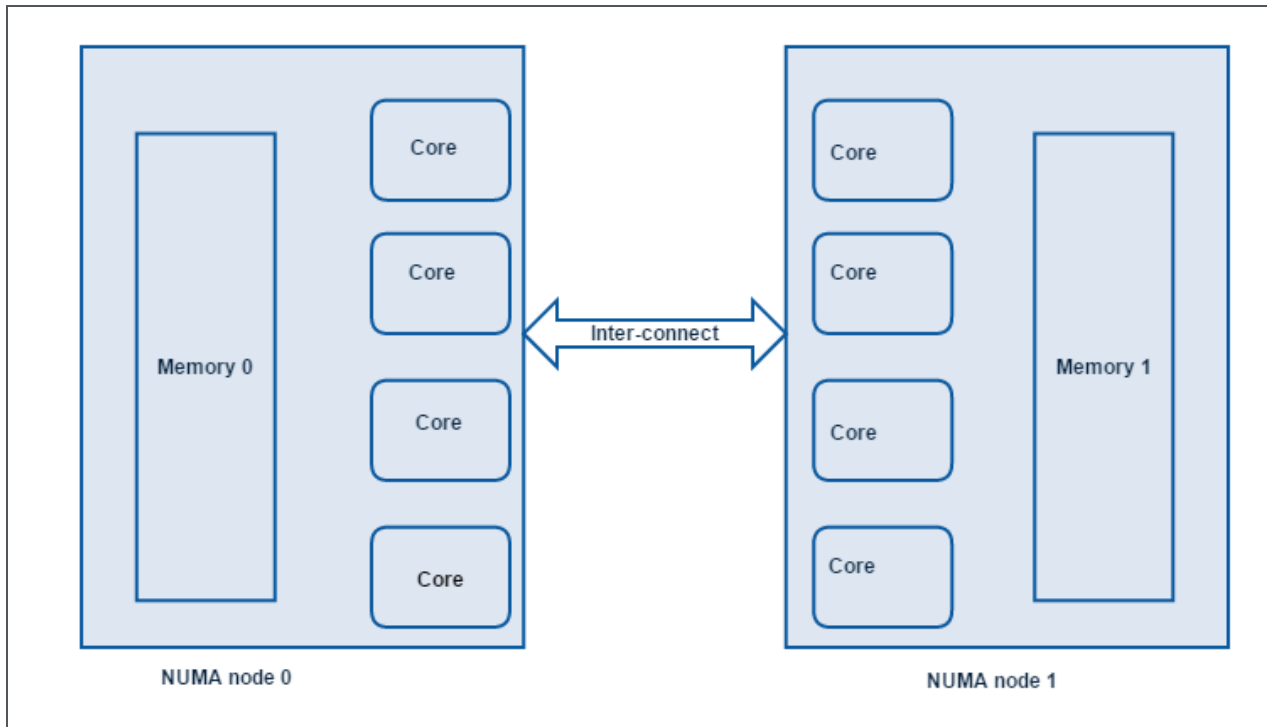
Installation and Configuration

Once Torque is first installed, you need to perform configuration steps.

See:

- Torque NUMA-Aware Configuration on page 57

Job Resource Requests

NUMA-aware resources can be requested at the time of job submission using the qsub/msub -L parameter. In addition, the req_infomation_max and req_information_min queue attributes let you specify the maximum and minimum resource limits allowed for jobs submitted to a queue.

> ⓘ Jobs requesting resources with -L can be run via qrun without a hostlist.

See:

- Requesting NUMA-Aware Resources on page 83
- -L NUMA Resource Request on page 185
- Queue Attributes on page 409

Job Monitoring

When using NUMA-aware, job resources are tracked per task. `qstat -f` produces a new category of information that begins with the " req_information" keyword. Following each "req_information keyword" is another keyword giving information about how the job was allocated. When the job has completed, the output will also include the per task resident memory used and per task cpu time used.

See

- [Monitoring NUMA Job Task Placement on page 90](#)

Moab/Torque NUMA Configuration

Moab does not require special configuration to support this NUMA-aware system. However, there are a few Moab-specific things that would be helpful to know and understand.

See

- [Using NUMA with Moab](#) in the *Moab Workload Manager Administrator Guide*

NUMA Tutorials

This section contains links to tutorials and other documentation useful in understanding NUMA-Aware systems.

In section:

- [NUMA Primer on page 172](#)
- [How NUMA Places Jobs on page 181](#)
- [NUMA Discovery and Persistence on page 184](#)

Related Topics

[NUMA-Aware Systems on page 169](#)

NUMA Primer

Torque 6.0 provides users with two brand new major features. First, cgroups are now used to manage each job. Second, resource request syntax has been updated, which gives users the ability to request resources on a per task basis, have multiple asymmetric resource requests in the same job, and control where jobs execute on cores and memory within NUMA hardware.

Control groups (cgroups) provide the ability to partition sets of tasks and their children into hierarchical groups with specialized behavior. In RHEL 7, the

default directory for cgroups became /sys/fs/cgroup. The following examples use this standard.

> ℹ If you are building with cgroups enabled, you must have boost version 1.41 or later.

Torque cgroup Hierarchy

Torque 6.0 only uses the cpu, devices, cpuacct, cpuset, and memory subsystems. While cpu and devices subsystems are required for Torque 6.0 they are not yet fully implemented. When pbs_mom is initialized it creates a sub-directory named torque in each of the five subsystem directories. When a job is started on the MOM a directory that is the full job id is created under each torque directory. The following is an "ls -al" submission command example from the cpuset/torque hierarchy:

```
total 0
drwxr-xr-x 3 root root 0 Aug 28 13:36 .
drwxr-xr-x 4 root root 0 Aug 28 13:35 ..
drwx------ 4 root root 0 Aug 31 10:20 1301.hosta
-rw-r--r-- 1 root root 0 Aug 28 13:35 cgroup.clone_children
--w--w--w- 1 root root 0 Aug 28 13:35 cgroup.event_control
-rw-r--r-- 1 root root 0 Aug 28 13:35 cgroup.procs
-rw-r--r-- 1 root root 0 Aug 28 13:35 cpuset.cpu_exclusive
-rw-r--r-- 1 root root 0 Aug 28 13:35 cpuset.cpus
-rw-r--r-- 1 root root 0 Aug 28 13:35 cpuset.mem_exclusive
-rw-r--r-- 1 root root 0 Aug 28 13:35 cpuset.mem_hardwall
-rw-r--r-- 1 root root 0 Aug 28 13:35 cpuset.memory_migrate
-r--r--r-- 1 root root 0 Aug 28 13:35 cpuset.memory_pressure
-rw-r--r-- 1 root root 0 Aug 28 13:35 cpuset.memory_spread_page
-rw-r--r-- 1 root root 0 Aug 28 13:35 cpuset.memory_spread_slab
-rw-r--r-- 1 root root 0 Aug 28 13:35 cpuset.mems
-rw-r--r-- 1 root root 0 Aug 28 13:35 cpuset.sched_load_balance
-rw-r--r-- 1 root root 0 Aug 28 13:35 cpuset.sched_relax_domain_level
-rw-r--r-- 1 root root 0 Aug 28 13:35 notify_on_release
-rw-r--r-- 1 root root 0 Aug 28 13:35 tasks
```

Line 4 shows that the subdirectory is "1301.hosta". This is the cpuset cgroup for job "1301.hosta". If you were to issue an ls command on the "1301.hosta" subdirectory in this example, you would see the following:

```
total 0
drwx------ 4 root root 0 Aug 31 10:24 .
drwxr-xr-x 3 root root 0 Aug 31 10:22 ..
-rw-r--r-- 1 root root 0 Aug 31 10:24 cgroup.clone_children
--w--w--w- 1 root root 0 Aug 31 10:24 cgroup.event_control
-rw-r--r-- 1 root root 0 Aug 31 10:24 cgroup.procs
-rw-r--r-- 1 root root 0 Aug 31 10:24 cpuset.cpu_exclusive
-rw-r--r-- 1 root root 0 Aug 31 10:24 cpuset.cpus
-rw-r--r-- 1 root root 0 Aug 31 10:24 cpuset.mem_exclusive
-rw-r--r-- 1 root root 0 Aug 31 10:24 cpuset.mem_hardwall
-rw-r--r-- 1 root root 0 Aug 31 10:24 cpuset.memory_migrate
-r--r--r-- 1 root root 0 Aug 31 10:24 cpuset.memory_pressure
-rw-r--r-- 1 root root 0 Aug 31 10:24 cpuset.memory_spread_page
-rw-r--r-- 1 root root 0 Aug 31 10:24 cpuset.memory_spread_slab
```

```
-rw-r--r-- 1 root root 0 Aug 31 10:24 cpuset.mems
-rw-r--r-- 1 root root 0 Aug 31 10:24 cpuset.sched_load_balance
-rw-r--r-- 1 root root 0 Aug 31 10:24 cpuset.sched_relax_domain_level
-rw-r--r-- 1 root root 0 Aug 31 10:24 notify_on_release
drwx------ 2 root root 0 Aug 31 10:24 R0.t0
drwx------ 2 root root 0 Aug 31 10:24 R0.t1
-rw-r--r-- 1 root root 0 Aug 31 10:24 tasks
-rw-r--r-- 1 root root 0 Aug 28 13:35 tasks
```

For this job the -L resource request was:

```
qsub -L tasks=2:lprocs=2
```

This job has a single request and two tasks. The R0 represents request 0 and the t0 and t1 represent the two tasks. In this case, cpuset information would be set for each task in their respective subdirectories. The cpu, cpuacct, memory and devices subsystems also utilize the same subdirectory structure.

cpuset subsystem

The Linux cpuset functionality was integrated into cgroups so that when Torque is configured with the "--enable-cgroups" option, cpuset functionality is also included. When jobs are submitted using the -L resource request syntax. Torque allocates a cpu set and memory set for each task in the job request. Examples of how cpusets and memory sets are allocated will be shown in the examples at the end of this primer.

cpuacct subsystem

The cpuacct subsystem keeps track of cpu time used for a cgroup. Torque now uses the cpuacct data to calculate cpu time used for a job. Also when using the -L resource request, cpu time per task is also recorded. Another advantage of cgroups is that the accounting information of a job does not disappear when the job process exits. So if pbs_mom goes down for any reason while running jobs the cpu time and memory used can still be tracked when pbs_mom is restarted.

memory subsystem

The memory subsytem keeps track of the maximum memory used by a cgroup and also can be used to limit the maximum amount of resident memory a task can use or the maximum amount of swap a task can use. The -L resource request syntax has a memory and a swap option. Following are examples of how to request memory restrictions with the -L resource request.

```
qsub -L tasks=2:memory=300mb
```

Two tasks are created. The memory=300mb option restricts each task to a maximum of 300 megabytest of resident memory. If a task exceeds 300 mb, then the excess memory is sent to swap.

```
qsub -L tasks=2:swap=1Gb
```

> *Two tasks are created. The memory=300mb option restricts each task to a maximum of 300 megabytest of resident memory. If a task exceeds 300 mb, then the excess memory is sent to swap.*

> ℹ️ In order to be able to set swap and memory limits the Linux kernel must be built using the options CONFIG_MEMCG=y, CONFIG_MEMCG_SWAP=y and CONFIG_MEMCG_SWAP_ENABLED=y. For Red Hat 7-based systems, these options are set by default.

Resource Request 2.0

Following are several different types of -L resource requests. The examples show how to use the syntax to be able to have resources allocated which can best fit your job needs.

Single Resource Request With Two Tasks and Default settings

```
qsub -L tasks=2:lprocs=1
```

After this job runs, the summarized qstat -f output is shown:

```
Job Id: 1306.hosta
Job_Name = bigmem.sh
Job_Owner = knielson@hosta
resources_used.cput = 00:00:01
resources_used.energy_used = 0
resources_used.mem = 1956984kb
resources_used.vmem = 2672652kb
resources_used.walltime = 00:00:10
job_state = C
. . .
exit_status = 0
submit_args = -L tasks=2:lprocs=1 ../scripts/bigmem.sh
. . .
req_information.task_count.0 = 2
req_information.lprocs.0 = 1
req_information.thread_usage_policy.0 = allowthreads
req_information.hostlist.0 = hosta:ppn=2
req_information.task_usage.0.task.0.cpu_list = 0
req_information.task_usage.0.task.0.mem_list = 0
req_information.task_usage.0.task.0.memory_used = 976816kb
req_information.task_usage.0.task.0.cores = 0
req_information.task_usage.0.task.0.threads = 1
req_information.task_usage.0.task.0.host = hosta
req_information.task_usage.0.task.1.cpu_list = 4
req_information.task_usage.0.task.1.mem_list = 0
req_information.task_usage.0.task.1.memory_used = 976752kb
req_information.task_usage.0.task.1.cores = 0
req_information.task_usage.0.task.1.threads = 1
req_information.task_usage.0.task.1.host = hosta
```

resources_used is the same as in previous versions of Torque. In this job, 1 second of cpu time was used. 1956984kb of resident memory was used, but with the new -L syntax there is a new set of information which starts with req_ information. This is the per task information of the job.

Output	Description
req_information.task_count.0 = 2	Two tasks are requested for this resource request; named tasks 0 and 1 respectively.
req_information.lprocs.0 = 1	One logical processor is requested per task. The lprocs value becomes the number of processing units per task allocated in the cpuset.
req_information.thread_usage_policy.0 = allow-threads	The processing unit allocation policy for the task. allow-threads is the user-specified default policy. allowthreads uses the first available core or thread. Processing units allocated in the cpuset are adjacent to each other unless other processors are also allocated.
req_information.hostlist.0 = hosta:ppn=2	On hostname hosta, two processing units are necessary . A single resource request can run on more than one host.
req_information.task_usage.0.task.0.cpu_list = 0	The task_usage keyword refers to the per task resource usage. 0 is the processing unit assigned to this task. In req_information.task_usage.0.cpu_list.1, the processing unit assigned is 4. This particular hardware is a 4 core system with hyper threading. So the core numbering is (0,4), (1,5), (2,6) and (3.7). Because the thread_usage_policy is allow-threads, the first two processing units are taken by default.
req_information.task_usage.0.task0.mem_list = 0	Memory location 0 is allocated to this task.
req_information.task_usage.0.task0.memory_used = 976816kb	The amount of resident memory used at task 0 is 976816kb.
req_information.task_usage.0.task0.cores = 0	This is the number of cores used by the task. In this case no cores were used because the allowthreads policy uses only threads and not discrete cores.
req_information.task_usage.0.task0.host = hosta	The task was run on hostname hosta.

> ⓘ The information for req_information.task_usage.0.*task.1* as opposed to *task.0*, means that the information displayed is referring to what was performed on task 1, rather than task 0.

Multiple lprocs

```
qsub -L tasks=1:lprocs=2
```

Two logical processors are specified with one task. The output of this job is as follows:

```
req_information.task_count.0 = 1
req_information.lprocs.0 = 2
req_information.thread_usage_policy.0 = allowthreads
req_information.hostlist.0 = hosta:ppn=2
req_information.task_usage.0.task.0.cpu_list = 0,4
req_information.task_usage.0.task.0.mem_list = 0
req_information.task_usage.0.task.0.cores = 0
req_information.task_usage.0.task.0.threads = 2
req_information.task_usage.0.task.0.host = hosta
```

The req_information for this syntax shows a cpu_list with two processing units. 0 and 4 are the first two processing units available so they are in the cpu_list. Notice that now there are two threads running.

usecores

The following example shows how a request to use cores changes the cpu_list allocation.

```
qsub -L tasks=1:lprocs=4:usecores
```

```
req_information.task_count.0 = 1
req_information.lprocs.0 = 4
req_information.thread_usage_policy.0 = usecores
req_information.hostlist.0 = hosta:ppn=4
req_information.task_usage.0.task.0.cpu_list = 0-3
req_information.task_usage.0.task.0.mem_list = 0
req_information.task_usage.0.task.0.cores = 4
req_information.task_usage.0.task.0.threads = 8
req_information.task_usage.0.task.0.host = hosta
```

Output	Description
req_information.task_usage.0.task.0.cores = 4	Four cores are used for task 0.
req_information.task_usage.0.task.0.threads = 8	When a core is requested, any threads for that core are no longer available for use by another task. In this case, each core has two threads. As a result, when one core is used two threads are also used. In this case, 8 threads are used in total.

usethreads

```
qsub -L tasks=1:lprocs=4:usethreads
```

The output of this job is as follows:

```
req_information.task_count.0 = 1
req_information.lprocs.0 = 4
req_information.thread_usage_policy.0 = usecores
req_information.hostlist.0 = hosta:ppn=4
req_information.task_usage.0.task.0.cpu_list = 0,4,1,5
req_information.task_usage.0.task.0.mem_list = 0
req_information.task_usage.0.task.0.cores = 0
req_information.task_usage.0.task.0.threads = 4
req_information.task_usage.0.task.0.host = hosta
```

Requesting usethreads gives adjacent processing units 0,4,1,5 and uses only 4 threads as indicated by req_information.task_usage.0.task.0.threads = 4.

Multiple Resource Requests

The -L resource requests makes it easier to request asymmetric resources for a single job. For example, you might have a job which needs several processors on a host to do work but only one or two processors on another host. The -L syntax easily accommodates this.

```
qsub -L tasks=2:lprocs=6:usecores  -L tasks=1:lprocs=1:place=socket
```

```
req_information.task_count.0 = 2
req_information.lprocs.0 = 6
req_information.thread_usage_policy.0 = usecores
req_information.hostlist.0 = hostb:ppn=12
req_information.task_usage.0.task.0.cpu_list = 0-5
req_information.task_usage.0.task.0.mem_list = 0
req_information.task_usage.0.task.0.cores = 6
req_information.task_usage.0.task.0.threads = 12
req_information.task_usage.0.task.0.host = hostb
req_information.task_usage.0.task.1.cpu_list = 6-11
req_information.task_usage.0.task.1.mem_list = 1
req_information.task_usage.0.task.1.cores = 6
req_information.task_usage.0.task.1.threads = 12
req_information.task_usage.0.task.1.host = hostb
req_information.task_count.1 = 1
req_information.lprocs.1 = 1
req_information.socket.1 = 1
req_information.thread_usage_policy.1 = allowthreads
req_information.hostlist.1 = hostb:ppn=1
req_information.task_usage.1.task.0.cpu_list = 0
req_information.task_usage.1.task.0.mem_list = 0
req_information.task_usage.1.task.0.cores = 1
req_information.task_usage.1.task.0.threads = 1
req_information.task_usage.1.task.0.host = hostb
```

Output	Description
req_information.task_count.1=1	Only one task on request 1.
req_information.socket.1 = 1	One socket is requested and then allocated for use.

place Directives

The place directive takes one of five arguments: node, socket, numanode, core, and thread. The node, core, and thread arguments do not take an assignment, however socket and numanode can be assigned a number value requesting the number of sockets or numanodes per task. The use of "place=core" or "place=thread" is the equivalent of using the usecores or usethreads syntax.

When processes share the same memory cache and are run on adjacent cores or threads, the likelihood of swapping out a cache line is high. When memory needs to be fetched from primary memory instead of the cache processing execution times are increased and become less predictable. In these examples, Torque disables linearly allocating cores. To help ensure best performance by avoiding the sharing of caches between processors, cores are spread as far apart as possible.

The following examples show the results of each directive:

```
place=socket
```

If a socket is not given a number, it defaults to the number 1.

```
qsub -L tasks=2:lprocs=2:place=socket
```

This request allocates two tasks with two logical processors each. Each task is placed on its own socket.

```
req_information.task_count.0 = 2
req_information.lprocs.0 = 2
req_information.socket.0 = 1
req_information.thread_usage_policy.0 = allowthreads
req_information.hostlist.0 = hosta:ppn=4
req_information.task_usage.0.task.0.cpu_list = 0,3
req_information.task_usage.0.task.0.mem_list = 0
req_information.task_usage.0.task.0.cores = 2
req_information.task_usage.0.task.0.threads = 4
req_information.task_usage.0.task.0.host = hosta
req_information.task_usage.0.task.1.cpu_list = 6,9
req_information.task_usage.0.task.1.mem_list = 1
req_information.task_usage.0.task.1.cores = 2
req_information.task_usage.0.task.1.threads = 4
req_information.task_usage.0.task.1.host = hosta
cpuset_string = hosta:0,3,6,9
memset_string = hosta:0-1
```

For the last example the job was run on a dual socket host with 12 cores. Each core has one thread for a total of 24 processing units. Each socket has 6 cores and 12 threads. The cores for socket 0 are numbered 0, 1, 2, 3, 4. The cores for socket 1 are numbered 5, 6, 7, 8. Task.0 uses cores 0 and 3 and task.1 uses cores 6 and 9.

```
place=numanode=2
```

```
qsub -L tasks=2:lprocs=2:place=numanode=2
```

This request allocates two numanodes, one for each task.

```
 req_information.task_count.0 = 2
req_information.lprocs.0 = 2
req_information.numanode.0 = 2
req_information.thread_usage_policy.0 = allowthreads
req_information.hostlist.0 = hostb:ppn=2
req_information.task_usage.0.task.0.cpu_list = 0, 3
req_information.task_usage.0.task.0.mem_list = 0
req_information.task_usage.0.task.0.cores = 0
req_information.task_usage.0.task.0.threads = 0
req_information.task_usage.0.task.0.host = hostb
req_information.task_usage.0.task.1.cpu_list = 6, 9
req_information.task_usage.0.task.1.mem_list = 1
req_information.task_usage.0.task.1.cores = 2
req_information.task_usage.0.task.1.threads = 4
req_information.task_usage.0.task.1.host = hostb
```

pbsnodes and Dedicated Resources

When a resource is requested (core, numanode, socket, etc.), the entire resource is no longer available for other jobs to use, and enters a dedicated state. Starting with Torque 6.0, pbsnodes tracks total sockets, numanodes, cores and threads per node. pbsnodes also tracks dedicated sockets, numanodes, cores, and threads. Following is and example of node output in pbsnodes for Torque 6.0

```
state = free
power_state = Running
np = 12
ntype = cluster
status =
rectime=1441054213,macaddr=78:e3:b5:0a:c0:58,cpuclock=Fixed,varattr=,jobs=,state=fre
e...
mom_service_port = 15002
mom_manager_port = 15003
total_sockets = 2
total_numa_nodes = 2
total_cores = 12
total_threads = 12
dedicated_sockets = 0
dedicated_numa_nodes = 0
dedicated_cores = 0
dedicated_threads = 0
```

This node has a total of 2 sockets, 2 numanodes, 12 cores and 12 threads. The number of dedicated sockets, numanodes, cores, and threads are set to 0 indicating there are currently no jobs running on this nodes. If a job is run with a syntax of:

```
qsub -L tasks=2:lprocs=2
```

the number of dedicated threads becomes four.

Once a job is completed, dedicated_threads returns to 0.

Related Topics

How NUMA Places Jobs

This topic discusses how jobs are placed on their specific NUMA resources.

> ℹ️ In this topic, placing is defined as determining where on the node the job will go.

Moab determines where to place a job on a node and pbs_server places that job inside the node. pbs_server decides where to place the job based on the parameters specified by the job itself and optimal locality.

The Placement Algorithm

The following diagram shows the decision making process for each request not using the "place=" syntax.

Whenever possible, jobs are given placement preference next to the GPUs that they will use. If the job's tasks' memory and CPUs are available on the socket that has the GPUs, that job will be placed at that location.

> ℹ️ This placement fails if either there is no socket that contains all of the required resources, or if jobs are and the GPUs are on socket - but all of the cores are used by another job.

```
          ╱╲
         ╱  ╲
        ╱    ╲   Does the req fit inside      Yes  ⟹   ┌─────────────┐       ┌──────┐
        ╲    ╱   one numa node                          │ Place the req│  ⟹   │ Done │
         ╲  ╱                                           │ inside the   │       └──────┘
          ╲╱                                            │ numa node.   │
           │                                            └─────────────┘
           │ No
           ⬇
          ╱╲
         ╱  ╲
        ╱    ╲   Does the req fit inside      Yes  ⟹   ┌─────────────┐       ┌──────┐
        ╲    ╱   one socket?                            │ Place the req│  ⟹   │ Done │
         ╲  ╱                                           │ inside the   │       └──────┘
          ╲╱                                            │ socket.      │
           │                                            └─────────────┘
           │ No
           ⬇
          ╱╲
         ╱  ╲
        ╱    ╲   Do individual tasks fit      Yes  ⟹   ┌─────────────┐
        ╲    ╱   inside numa nodes?                     │ Place as many│
         ╲  ╱                                           │ tasks as fit in│
          ╲╱                                            │ individual   │
           │                                            │ numa nodes.  │
           │ No                                         └─────────────┘
           │                                                   │
           │                                                   ⬇
           │                                                  ╱╲
           │                                                 ╱  ╲
           │                                                ╱    ╲  Are there remaining   No ⟹  ┌──────┐
           │                                                ╲    ╱  tasks to place?              │ Done │
           │                                                 ╲  ╱                                └──────┘
           │                                                  ╲╱
           │                                                   │ Yes
           │        ⬅───────────────────────────────────────
           ⬇
          ╱╲
         ╱  ╲
        ╱    ╲   Do individual tasks fit      Yes  ⟹   ┌─────────────┐
        ╲    ╱   inside any sockets?                    │ Place as many│
         ╲  ╱                                           │ tasks as fit in│
          ╲╱                                            │ individual   │
           │                                            │ numa nodes.  │
           │                                            └─────────────┘
```

For jobs using the "place=" syntax, the decision making algorithm is much simpler. When the place is specified, it will become reserved and the job will be placed at that location.

> ⓘ For a job to occupy the user-requested place option, that option must be completely available for use.

The following is an example of a job submitted using the -L option.

```
-L tasks=1:lprocs=2:sockets=2
```

This job placed on two sockets with one core reserved per socket.

> ⓘ The use of the word "core" is intentional. If a "place=socket" or "place=numanode" is requested and the lprocs request is less that the number of cores inside the socket or NUMA node, then the job is given only cores.

Once pbs_server has determined where each task should run, that decision is stored in attributes on the job itself. The complete_req attribute stores where each task is allocated, and the mom reads that information to create appropriate cgroups for the job and for the entire task. This information is available to the user via qstat.

Related Topics

 NUMA Tutorials on page 172

NUMA Discovery and Persistence

Initial Discovery

First, The mom performs the initial discovery of the host's layout, including the number of sockets, numa nodes, pci devices, cores, and threads. This is done using the hwloc library. Next, the mom sends that information to pbs_server, which notes it. Last, pbs_server writes files with the node's layout information locally. Following restarts, node information is gathered from these files.

Job Placement Decisions

Job placement decisions are done by pbs_server so that it immediately knows which NUMA resources have been used by which job. As a result, the second a job starts or finishes the information for available numa resources is updated and accurate. This information is then communicated to the mom daemon.

For more information on how jobs are placed, see How NUMA Places Jobs on page 181.

Persistence Across Restarts.

To maintain correct usage information pbs_server writes files to a new directory in /server_priv/node_usage. The files are written in a json format. The following is a reqresentation of what these files look like:

```
# Simple Node
{"node":{"socket":{"os_index":0,"numanode":{"os_index":0,"cores":"0-
5","threads":"","mem":16435852}}}}

# More Complicated Node
"node":{"socket":{"os_index":0,"numanode":{"os_index":0,"cores":"0-
7","threads":"","mem":16775316}, "numanode":{"os_index":1,"cores":"8-
15","threads":"","mem":16777216}},"socket": {"os_index":1,"numanode":{"os_
index":2,"cores":"16-23","threads":"","mem":8388608}, "numanode":{"os_
index":3,"cores":"24-31","threads":"","mem":8388608}}}}
```

When jobs are present, an allocation object will also be there to record what resources are being used by the job. The allocation object will be beneath the numanode object, so it is possible to have more than one per job.

Related Topics

 NUMA Tutorials on page 172

-L NUMA Resource Request

The -L option is available in the qsub and msub commands to allow administrators the ability to place jobs at the "task" or "OS process" level to get maximum efficiency out of the available hardware.

Using the -L option requires a basic knowledge of the topologies of the available hardware where jobs will run. You will need to know how many cores, numanodes, sockets, etc. are available on the hosts within the cluster. The -L syntax is designed to allow for a wide variety of requests. However, if requests do not match the available hardware, you may have unexpected results.

In addition, multiple, non-symmetric resource requests can be made for the same job using the -L job submission syntax.

For example, the following command:

```
qsub -L tasks=4:lprocs=2:usecores:memory=500mb -L tasks=8:lprocs=4:memory=2gb
```

> *Creates two requests. The first request creates 4 tasks with two logical processors and 500 mb of memory per task. The logical processors are placed on cores. The second request calls for 8 tasks with 4 logical processors and 2 gb of memory per task. Logical processors may be placed on cores or threads since the default placement is allowthreads.*

This topic provides the -L option syntax and a description of the valid value and allocation options.

Syntax

```
 -L tasks=#[:lprocs=#|all]
[:{usecores|usethreads|allowthreads}]
[:place={socket|numanode|core|thread}[=#]{node}][:memory=#]
[:swap=#][:maxtpn=#][:gpus=#[:<mode>]][:mics=#][:gres=<gres>]
[:feature=<feature>]
[[:{cpt|cgroup_per_task}]|[:{cph|cgroup_per_host}]]
```

Valid Value

Value	Description
tasks	Specifies the quantity of job tasks for which the resource request describes the resources needed by a single task. • **Distributed memory systems** - A single task must run within a single compute node/server; i.e., the task's resources must all come from the same compute node/server. • **Shared memory systems** - A single task may run on multiple compute nodes; i.e., the task's resources may come from multiple compute nodes. This option is *required* for task-based resource allocation and placement. ```qsub -L tasks=4``` *Creates four tasks, each with one logical process. The tasks can be run on a core or thread (default allowthreads).*

Available Options

The following table identifiers the various allocation options you can specify per task.

Value	Description
lprocs	Specifies the quantity of "logical processors" required by a single task to which it will be pinned by its control-group (cgroup).

The "place" value specifies the total number of physical cores/threads to which a single task has exclusive access. The lprocs= keyword indicates the *actual* number of cores/threads to which the task has exclusive access for the task's cgroup to pin to the task.

- When :lprocs is specified, and nothing is specified for #, the default is 1.
- When :lprocs=all is specified, all cores or threads in any compute node/server's available resource locality placement specified by the "place" option is eligible for task placement (the user has not specified a quantity, other than "give me all logical processors within the resource locality or localities"), which allows a user application to take whatever it can get and adapt to whatever it receives, which cannot exceed one node.

```
qsub -L tasks=1:lprocs=4
```

One task is created which allocates four logical processors to the task. When the job is executed, the pbs_mom where the job is running will create a cpuset with four processors in the set. Torque will make a best effort to allocate the four processors next to each other but the placement is not guaranteed.

```
qsub -L tasks=1:lprocs=all:place=node
```

Places one task on a single node, and places all processing units in the cpuset of the task. The "lprocs=all" parameter specifies that the task will use all cores and/or threads available on the resource level requested.

Value	Description

usecores, usethreads, allow threads	The usecores, usethreads, and allowthreads parameters are used to indicate whether the cgroup pins cores, threads, or either to a task, respectively. If no logical processor definition is given, the default is allowthreads for backward-compatible Moab scheduler and Torque resource manager behavior.
	
	In this context, "cores" means an AMD Opteron core, a hyperthread-disabled Intel Xeon core, or thread 0 *and only thread 0* of a hyperthread-enabled Intel Xeon core. The term "threads" refers to a hyperthread-enabled Intel Xeon thread. Likewise, "either" refers to an AMD Opteron core, a hyperthread-disabled Intel Xeon core, or any thread of a hyperthread-enabled Intel Xeon.
	
	• **:usecores** – Denotes that the logical processor definition for a task resource request is a physical core. This means if a core has hyper-threading enabled, the task will use only thread 0 of the core.
	
	```qsub -L tasks=2:lprocs=2:usecores```
	
	Two tasks are allocated with two logical processors per task. The usecores parameter indicates the processor types must be a core or thread 0 of a hyper-threaded core.
	
	• **:usethreads** – Specifies the logical processor definition for a task resource request is a hardware-based thread or virtual core.
	
	```qsub -L tasks=2:lprocs=2:usethreads```
	
	Two tasks are allocated with two logical processors per task. The usethreads parameter indicates that any type of hardware-based thread or virtual core can be used.
	
	• **:allowthreads** – Specifies that the logical processor definition for a task resource request can be either a physical core (e.g. AMD Opteron), or hardware-based thread of a core (hyperthread-enabled Intel Xeon).
	
	```qsub -L tasks=2:lprocs=2:allowthreads```
	
	Two tasks are allocated with two logical processors per task. The allowthreads parameter indicates hardware threads or cores can be used..

Value	Description

place	Specifies placement of a single task on the hardware. Specifically, this designates what hardware resource locality level and identifies the quantity of locality-level resources. Placement at a specific locality level is always exclusive, meaning a job task has exclusive use of all logical processor and physical memory resources at the specified level of resource locality, even if it does not use them.
	
	Valid Options:
	
	ⓘ If a valid option is not specified, the usecusecores, usethreads, and allowthreads parameters are used.
	
	• **socket[=#]** – Refers to a socket within a compute node/server and specifies that each task is placed at the socket level with exclusive use of all logical processors and memory resources of the socket(s) allocated to a task. If a count is not specified, the default setting is 1.
	
	`qsub -L tasks=2:lprocs=4:place=socket`
	
	Two tasks are allocated with four logical processors each. Each task is placed on a socket where it will have exclusive access to all of the cores and memory of the socket. Although the socket may have more cores/threads than four, only four cores/threads will be bound in a cpuset per task per socket as indicated by "lprocs=4".
	
	• **numanode[=#]** – Refers to the numanode within a socket and specifies that each task is placed at the NUMA node level within a socket with exclusive use of all logical processor and memory resources of the NUMA node(s) allocated to the task. If a count is not given, the default value is 1. If a socket does not contain multiple numanodes, by default the socket contains one numanode.
	
	To illustrate the locality level to which this option refers, the following examples are provided:
	
	First, a Haswell-based Intel Xeon v3 processor with 10 or more cores is divided internally into two separate "nodes", each with an equal quantity of cores and its own local memory (referred to as a "numanode" in this topic).
	
	Second, an AMD Opteron 6xxx processor is a "multi-chip module" that contains two separate physical silicon chips each with its own local memory (referred to as a "numanode" in this topic).
	
	In both of the previous examples, a core in one "node" of the processor can access its own local memory faster than it can access the remote memory of the other "node" in the processor, which results in NUMA behavior.
	
	`qsub -L tasks=2:lprocs=4:place=numanode`
	
	Places a single task on a single numanode and the task has exclusive use of all the logical processors and memory of the numanode.

Value	Description

```
qsub -L tasks=2:lprocs=all:place=numanode=2
```

> *Allocates two tasks with each task getting two numanodes each. The "lprocs=all"*
> *specification indicates all of the cores/threads of each numanode will be bound in*
> *the cpuset for the task.*

- **core[=#]** – Refers to a core within a numanode or socket and specifies each task is placed at the core level within the numanode or socket and has exclusive use of all logical processor and memory resources of the core(s) allocated to the task. Whether a core has SMT/hyper-threading enabled or not is irrelevant to this locality level. If a number of cores is not specified, it will default to the number of lprocs specified.

> 🛈 The amount of cores specified must be greater than or equal to the number of lprocs available, otherwise the job submission will be rejected.

```
qsub -L tasks=2:place=core=2
```

> *Two tasks with one logical processor each will be placed on two cores per task.*

```
qsub -L tasks=2:lprocs=2:place=core
```

> *Two tasks are allocated with two logical processors per task. Each logical process*
> *will be assigned to one core each (two cores total, the same as the number of*
> *lprocs). Torque will attempt to place the logical processors on non-adjacent cores.*

- **thread[=#]** – Specifies each task is placed at the thread level within a core and has exclusive use of all logical processor and memory resources of the thread(s) allocated to a task.

  This affinity level refers to threads within a core and is applicable only to nodes with SMT or hyper-threading enabled. If a node does not have SMT or hyper-threading enabled, Moab will consider the node ineligible when allocating resources for a task. If a specific number of threads is not specified, it will default the the number of lprocs specified.

```
qsub -L tasks=2:lprocs=4:place=thread
```

> *Allocates two tasks, each with four logical processors, which can be bound to any*
> *thread. Torque will make a best effort to bind the threads on the same numanode*
> *but placement is not guaranteed. Because the amount of threads is not specified,*
> *Torque will place the number of lprocs requested.*

- **node** – Specifies that each task is placed at the node level and has exclusive use of all the resources of the node(s) allocated to a task. This locality level usually refers to a physical compute node, blade, or server within a cluster.

Value	Description
	``` qsub -L tasks=2:lprocs=all:place=node ``` *Two tasks are allocated with one task per node, where the task has exclusive access to all the resources on the node. The "lprocs=all" specification directs Torque to create a cpuset with all of the processing units on the node. The "place=node" speficiation also claims all of the memory for the node/server.*
memory	ℹ️ "memory" is roughly equivalent to the mem request for the qsub/msub -l resource request. However, with the -L qsub syntax, cgroups monitors the job memory usage and puts a ceiling on resident memory for each task of the job. Specifies the maximum resident memory allocated per task. Allowable suffixes are kb (kilobytes), mb (megabytes), gb (gigabytes), tb (terabyte), pb (petabytes), and eb (exabyte). If a suffix is not provided by the user, mb (megabytes) is default. ℹ️ If a task uses more resident memory than specified the excess memory is moved to `swap`. ``` qsub -L tasks=4:lprocs=2:usecores:memory=1gb ``` *Allocates four tasks with two logical processors each. Each task is given a limit of 1 gb of resident memory.* ``` qsub -L tasks=2:memory=3500 ``` *Allocates two tasks with 3500 mb (the suffix was not specified so megabytes is assumed).*

Value	Description

swap	Specifies the maximum allocated resident memory *and* swap space allowed per task.
	
	Allowable suffixes are kb (kilobytes), mb (megabytes), gb (gigabytes), tb (terabyte), pb (petabytes), and eb (exabyte). If a suffix is not given, mb (megabytes) is assumed.
	
	If a task exceeds the specified limit, the task will be killed; the associated job will be terminated.
	
	⚠️ If the swap limit is unable to be set, the job will still be allowed to run. All other cgroup-related failures will cause the job to be rejected.
	
	When requesting swap, it is not required that you give a value for the :memory option.
	   - If using :swap *without* a specified :memory value, Torque will supply a memory value up to the value of :swap; but not larger than available physical memory.
	
	`qsub -L tasks=4:lprocs=2:swap=4gb`
	
	Allocates four tasks with two logical processors each. Each task is given a combined limit of 4 gb of resident memory and swap space. If a task exceeds the limit, the task is terminated. - If using :swap *with* a specified :memory value, Torque will only supply resident memory up to the :memory value. The rest of the swap can *only* be supplied from the swap space.
	
	ℹ️ The :memory value *must* be smaller than or equal to the :swap value.
	
	`qsub -L tasks=2:memory=3.5gb:swap=5gb`
	
	Allocates two tasks and each task has up to 3.5 gb of resident memory and a maximum of 5 gb of swap. If a task exceed 3.5 gb of resident memory, the excess will be moved to the swap space. However, if the task exceed 5 gb of total swap, the task and job will be terminated.

maxtpn	Specifies the maximum tasks per node; where "#" is the maximum tasks allocated per physical compute node. This restricts a task type to no more than "#" tasks per compute node and allows it to share a node with other task types or jobs. For example, a communication-intensive task may share a compute node with computation-intensive tasks. ℹ️ The number of nodes and tasks per node will not be known until the job is run. `qsub -L tasks=7:maxtpn=4` *Allocates seven tasks but a maximum of four tasks can run on a single node.*

Value	Description

gpus	Specifies the quantity of GPU accelerators to allocate to a task, which requires placement at the locality-level to which an accelerator is connected or higher. *<MODE>* can be `exclusive_process`, `exclusive_thread`, or `reseterr`.
	
	The task resource request must specify placement at the numanode- (AMD only), socket-, or node-level. `place=core` and `place=thread` are invalid placement options when a task requests a PCIe-based accelerator device, since allowing other tasks to use cores and threads on the same NUMA chip or socket as the task with the PCIe device(s) would violate the consistent job execution time principle since these other tasks would likely interfere with the data transfers between the task's logical processors and its allocated accelerator(s).
	
	`:gpus=1`
	
	Allocates one GPU per task.
	
	`:gpus=2:exclusive_process:reseterr`
	
	Allocates two GPUs per task with exclusive access by process and resets error counters.

mics	Specifies the quantity of Intel MIC accelerators to allocate to a task, which requires placement at the locality-level to which a MIC is connected or higher.
	
	The task resource request must specify placement at the NUMA chip- (makes sense for AMD only), socket-, or node-level. place=core and place=thread are invalid placement options when a task requests a PCIe-based accelerator device since allowing other tasks to use cores and threads on the same NUMA chip or socket as the task with the PCIe device(s) would violate the consistent job execution time principle since these other tasks would likely interfere with the data transfers between the task's logical processors and its allocated accelerator(s).
	
	ⓘ Allocating resources for MICs operates in the exact same manner as for GPUs. See gpus.
	
	`:mics=1`
	
	Allocates on MIC per task.
	
	`:mics=2`
	
	Allocates two MICs per task.

Value	Description
gres	Specifies the quantity of a specific generic resource `<gres>` to allocate to a task. If a quantity is not given, it defaults to one.
	ⓘ Specify multiple GRES by separating them with commas and enclosing all the GRES names, their quantities, and the commas within single quotation marks.
	``` :gres=matlab=1 ```
	Allocates one Matlab license per task.
	``` :gres='dvd,blu=2' ```
	Allocates one DVD drive and two Blu-ray drives per task, represented by the "dvd" and "blu" generic resource names, respectively.
	ⓘ When scheduling, if a generic resource is node-locked, only compute nodes with the generic resource are eligible for allocation to a job task. If a generic resource is floating, it does not qualify or disqualify compute node resources from allocation to a job task.
feature	Specifies one or more node feature names used to qualify compute nodes for task resources; i.e., a compute node must have all ("&") or and ("|") of the specified feature name(s) assigned or the compute node's resources are ineligible for allocation to a job task.
	``` :feature=bigmem :feature='bmem&fio' :feature='bmem|fio' ```

Value	Description
cpt, cgroup_ per_task, cph, cgroup_ per_host	Specifies whether cgroups are created per-task or per-host. If submitting using msub, this information is passed through to Torque; there is no affect to Moab operations.

> ⓘ This option lets you specify how cgroups are created during job submission. This option can be used to override the Torque cgroup_per_task sever parameter. If this option is not specified, the server parameter value is used. See Server Parameters for more information.

- **:cpt, :cgroup_per_task** – Job request will have one cgroup created per task; all the processes on that host will be placed in the *first* task's cgroup.
- **:cph, :cgroup_per_host** – Job request will have one cgroup created per host; this is similar to pre-6.0 cpuset implementations.

> ⚠ Some MPI implementations only launch one process through the TM API, and then fork each subsequent process that should be launched on that host. If the job is set to have one cgroup per task, this means that all of the processes on that host will be placed in the *first* task's cgroup. Confirm that the cgroup_per_task Torque server parameter is set to FALSE (default) or specify :cph or :cgroup_per_host at job submission.
>
> If you know that your MPI will communicate each process launch to the mom individually, then set the cgroup_per_task Torque server parameter is set to TRUE or specify :cpt or :cgroup_per_task at job submission.

Related Topics

-
-

pbsnodes with NUMA-Awareness

When Torque is configured with NUMA-awareness and configured with --enable-groups, the number of total *and* the number of available sockets, numachips (numa nodes), cores, and threads are returned when the status of nodes are queried by Moab (a call is made to pbsnodes).

The example output that follows shows a node with two sockets, four numachips, 16 cores and 32 threads. In this example, no jobs are currently running on this node; therefore, the available resources are the same as the total resources.

```
torque-devtest-01
     state = free
     power_state = Running
     np = 16
     ntype = cluster
     status =
rectime=1412732948,macaddr=00:26:6c:f4:66:a0,cpuclock=Fixed,varattr=,jobs=,state=free,
netload=17080856592,gres=,loadave=10.74,ncpus=16,physmem=49416100kb,availmem=50056608k
b,totmem=51480476kb,idletime=29,nusers=2,nsessions=3,sessions=8665
8671 1994,uname=Linux torque-devtest-01 2.6.32-358.el6.x86_64 #1 SMP
Fri Feb 22 00:31:26 UTC 2013 x86_64,opsys=linux
     mom_service_port = 15002
     mom_manager_port = 15003
     total_sockets = 2
     total_numachips = 4
     total_cores = 16
     total_threads = 32
     available_sockets = 2
     available_numachips = 4
     available_cores = 16
     available_threads = 32
```

However, if a job requesting only a single core was started on this node, the pbsnodes output will look like:

```
torque-devtest-01
     state = free
     power_state = Running
     np = 16
     ntype = cluster
     jobs = 0/112.torque-devtest-01
     status =
rectime=1412732948,macaddr=00:26:6c:f4:66:a0,cpuclock=Fixed,varattr=,jobs=,state=free,
netload=17080856592,gres=,loadave=10.74,ncpus=16,physmem=49416100kb,availmem=50056608k
b,totmem=51480476kb,idletime=29,nusers=2,nsessions=3,sessions=8665
8671 1994,uname=Linux torque-devtest-01 2.6.32-358.el6.x86_64 #1 SMP
Fri Feb 22 00:31:26 UTC 2013 x86_64,opsys=linux
     mom_service_port = 15002
     mom_manager_port = 15003
     total_sockets = 2
     total_numachips = 4
     total_cores = 16
     total_threads = 32
     available_sockets = 1
     available_numachips = 3
     available_cores = 15
     available_threads = 30
```

In looking at the output for this example, you will see that even though only one core was requested the available sockets, numachip, cores and threads were all reduced. This is because the NUMA architecture is hierarchical: socket contains one or more numachips; a numachip contains two or more cores; cores contain one or more threads (one thread in the case of non-threaded cores). In order for a resource to be available, the entire resource must be free. When a job uses one core, the use of that core consumes part of the associated socket, and numa chip resources. As a result, the affected socket and numachip cannot be used when subsequent jobs request sockets and numachips as resources. Also, because the job asked for one core, the number of threads for that core are consumed. As a result, the number of threads available on the machine is reduced by the number of threads in the core.

As another example, suppose a user makes an job request and they want to use a socket. The pbsnodes output will look like:

```
torque-devtest-01
     state = free
     power_state = Running
     np = 16
     ntype = cluster
     jobs = 113.torque-devtest-01
     status =
rectime=1412732948,macaddr=00:26:6c:f4:66:a0,cpuclock=Fixed,varattr=,jobs=,state=free,
netload=17080856592,gres=,loadave=10.74,ncpus=16,physmem=49416100kb,availmem=50056608k
b,totmem=51480476kb,idletime=29,nusers=2,nsessions=3,sessions=8665
8671 1994,uname=Linux torque-devtest-01 2.6.32-358.el6.x86_64 #1 SMP
Fri Feb 22 00:31:26 UTC 2013 x86_64,opsys=linux
     mom_service_port = 15002
     mom_manager_port = 15003
     total_sockets = 2
     total_numachips = 4
     total_cores = 16
     total_threads = 32
     available_sockets = 1
     available_numa_chips = 2
     available_cores = 8
     available_threads = 16
```

In looking at the output in this example, you will see that not only are the available sockets reduced to one, but all of the numachips, cores, and threads associated with the socket are no longer available. In other words, by requesting a job placement of "socket" all of the resources of the socket are reserved and are no longer available to other jobs.

NUMA-Support Systems

This topic serves as a central information repository for NUMA-support systems. This topic provides basic information and contains links to the various NUMA-aware topics found throughout the documentation.

> ⓘ Support for NUMA-support systems is available only on large-scale SLES systems using SGI Altix and UV hardware and requires Torque 3.0 or later.

In this chapter:

About NUMA-Supported Systems

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.

Torque Installation and Configuration

To enable Torque for NUMA-support, you will need to add the `--enable-numa-support` option during the configure portion of the installation. You will also need create the mom.layout file and configure the server_priv/nodes file.

See:

Moab/Torque NUMA Configuration

Moab *requires* additional configuration to enable NUMA-support.

See:

Moab-NUMA-Support Integration Guide in the *Moab Workload Manager Administrator Guide*.

Chapter 13 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:

- Automatic Queue and Job Recovery
- Host Resolution
- Firewall Configuration
- Torque Log Files
- Using "tracejob" to Locate Job Failures
- Using GDB to Locate Job Failures
- Other Diagnostic Options
- Stuck Jobs
- Frequently Asked Questions (FAQ)
- Compute Node Health Check
- Debugging

Automatic Queue and Job Recovery

When pbs_server restarts and recovers a job but cannot find that job's queue, it will create a new queue with the original name, but with a ghost_queue attribute (as seen in qmgr) and then add the job to that queue. This will happen for each queue the server does not recognize. Ghost queues will not accept new jobs, but will allow the jobs in it to run and be in a running state. If users attempt to submit any new jobs to these queues, the user will get an error stating that the queue had an error on recovery, and is in a ghost state. Once the admin reviews and corrects the queue's settings, the admin may remove the ghost setting and then the queue will function normally.

See ghost_queue for more information.

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

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

Torque Log Files

pbs_server and pbs_mom Log Files

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

For both pbs_mom and pbs_server daemons, the log verbosity level can also be adjusted by setting the environment variable **PBSLOGLEVEL** to a value

between 0 and 7. Further, to dynamically change the log level of a running daemon, use the `SIGUSR1` and `SIGUSR2` signals to increase and decrease the active loglevel by one. Signals are sent to a process using the `kill` command.

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

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

trqauthd Log Files

As of Torque 4.1.3, trqauthd logs its events in the `TORQUE_HOME/client_logs` directory. It names the log files in the format *<YYYYMMDD>*, creating a new log daily as events occur.

> ⓘ You might see some peculiar behavior if you mount the `client_logs` directory for shared access via network-attached storage.
>
> When trqauthd first gets access on a particular day, it writes an "open" message to the day's log file. It also writes a "close" message to the last log file it accessed prior to that, which is usually the previous day's log file, but not always. For example, if it is Monday and no client commands were executed over the weekend, trqauthd writes the "close" message to Friday's file.
>
> Since the various trqauthd binaries on the submit hosts (and potentially, the compute nodes) each write an "open" and "close" message on the first access of a new day, you'll see multiple (seemingly random) accesses when you have a shared log.

The trqauthd records the following events along with the date and time of the occurrence:

- When trqauthd successfully starts. It logs the event with the IP address and port.
- When a user successfully authenticates with trqauthd.
- When a user fails to authenticate with trqauthd.
- When trqauthd encounters any unexpected errors.

Example 13-1: trqauthd logging sample

```
2012-10-05 15:05:51.8404 Log opened
2012-10-05 15:05:51.8405 Torque authd daemon started and listening on IP:port
101.0.1.0:12345
2012-10-10 14:48:05.5688 User hfrye at IP:port abc:12345 logged in
```

Related Topics

Troubleshooting

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


...
```

> ℹ 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`

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

> ℹ The value of `Resource_List.*` is the amount of resources requested, and the value of `resources_used.*` is the amount of resources actually used.

Related Topics

Troubleshooting

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.

> ℹ See the PBSCOREDUMP parameter for enabling creation of core files (see Debugging).

Related Topics

Troubleshooting

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

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

Frequently Asked Questions (FAQ)

- Cannot connect to server: error=15034
- Deleting 'stuck' jobs
- Which user must run Torque?
- Scheduler cannot run jobs - rc: 15003
- PBS_Server: pbsd_init, Unable to read server database
- qsub will not allow the submission of jobs requesting many processors
- qsub reports 'Bad UID for job execution'
- Why does my job keep bouncing from running to queued?
- How do I use PVM with Torque?
- My build fails attempting to use the TCL library

- My job will not start, failing with the message 'cannot send job to mom, state=PRERUN'
- How do I determine what version of Torque I am using?
- How do I resolve autogen.sh errors that contain "error: possibly undefined macro: AC_MSG_ERROR"?
- How do I resolve compile errors with libssl or libcrypto for Torque 4.0 on Ubuntu 10.04?
- Why are there so many error messages in the client logs (trqauthd logs) when I don't notice client commands failing?

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 `-p` command or can be removed manually using the following steps:

To remove job X

1. Shut down pbs_server.

```
> qterm
```

2. Remove job spool files.

```
> rm TORQUE_HOME/server_priv/jobs/X.SC TORQUE_HOME/server_priv/jobs/X.JB
```

3. Restart pbs_server

```
> pbs_server
```

Which user must run Torque?

Torque (pbs_server & pbs_mom) must be started by a user with root privileges.

Scheduler cannot run jobs - rc: 15003

For a scheduler, such as Moab or Maui, to control jobs with Torque, the scheduler needs to be run be a user in the server operators / managers list (see **qmgr**). The default for the server operators / managers list is root@localhost. For Torque to be used in a grid setting with Silver, the scheduler needs to be run as root.

PBS_Server: pbsd_init, Unable to read server database

If this message is displayed upon starting pbs_server it means that the local database cannot be read. This can be for several reasons. The most likely is a version mismatch. Most versions of Torque can read each other's databases. However, there are a few incompatibilities between OpenPBS and Torque. Because of enhancements to Torque, it cannot read the job database of an OpenPBS server (job structure sizes have been altered to increase functionality). Also, a compiled in 32-bit mode cannot read a database generated by a 64-bit pbs_server and vice versa.

To reconstruct a database (excluding the job database)

1. First, print out the old data with this command:

```
%> qmgr -c "p s"
#
# Create queues and set their attributes.
#
#
# Create and define queue batch
# create queue batch
set queue batch queue_type = Execution
set queue batch acl_host_enable = False
set queue batch resources_max.nodect = 6
set queue batch resources_default.nodes = 1
set queue batch resources_default.walltime = 01:00:00
set queue batch resources_available.nodect = 18
set queue batch enabled = True
set queue batch started = True
#
# Set server attributes.
#
set server scheduling = True
set server managers = griduser@oahu.icluster.org
set server managers += scott@*.icluster.org
set server managers += wightman@*.icluster.org
set server operators = griduser@oahu.icluster.org
set server operators += scott@*.icluster.org
set server operators += wightman@*.icluster.org
set server default_queue = batch
set server log_events = 511
set server mail_from = adm
set server resources_available.nodect = 80
set server node_ping_rate = 300
set server node_check_rate = 600
set server tcp_timeout = 6
```

2. Copy this information somewhere.

3. Restart pbs_server with the following command:

```
> pbs_server -t create
```

4. When you are prompted to overwrite the previous database, enter *y*, then enter the data exported by the qmgr command as in this example:

```
> cat data | qmgr
```

5. Restart pbs_server without the flags:

```
> qterm
> pbs_server
```

This will reinitialize the database to the current version.

ⓘ Reinitializing the server database will reset the next jobid to 1

qsub will not allow the submission of jobs requesting many processors

Torque's definition of a node is context sensitive and can appear inconsistent. The **qsub** `-l nodes=<X>` expression can at times indicate a request for X processors and other time be interpreted as a request for X nodes. While qsub allows multiple interpretations of the keyword nodes, aspects of the Torque server's logic are not so flexible. Consequently, if a job is using `-l nodes` to specify processor count and the requested number of processors exceeds the available number of physical nodes, the server daemon will reject the job.

To get around this issue, the server can be told it has an inflated number of nodes using the `resources_available` attribute. To take effect, this attribute should be set on both the server and the associated queue as in the example below. See **resources_available** for more information.

```
> qmgr
Qmgr: set server resources_available.nodect=2048
Qmgr: set queue batch resources_available.nodect=2048
```

> ℹ The pbs_server daemon will need to be restarted before these changes will take effect.

qsub reports 'Bad UID for job execution'

```
[guest@login2]$ qsub test.job
qsub: Bad UID for job execution
```

Job submission hosts must be explicitly specified within Torque or enabled via RCmd security mechanisms in order to be trusted. In the example above, the host 'login2' is not configured to be trusted. This process is documented in Configuring Job Submission Hosts.

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)

> ℹ Access can be managed by rsh/ssh without passwords between the batch nodes, but denying it from anywhere else, including the interactive nodes. This can be done with xinetd and sshd configuration (root is allowed to ssh everywhere). This way, the pvm daemons can be started and killed from the job script.

The problem is that this setup allows the users to bypass the batch system by writing a job script that uses rsh/ssh to launch processes on the batch nodes. If there are relatively few users and they can more or less be trusted, this setup can work.

My build fails attempting to use the TCL library

Torque builds can fail on TCL dependencies even if a version of TCL is available on the system. TCL is only utilized to support the xpbsmon client. If your site does not use this tool (most sites do not use xpbsmon), you can work around this failure by rerunning `configure` with the `--disable-gui` argument.

My job will not start, failing with the message 'cannot send job to mom, state=PRERUN'

If a node crashes or other major system failures occur, it is possible that a job may be stuck in a corrupt state on a compute node. Torque 2.2.0 and higher automatically handle this when the `mom_job_sync` parameter is set via **qmgr** (the default). For earlier versions of Torque, set this parameter and restart the pbs_mom daemon.

This error can also occur if not enough free space is available on the partition that holds Torque.

How do I determine what version of Torque I am using?

There are times when you want to find out what version of Torque you are using. An easy way to do this is to run the following command:

```
qmgr

> qmgr -c "p s" | grep pbs_ver
```

How do I resolve autogen.sh errors that contain "error: possibly undefined macro: AC_MSG_ERROR"?

Verify the `pkg-config` package is installed.

How do I resolve compile errors with libssl or libcrypto for Torque 4.0 on Ubuntu 10.04?

When compiling Torque 4.0 on Ubuntu 10.04 the following errors might occur:

```
libtool: link: gcc -Wall -pthread -g -D_LARGEFILE64_SOURCE -o .libs/trqauthd trq_auth_
daemon.o trq_main.o -ldl -lssl -lcrypto -L/home/adaptive/torques/torque-
4.0.0/src/lib/Libpbs/.libs /home/adaptive/torques/torque-
4.0.0/src/lib/Libpbs/.libs/libtorque.so -lpthread -lrt -pthread
/usr/bin/ld: cannot find -lssl
collect2: ld returned 1 exit status
make[3]: *** [trqauthd] Error 1

libtool: link: gcc -Wall -pthread -g -D_LARGEFILE64_SOURCE -o .libs/trqauthd trq_auth_
daemon.o trq_main.o -ldl -lssl -lcrypto -L/home/adaptive/torques/torque-
4.0.0/src/lib/Libpbs/.libs /home/adaptive/torques/torque-
4.0.0/src/lib/Libpbs/.libs/libtorque.so -lpthread -lrt -pthread
/usr/bin/ld: cannot find -lcrypto
collect2: ld returned 1 exit status
make[3]: *** [trqauthd] Error 1
```

To resolve the compile issue, use these commands:

```
> cd /usr/lib
> ln -s /lib/libcrypto.so.0.9. libcrypto.so
> ln -s /lib/libssl.so.0.9.8 libssl.so
```

Why are there so many error messages in the client logs (trqauthd logs) when I don't notice client commands failing?

If a client makes a connection to the server and the trqauthd connection for that client command is authorized *before* the client's connection, the trqauthd connection is rejected. The connection is retried, but if all retry attempts are rejected, trqauthd logs a message indicating a failure. Some client commands then open a new connection to the server and try again. The client command fails only if all its retries fail.

Related Topics

Troubleshooting

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 Moab Parameters
in the *Moab Workload Manager Administrator Guide* for more information.

Additionally, Michael Jennings at LBNL has authored an open-source bash node
health check script project. It offers an easy way to perform some of the most
common node health checking tasks, such as verifying network and filesystem
functionality. More information is available on the [project's page](#).

For more information about node health checks, see these topics:

- [Configuring MOMs to Launch a Health Check](#)
- [Creating the Health Check Script](#)
- [Adjusting Node State Based on the Health Check Output](#)
- [Example Health Check Script](#)

Related Topics

[Troubleshooting](#)

Configuring MOMs to Launch a Health Check

The health check feature is configured via the `mom_priv/config` file using the
parameters described below:

Parameter	Format	Default	Description
$node_ check_ script	<STRING>	N/A	(Required) Specifies the fully qualified pathname of the health check script to run
$node_ check_inter- val	<INTEGER>	1	(Optional) Specifies the number of MOM intervals between health checks (by default, each MOM interval is 45 seconds long - this is controlled via the [$status_update_time](#) node parameter. The integer may be followed by a list of event names (`jobstart` and `jobend` are currently supported). See [pbs_mom](#) for more information. xref>. ⓘ The node health check may be configured to run before the prologue script by including the "jobstart" option. However, the job environment variables are not in the health check at that point.

Related Topics

[Compute Node Health Check](#)

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.

By default, the script looks for the **EVENT:** keyword to indicate successes. 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 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.

> 🛈 In order for the node health check script to log a positive run, it is necessary to include the keyword **EVENT:** at the beginning of the message your script returns. Failure to do so may result in unexpected outcomes.

> 🛈 Both the **ERROR** and **EVENT:** keywords are case insensitive.

Related Topics

Compute Node Health Check

Adjusting Node State Based on the Health Check Output

If the health check reports an error, the node attribute "message" is set to the error string returned. Cluster schedulers can be configured to adjust a given node's state based on this information. For example, by default, Moab sets a node's state to down if a node error message is detected. The node health script continues to run at the configured interval (see Configuring MOMs to Launch a Health Check for more information), and if it does not generate the error message again during one of its later executions, Moab picks that up at the beginning of its next iteration and restores the node to an online state.

Related Topics

Compute Node Health Check

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

Debugging

Torque supports a number of diagnostic and debug options including the following:

PBSDEBUG environment variable - If set to 'yes', this variable will prevent pbs_server, pbs_mom, and/or pbs_sched from backgrounding themselves allowing direct launch under a debugger. Also, some client commands will provide additional diagnostic information when this value is set.

PBSLOGLEVEL environment variable - Can be set to any value between 0 and 7 and specifies the logging verbosity level (default = 0)

PBSCOREDUMP environment variable - If set, it will cause the offending resource manager daemon to create a core file if a *SIGSEGV*, *SIGILL*, *SIGFPE*, *SIGSYS*, or *SIGTRAP* signal is received. The core dump will be placed in the daemon's home directory ($PBSHOME/mom_priv for pbs_mom and $PBSHOME/server_priv for pbs_server).

> To enable core dumping in a Red Hat-based system, you must add the following line to the /etc/init.d/pbs_mom and /etc/init.d/pbs_server scripts:
>
> export DAEMON_COREFILE_LIMIT=unlimited

NDEBUG #define - if set at build time, will cause additional low-level logging information to be output to stdout for pbs_server and pbs_mom daemons.

tracejob reporting tool - can be used to collect and report logging and accounting information for specific jobs (See Using "tracejob" to Locate Job Failures) for more information.

> ℹ️ *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

Error code name	Number	Description
PBSE_UNKSIG	15015	Unknown signal name
PBSE_BADATVAL	15016	Bad attribute value
PBSE_MODATRRUN	15017	Cannot modify attribute in run state
PBSE_BADSTATE	15018	Request invalid for job state
PBSE_UNKQUE	15020	Unknown queue name
PBSE_BADCRED	15021	Invalid credential in request
PBSE_EXPIRED	15022	Expired credential in request
PBSE_QUNOENB	15023	Queue not enabled
PBSE_QACESS	15024	No access permission for queue
PBSE_BADUSER	15025	Bad user - no password entry
PBSE_HOPCOUNT	15026	Max hop count exceeded
PBSE_QUEEXIST	15027	Queue already exists
PBSE_ATTRTYPE	15028	Incompatible queue attribute type
PBSE_QUEBUSY	15029	Queue busy (not empty)
PBSE_QUENBIG	15030	Queue name too long
PBSE_NOSUP	15031	Feature/function not supported
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

Error code name	Number	Description
PBSE_NOSERVER	15036	No server to connect to
PBSE_UNKRESC	15037	Unknown resource
PBSE_EXCQRESC	15038	Job exceeds queue resource limits
PBSE_QUENODFLT	15039	No default queue defined
PBSE_NORERUN	15040	Job not rerunnable
PBSE_ROUTEREJ	15041	Route rejected by all destinations
PBSE_ROUTEEXPD	15042	Time in route queue expired
PBSE_MOMREJECT	15043	Request to MOM failed
PBSE_BADSCRIPT	15044	(qsub) Cannot access script file
PBSE_STAGEIN	15045	Stage-In of files failed
PBSE_RESCUNAV	15046	Resources temporarily unavailable
PBSE_BADGRP	15047	Bad group specified
PBSE_MAXQUED	15048	Max number of jobs in queue
PBSE_CKPBSY	15049	Checkpoint busy, may be retries
PBSE_EXLIMIT	15050	Limit exceeds allowable
PBSE_BADACCT	15051	Bad account attribute value
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

Error code name	Number	Description
PBSE_BADDEPEND	15056	Invalid dependency
PBSE_DUPLIST	15057	Duplicate entry in list
PBSE_DISPROTO	15058	Bad DIS based request protocol
PBSE_EXECTHERE	15059	Cannot execute there
PBSE_SISREJECT	15060	Sister rejected
PBSE_SISCOMM	15061	Sister could not communicate
PBSE_SVRDOWN	15062	Requirement rejected -server shutting down
PBSE_CKPSHORT	15063	Not all tasks could checkpoint
PBSE_UNKNODE	15064	Named node is not in the list
PBSE_UNKNODEATR	15065	Node-attribute not recognized
PBSE_NONODES	15066	Server has no node list
PBSE_NODENBIG	15067	Node name is too big
PBSE_NODEEXIST	15068	Node name already exists
PBSE_BADNDATVAL	15069	Bad node-attribute value
PBSE_MUTUALEX	15070	State values are mutually exclusive
PBSE_GMODERR	15071	Error(s) during global modification of nodes
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

Error code name	Number	Description
PBSE_MAXUSERQUED	15076	Maximum number of jobs already in queue for user
PBSE_BADDISALLOWTYPE	15077	Bad type in "disallowed_types" list
PBSE_NOINTERACTIVE	15078	Interactive jobs not allowed in queue
PBSE_NOBATCH	15079	Batch jobs not allowed in queue
PBSE_NORERUNABLE	15080	Rerunable jobs not allowed in queue
PBSE_NONONRERUNABLE	15081	Non-rerunable jobs not allowed in queue
PBSE_UNKARRAYID	15082	Unknown array ID
PBSE_BAD_ARRAY_REQ	15083	Bad job array request
PBSE_TIMEOUT	15084	Time out
PBSE_JOBNOTFOUND	15085	Job not found
PBSE_NOFAULTTOLERANT	15086	Fault tolerant jobs not allowed in queue
PBSE_NOFAULTINTOLERANT	15087	Only fault tolerant jobs allowed in queue
PBSE_NOJOBARRAYS	15088	Job arrays not allowed in queue
PBSE_RELAYED_TO_MOM	15089	Request was relayed to a MOM
PBSE_MEM_MALLOC	15090	Failed to allocate memory for memmgr
PBSE_MUTEX	15091	Failed to allocate controlling mutex (lock/unlock)
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

Error code name	Number	Description
PBSE_SOCKET_WRITE	15096	Failed to write data to socket
PBSE_SOCKET_READ	15097	Failed to read data from socket
PBSE_SOCKET_CLOSE	15098	Socket closed
PBSE_SOCKET_LISTEN	15099	Failed to listen in on socket
PBSE_AUTH_INVALID	15100	Invalid auth type in request
PBSE_NOT_IMPLEMENTED	15101	Functionality not yet implemented
PBSE_QUENOTAVAILABLE	15102	Queue is not available

Related Topics

[Troubleshooting](#)

Appendices

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

Debugging

Commands Overview

Client Commands

Command	Description
momctl	Manage/diagnose MOM (node execution) daemon
pbsdsh	Launch tasks within a parallel job
pbsnodes	View/modify batch status of compute nodes
qalter	Modify queued batch jobs
qchkpt	Checkpoint batch jobs
qdel	Delete/cancel batch jobs
qgpumode	Specifies new mode for GPU
qgpureset	Reset the GPU
qhold	Hold batch jobs
qmgr	Manage policies and other batch configuration
qmove	Move batch jobs
qorder	Exchange order of two batch jobs in any queue
qrerun	Rerun a batch job
qrls	Release batch job holds
qrun	Start a batch job
qsig	Send a signal to a batch job

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)

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
Server Parameters

momctl

(PBS MOM Control)

Synopsis

```
momctl -c { <JOBID> | all }
momctl -C
momctl -d { <INTEGER> | <JOBID> }
momctl -f <FILE>
momctl -h <HOST>[,<HOST>]...
momctl -l
momctl -p <PORT_NUMBER>
momctl -q <ATTRIBUTE>
momctl -r { <FILE> | LOCAL:<FILE> }
momctl -s
```

Overview

The momctl command allows remote shutdown, reconfiguration, diagnostics, and querying of the pbs_mom daemon.

Format

-c — Clear	
Format	{ <JOBID> | *all* }
Default	---
Description	Makes the MOM unaware of the job's existence. It does not clean up any processes associated with the job.
Example	`momctl - node1 -c 15406`

-C — Cycle	
Format	---
Default	---
Description	Cycle pbs_mom(s).
Example	`momctl - node1 -C` Cycle pbs_mom on node1.

-d — Diagnose	
Format	{ <INTEGER> | <JOBID> }
Default	0
Description	Diagnose MOM(s). (For more details, see Diagnose detail 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.

-l — Layout

Format	---
Default	---
Description	Display the layout of the machine as it is understood by Torque.

-l — Layout

Example	
| **Example** | ```
[root@c04a numa]# momctl -l
Machine (50329748KB)
 Socket 0 (33552532KB)
  Chip 0 (16775316KB)
   Core 0 (1 threads)
   Core 1 (1 threads)
   Core 2 (1 threads)
   Core 3 (1 threads)
   Core 4 (1 threads)
   Core 5 (1 threads)
   Core 6 (1 threads)
   Core 7 (1 threads)
  Chip 1 (16777216KB)
   Core 8 (1 threads)
   Core 9 (1 threads)
   Core 10 (1 threads)
   Core 11 (1 threads)
   Core 12 (1 threads)
   Core 13 (1 threads)
   Core 14 (1 threads)
   Core 15 (1 threads)
 Socket 1 (16777216KB)
  Chip 2 (8388608KB)
   Core 16 (1 threads)
   Core 17 (1 threads)
   Core 18 (1 threads)
   Core 19 (1 threads)
   Core 20 (1 threads)
   Core 21 (1 threads)
   Core 22 (1 threads)
   Core 23 (1 threads)
  Chip 3 (8388608KB)
   Core 24 (1 threads)
   Core 25 (1 threads)
   Core 26 (1 threads)
   Core 27 (1 threads)
   Core 28 (1 threads)
   Core 29 (1 threads)
   Core 30 (1 threads)
   Core 31 (1 threads)
``` |

-p — Port

Field	Value
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
``` <br> Request diagnose information over port 5455 on node1. |

-q — Query

Format	<ATTRIBUTE>
Default	---
Description	Query <ATTRIBUTE> on specified MOM, where *<ATTRIBUTE>* is a property listed by **pbsnodes** −a (see Query attributes 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_config 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.config 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
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] [-F] [-h help] [-H hostname] [-L logfile] [-M
MOMport] [-R RMPport] [-p|-r] [-P purge] [-w] [-x]
```

Description

The `pbs_mom` command is located within the `TORQUE_HOME` directory and starts the operation of a batch Machine Oriented Mini-server (MOM) on the execution host. To ensure that the `pbs_mom` command is not runnable by the general user community, the server will only execute if its real and effective uid is zero.

The first function of `pbs_mom` is to place jobs into execution as directed by the server, establish resource usage limits, monitor the job's usage, and notify the server when the job completes. If they exist, `pbs_mom` will execute a prologue script before executing a job and an epilogue script after executing the job.

The second function of `pbs_mom` is to respond to resource monitor requests. This was done by a separate process in previous versions of PBS but has now been combined into one process. It provides information about the status of running jobs, memory available, etc.

The last function of `pbs_mom` is to respond to task manager requests. This involves communicating with running tasks over a TCP socket as well as communicating with other MOMs within a job (a.k.a. a "sisterhood").

`pbs_mom` will record a diagnostic message in a log file for any error occurrence. The log files are maintained in the mom_logs directory below the home directory of the server. If the log file cannot be opened, the diagnostic message is written to the system console.

Options

Flag	Name	Description
-a	alarm	Specifies 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	Specifies 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.

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.
-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.
-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`.
-F	fork	Do not fork. ⓘ This option is useful when running under systemd (Red Hat 7-based or SUSE 12-based systems).
-h	help	Displays the help/usage message.
-H	hostname	Sets the MOM's hostname. This can be useful on multi-homed networks.
-L	logfile	Specifies 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.

Flag	Name	Description
-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	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.
-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.
-w	wait_for_ server	When started with -w, pbs_moms wait until they get their MOM hierarchy file from pbs_server to send their first update, or until 10 minutes pass. This reduces network traffic on startup and can bring up clusters faster.
-x	n/a	Disables the check for privileged port resource monitor connections. This is used mainly for testing since the privileged port is the only mechanism used to prevent any ordinary user from connecting.

Configuration file

The configuration file, located at `mom_priv/config` by default, can be specified on the command line at program start with the **-c** flag. The use of this file is to provide several types of run time information to `pbs_mom`: static resource names and values, external resources provided by a program to be run on request via a shell escape, and values to pass to internal set up functions at initialization (and re-initialization).

See the Parameters page for a full list of `pbs_mom` parameters.

Each item type is on a single line with the component parts separated by white space. If the line starts with a hash mark (pound sign, #), the line is considered to be a comment and is skipped.

Static Resources

For static resource names and values, the configuration file contains a list of resource names/values pairs, one pair per line and separated by white space. An example of static resource names and values could be the number of tape drives of different types and could be specified by:

- tape3480 4

- tape3420 2

- tapedat 1

- tape8mm 1

Shell Commands

If the first character of the value is an exclamation mark (*!*), the entire rest of the line is saved to be executed through the services of the system(3) standard library routine.

The shell escape provides a means for the resource monitor to yield arbitrary information to the scheduler. Parameter substitution is done such that the value of any qualifier sent with the query, as explained below, replaces a token with a percent sign (%) followed by the name of the qualifier. For example, here is a configuration file line which gives a resource name of "escape":

```
escape !echo %xxx %yyy
```

If a query for "escape" is sent with no qualifiers, the command executed would be `echo %xxx %yyy`.

If one qualifier is sent, `escape[xxx=hi there]`, the command executed would be `echo hi there %yyy`.

If two qualifiers are sent, `escape[xxx=hi][yyy=there]`, the command executed would be `echo hi there`.

If a qualifier is sent with no matching token in the command line, `escape [zzz=snafu]`, an error is reported.

Resources

Resource Manager queries can be made with **momctl** `-q` options to retrieve and set pbs_mom options. Any configured static resource may be retrieved with a request of the same name. These are resource requests not otherwise documented in the PBS ERS.

Request	Description
cycle	Forces an immediate MOM cycle.
status_update_time	Retrieve or set the $status_update_time parameter.
check_poll_time	Retrieve or set the $check_poll_time parameter.
configversion	Retrieve the config version.
jobstartblocktime	Retrieve or set the $jobstartblocktime parameter.
enablemomrestart	Retrieve or set the $enablemomrestart parameter.
loglevel	Retrieve or set the $loglevel parameter.
down_on_error	Retrieve or set the $down_on_error parameter.
diag0 - diag4	Retrieves varied diagnostic information.
rcpcmd	Retrieve or set the $rcpcmd parameter.
version	Retrieves the pbs_mom version.

Health check

The health check script is executed directly by the pbs_mom daemon under the root user id. It must be accessible from the compute node and may be a script or compiled executable program. It may make any needed system calls and execute any combination of system utilities but should not execute resource manager client commands. Also, the pbs_mom daemon blocks until the health check is completed and does not possess a built-in timeout. Consequently, it is advisable to keep the launch script execution time short and verify that the script will not block even under failure conditions.

If the script detects a failure, it should return the ERROR keyword to stdout followed by an error message. The message (up to 1024 characters) immediately following the **ERROR** string will be assigned to the node attribute message of the associated node.

If the script detects a failure when run from "jobstart", then the job will be rejected. You can use this behavior with an advanced scheduler, such as Moab Workload Manager, to cause the job to be routed to another node. Torque currently ignores Error messages by default, but you can configure an advanced scheduler to react appropriately.

If the `$down_on_error` MOM setting is enabled, the MOM will set itself to state down and report to pbs_server. Additionally, the `$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_server's `down_on_error` features enabled. Also see down_on_error (in **Server Parameters**).

See Creating the Health Check Script on page 214 for more information.

Files

File	Description
$PBS_SERVER_HOME/server_name	Contains the hostname running pbs_server
$PBS_SERVER_HOME/mom_priv	The default directory for configuration files, typically (`/usr/spool/pbs`)/mom_priv
$PBS_SERVER_HOME/mom_logs	Directory for log files recorded by the server
$PBS_SERVER_HOME/mom_priv/-prologue	The administrative script to be run before job execution
$PBS_SERVER_HOME/mom_priv/e-pilogue	The administrative script to be run after job execution

Signal handling

pbs_mom handles the following signals:

Signal	Description
SIGHUP	Causes pbs_mom to re-read its configuration file, close and reopen the log file, and reinitialize resource structures.
SIGALRM	Results in a log file entry. The signal is used to limit the time taken by certain children processes, such as the prologue and epilogue.
SIGINT and SIGTERM	Results in pbs_mom exiting without terminating any running jobs. This is the action for the following signals as well: SIGXCPU, SIGXFSZ, SIGCPULIM, and SIGSHUTDN.
SIGUSR1, SIGUSR2	Causes the MOM to increase and decrease logging levels, respectively.

Signal	Description
SIGPIPE, SIGINFO	Are ignored.
SIGBUS, SIGFPE, SIGILL, SIGTRAP, and SIGSYS	Cause a core dump if the PBSCOREDUMP environmental variable is defined.

All other signals have their default behavior installed.

Exit status

If the pbs_mom command fails to begin operation, the server exits with a value greater than zero.

Related Topics

pbs_server(8B)

Non-Adaptive Computing topics

- pbs_scheduler_basl(8B)
- pbs_scheduler_tcl(8B)
- PBS External Reference Specification
- PBS Administrators Guide

pbs_server

(*PBS Server*) pbs batch system manager

Synopsis

```
pbs_server [-a active] [-A acctfile] [-c] [-d config_path] [-
F] [-f force overwrite] [-H hostname] [--ha] [-l location] [-L
logfile] [-n don't send hierarchy] [-p port] [-S scheduler_
port] [-t type] [-v] [--about] [--version]
```

Description

The pbs_server command starts the operation of a batch server on the local host. Typically, this command will be in a local boot file such as /etc/rc.local. If the batch server is already in execution, pbs_server will exit with an error. To ensure that the pbs_server command is not runnable by the general user community, the server will only execute if its real and effective uid is zero.

The server will record a diagnostic message in a log file for any error occurrence. The log files are maintained in the server_logs directory below the home directory of the server. If the log file cannot be opened, the diagnostic message is written to the system console.

As of Torque 4.0, the pbs_server is multi-threaded which leads to quicker response to client commands, is more robust, and allows for higher job throughput.

Options

Option	Name	Description
-a	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.
-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.
-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 server's configuration files, PBS_HOME. A host may have multiple servers. Each server must have a different configuration directory. The default configuration directory is given by the symbol `$PBS_SERVER_HOME` which is typically `var/spool/torque`.
-f	force overwrite	Forces an overwrite of the server database. This can be useful to bypass the `yes/no` prompt when running something like `pbs_server -t create` and can ease installation and configuration of Torque via scripts.
-F	fork	Do not fork. ⓘ This option is useful when running under systemd (Red Hat 7-based or SUSE 12-based systems).
--ha	high_availability	Starts server in high availability mode (for details, see Server High Availability).

Option	Name	Description
-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).
-l	location	Specifies where to find the scheduler (for example, Moab) when it does not reside on the same host as Torque. ```pbs_server -l <other_host>:<other_port>```
-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).
-n	no send	This directs `pbs_server` to not send the hierarchy to all the MOMs on startup. Instead, the hierarchy is only sent if a MOM requests it. This flag works only in conjunction with the local MOM hierarchy feature.
-p	port	Specifies the port number on which the server will listen for batch requests. If multiple servers are running on a single host, each must have its own unique port number. This option is for use in testing with multiple batch systems on a single host.
-S	scheduler_ port	Specifies the port number to which the server should connect when contacting the scheduler. The argument scheduler_conn is of the same syntax as under the -M option.
-t	type	If the job is rerunnable or restartable, and -t create is specified, the server will discard any existing configuration files, queues, and jobs, and initialize configuration files to the default values. The server is idled. ⓘ If -t is not specified, the job states will remain the same.

Files

File	Description
TORQUE_HOME/server_ priv	Default directory for configuration files, typically `/usr/spool/pbs/server_ priv`
TORQUE_HOME/server_ logs	Directory for log files recorded by the server

Signal handling

On receipt of the following signals, the server performs the defined action:

Action	Description
SIGHUP	The current server log and accounting log are closed and reopened. This allows for the prior log to be renamed and a new log started from the time of the signal.
SIGINT	Causes an orderly shutdown of pbs_server.
SIGUSR1, SIGURS2	Causes server to increase and decrease logging levels, respectively.
SIGTERM	Causes an orderly shutdown of pbs_server.
SIGSHUTDN	On systems (Unicos) where SIGSHUTDN is defined, it also causes an orderly shutdown of the server.
SIGPIPE	This signal is ignored.

All other signals have their default behavior installed.

Exit status

If the server command fails to begin batch operation, the server exits with a value greater than zero.

Related Topics

    pbs_mom(8B)

    pbsnodes(8B)

    qmgr(1B)

    qrun(8B)

    qsub(1B)

    qterm(8B)

Non-Adaptive Computing topics

- pbs_connect(3B)
- pbs_sched_basl(8B)
- pbs_sched_tcl(8B)
- qdisable(8B)
- qenable(8B)
- qstart(8B)

- qstop(8B)
- PBS External Reference Specification

pbs_track

Starts a new process and informs pbs_mom to start tracking it.

Synopsis

```
pbs_track -j <JOBID> [-b] <executable> [args]
```

Description

The pbs_track command tells a pbs_mom daemon to monitor the lifecycle and resource usage of the process that it launches using exec(). The pbs_mom is told about this new process via the Task Manager API, using tm_adopt(). The process must also be associated with a job that already exists on the pbs_mom.

By default, pbs_track will send its PID to Torque via tm_adopt(). It will then perform an exec(), causing <executable> to run with the supplied arguments. pbs_track will not return until the launched process has completed because it becomes the launched process.

This command can be considered related to the **pbsdsh** command which uses the tm_spawn() API call. The pbsdsh command asks a pbs_mom to launch and track a new process on behalf of a job. When it is not desirable or possible for the pbs_mom to spawn processes for a job, pbs_track can be used to allow an external entity to launch a process and include it as part of a job.

This command improves integration with Torque and SGI's MPT MPI implementation.

Options

Option	Description
-j <JOBID>	Job ID the new process should be associated with.
-b	Instead of having pbs_track send its PID to Torque, it will fork() first, send the child PID to Torque, and then execute from the forked child. This essentially "backgrounds" pbs_track so that it will return after the new process is launched.

Operands

The pbs_track command accepts a path to a program/executable
(<executable>) and, optionally, one or more arguments to pass to that
program.

Exit status

Because the pbs_track command becomes a new process (if used without **-b**),
its exit status will match that of the new process. If the **-b** option is used, the
exit status will be zero if no errors occurred before launching the new process.

If pbs_track fails, whether due to a bad argument or other error, the exit status
will be set to a non-zero value.

Related Topics

pbsdsh(1B)

Non-Adaptive Computing topics

- tm_spawn(3B)

pbsdsh

The pbsdsh command distributes tasks to nodes under pbs.

> ⓘ Some limitations exist in the way that pbsdsh can be used. Please note the
> following situations are not currently supported:
>
> - Running multiple instances of pbsdsh concurrently within a single job.
> - Using the -o and -s options concurrently; although requesting these
>   options together is permitted, only the output from the first node is
>   displayed rather than output from every node in the chain.

Synopsis

```
pbsdsh [-c copies] [-o] [-s] [-u] [-v] program [args]
pbsdsh [-n node] [-o] [-s] [-u] [-v] program [args]
pbsdsh [-h nodename] [-o] [-v] program [args]
```

Description

Executes (spawns) a normal Unix program on one or more nodes under control
of the Portable Batch System, PBS. Pbsdsh uses the Task Manager API (see
tm_spawn(3)) to distribute the program on the allocated nodes.

When run without the **-c** or the **-n** option, pbsdsh will spawn the program on all nodes allocated to the PBS job. The spawns take place concurrently – all execute at (about) the same time.

Users will find the PBS_TASKNUM, PBS_NODENUM, and the PBS_VNODENUM environmental variables useful. They contain the TM task id, the node identifier, and the cpu (virtual node) identifier.

> ⓘ Note that under particularly high workloads, the `pbsdsh` command may not function properly.

Options

Option	Name	Description
-c	copies	The program is spawned on the first Copies nodes allocated. This option is mutually exclusive with **-n**.
-h	hostname	The program is spawned on the node specified.
-n	node	The program is spawned on one node which is the n-th node allocated. This option is mutually 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 -m <running|standby|suspend|hibernate|shutdown> <host
list>
pbsnodes [-{c|d|o|r}] [-q] [-s server] [-n -l] [-N "note"] [-A
"append 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.

NUMA-Awareness

When Torque is configured with NUMA-awareness and configured with --enable-groups, the number of total *and* the number of available sockets, numachips (numa nodes), cores, and threads are returned when the status of nodes are queried by Moab (a call is made to pbsnodes).

See <u>pbsnodes with NUMA-Awareness on page 195</u> for additional information and examples.

Options

Option	Description
-A	Append a note attribute to existing note attributes. The -N note option will overwrite exiting note attributes. -A will append a new note attribute to the existing note attributes delimited by a ',' and a space.
-a	All attributes of a node or all nodes are listed. This is the default if no flag is given.
-c	Clear OFFLINE from listed nodes.
-d	Print MOM diagnosis on the listed nodes. Not yet implemented. Use **momctl** instead.
-m	Set the hosts in the specified host list to the requested power state. If a compute node does not support the energy-saving power state you request, the command returns an error and leaves the state unchanged. In order for the command to wake a node from a low-power state, Wake-on-LAN (WOL) must be enabled for the node. ⓘ In order for the command to wake a node from a low-power state, Wake-on-LAN must be enabled for the node and it must support the g WOL packet. For more information, see <u>Changing Node Power States</u>. The allowable power states are: • **Running**: The node is up and running. • **Standby**: CPU is halted but still powered. Moderate power savings but low latency entering and leaving this state. • **Suspend**: Also known as Suspend-to-RAM. Machine state is saved to RAM. RAM is put into self-refresh mode. Much more significant power savings with longer latency entering and leaving state. • **Hibernate**: Also known as Suspend-to-disk. Machine state is saved to disk and then powered down. Significant power savings but very long latency entering and leaving state. • **Shutdown**: Equivalent to shutdown now command as root. The host list is a space-delimited list of node host names. See <u>Examples on page 246</u>.
-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.

Option	Description
-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.
-x	Same as [-A](#), but the output has an XML-like format.

Examples

Example A-3: host list

```
pbsnodes -m shutdown node01 node02 node03 node04
```

> *With this command, pbs_server tells the pbs_mom associated with nodes01-04 to shut down the node.*

The pbsnodes output shows the current power state of nodes. In this example, note that pbsnodes returns the MAC addresses of the nodes.

```
pbsnodes
nuc1
     state = free
     power_state = Running
     np = 4
     ntype = cluster
     status = rectime=1395765676,macaddr=0b:25:22:92:7b:26
,cpuclock=Fixed,varattr=,jobs=,state=free,netload=1242652020,gres=,loadave=0.16,ncpus=
6,physmem=16435852kb,availmem=24709056kb,totmem=33211016kb,idletime=4636,nusers=3,nses
sions=12,sessions=2758 998 1469 2708 2797 2845 2881 2946 4087 4154 4373
6385,uname=Linux bdaw 3.2.0-60-generic #91-Ubuntu SMP Wed Feb 19 03:54:44 UTC 2014
x86_64,opsys=linux
     note = This is a node note
     mom_service_port = 15002
     mom_manager_port = 15003

nuc2
     state = free
     power_state = Running
     np = 4
     ntype = cluster
     status = rectime=1395765678,macaddr=2c:a8:6b:f4:b9:35
,cpuclock=OnDemand:800MHz,varattr=,jobs=,state=free,netload=12082362,gres=,loadave=0.0
0,ncpus=4,physmem=16300576kb,availmem=17561808kb,totmem=17861144kb,idletime=67538,nuse
rs=2,nsessions=7,sessions=2189 2193 2194 2220 2222 2248 2351,uname=Linux nuc2 2.6.32-
431.el6.x86_64 #1 SMP Fri Nov 22 03:15:09 UTC 2013 x86_64,opsys=linux
     mom_service_port = 15002
     mom_manager_port = 15003
```

Related Topics

   pbs_server(8B)

Non-Adaptive Computing topics

   - PBS External Reference Specification

qalter

Alter batch job.

Synopsis

```
qalter [-a date_time][-A account_string][-c interval][-e path_
name]
[-h hold_list][-j join_list][-k keep_list][-l resource_list]
[-m mail_options][-M mail_list][-n][-N name][-o path_name]
[-p priority][-r y|n][-S path_name_list][-u user_list]
[-v variable_list][-W additional_attributes]
[-t array_range]
job_identifier ...
```

Description

The qalter command modifies the attributes of the job or jobs specified by `job_identifier` on the command line. Only those attributes listed as options on the command will be modified. If any of the specified attributes cannot be modified for a job for any reason, none of that job's attributes will be modified.

The qalter command accomplishes the modifications by sending a Modify Job batch request to the batch server which owns each job.

Options

Option	Name	Description
-a	date_time	Replaces the time at which the job becomes eligible for execution. The date_time argument syntax is: `[[[[CC]YY]MM]DD]hhmm[.SS]` If the month, MM, is not specified, it will default to the current month if the specified day DD, is in the future. Otherwise, the month will be set to next month. Likewise, if the day, DD, is not specified, it will default to today if the time hhmm is in the future. Otherwise, the day will be set to tomorrow. This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun.
-A	account_string	Replaces the account string associated with the job. This attribute cannot be altered once the job has begun execution.
-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. ℹ 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.

Option	Name	Description
-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.
-h	hold_list	Updates the types of holds on the job. The hold_list argument is a string of one or more of the following characters: • *u* – Add the USER type hold. • *s* – Add the SYSTEM type hold if the user has the appropriate level of privilege. (Typically reserved to the batch administrator.) • *o* – Add the OTHER (or OPERATOR) type hold if the user has the appropriate level of privilege. (Typically reserved to the batch administrator and batch operator.) • *n* – Set to none and clear the hold types which could be applied with the user's level of privilege. Repetition of characters is permitted, but "n" may not appear in the same option argument with the other three characters. This attribute can be altered once the job has begun execution, but the hold will not take effect unless the job is rerun.
-j	join	Declares which standard streams of the job will be merged together. The join argument value may be the characters "oe" or"eo", or the single character "n". An argument value of oe directs that the standard output and standard error streams of the job will be merged, intermixed, and returned as the standard output. An argument value of eo directs that the standard output and standard error streams of the job will be merged, intermixed, and returned as the standard error. A value of n directs that the two streams will be two separate files. This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun. > ℹ If using either the -e or the -o option and the −j eo|oe option, the −j option takes precedence and all standard error and output messages go to the chosen output file.

Option	Name	Description

-k	keep	Defines which if either of standard output or standard error of the job will be retained on the execution host. If set for a stream, this option overrides the path name for that stream.
		
		The argument is either the single letter "e", "o", or "n", or one or more of the letters "e" and "o" combined in either order.
		
		• *n* – No streams are to be retained.
		• *e* – The standard error stream is to retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by:
		`job_name.esequence`
		where job_name is the name specified for the job, and sequence is the sequence number component of the job identifier.
		• *o* – The standard output stream is to be retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by:
		`job_name.osequence`
		where job_name is the name specified for the job, and sequence is the sequence number component of the job identifier.
		• *eo* – Both the standard output and standard error streams will be retained.
		• *oe* – Both the standard output and standard error streams will be retained.
		
		This attribute cannot be altered once the job has begun execution.

-l	resource_list	Modifies the list of resources that are required by the job. The resource_list argument is in the following syntax: `resource_name[=[value]][,resource_name[=[value]],...]` For the complete list of resources that can be modified, see Requesting Resources. 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-exclusive	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, nonwhite 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 an integer between -1024 and +1023 inclusive. This attribute can be altered once the job has begun execution, but it will not take effect unless the job is rerun.
-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 delimited list. Examples: `-t 1-100` or `-t 1,10,50-100` If an array range isn't specified, the command tries to operate on the entire array. The command acts on the array (or specified range of the array) just as it would on an individual job. An optional "slot limit" can be specified to limit the amount of jobs that can run concurrently in the job array. The default value is unlimited. The slot limit must be the last thing specified in the array_request and is delimited from the array by a percent sign (%). ``` qalter 15.napali[] -t %20 ``` Here, the array 15.napali[] 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 -W additional_attributes.

-W additional_attributes

The following table lists the attributes PBS currently supports with the -W option.

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 according to type. If argument is a count, it must be greater than 0. If it is a job id and is not fully specified in the form: `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 user's home directory on the destination host.

Operands

The qalter command accepts one or more job_identifier operands of the form:

`sequence_number[.server_name][@server]`

Standard error

Any error condition, either in processing the options or the operands, or any error received in reply to the batch requests will result in an error message being written to standard error.

Exit status

Upon successful processing of all the operands presented to the qalter command, the exit status will be a value of zero.

If the qalter command fails to process any operand, the command exits with a value greater than zero.

Copyright

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

```
$ # request a checkpoint for job 3233
$ qchkpt 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>|-b <secs>|-m <message>|-p
<purge>|-t <array_range>|-W <delay>}]
<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.
-b	seconds	Defines the maximum number of seconds qdel will block attempting to contact pbs_server. If pbs_server is down, or for a variety of communication failures, `qdel` 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.
-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.
-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). The command acts on the array (or specified range of the array) just as it would on an individual job. ⓘ When deleting a range of jobs, you must include the subscript notation after the job ID (for example, "qdel -t 1-3 98432[]").
-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.
-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.

Operands

The `qdel` command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

or

```
all
```

Examples

```
# Delete a job array
$ qdel 1234[]

# Delete one job from an array
$ qdel 1234[1]

# Delete all jobs, including job arrays
$ qdel all

# Delete selected jobs from an array
$ qdel -t 2-4,6,8-10 64[]
```

> ⓘ There is not an option that allows you to delete all job arrays without deleting jobs.

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

⚠ This command is deprecated, use the nvidia-smi utility instead. See
https://developer.nvidia.com/nvidia-system-management-interface and
http://developer.download.nvidia.com/compute/cuda/6_
0/rel/gdk/nvidia-smi.331.38.pdf for more information.

(GPU mode)

Synopsis

qgpumode -H host -g gpuid -m mode

Description

The qgpumode command specifies the mode for the GPU. This command
triggers an immediate update of the pbs_server.

ⓘ For additional information about options for configuring GPUs, see NVIDIA
GPUs in the Moab Workload Manager Administrator Guide.

Options

Option	Description
-H	Specifies the host where the GPU is located.
-g	Specifies the ID of the GPU. This varies depending on the version of the Nvidia driver used. For driver 260.x, it is 0, 1, and so on. For driver 270.x, it is the PCI bus address, i.e., 0:5:0.

Option	Description

-m	Specifies the new mode for the GPU:
	
	• **0 (Default/Shared)**: Default/shared compute mode. Multiple threads can use `cudaSetDevice()` with this device.
	• **1 (Exclusive Thread)**: Compute-exclusive-thread mode. Only one thread in one process is able to use `cudaSetDevice()` with this device.
	• **2 (Prohibited)**: Compute-prohibited mode. No threads can use `cudaSetDevice()` with this device.
	• **3 (Exclusive Process)**: Compute-exclusive-process mode. Many threads in one process are able to use `cudaSetDevice()` with this device.
	
	```qgpumode -H node01 -g 0 -m 1```
	
	This puts the first GPU on node01 into mode 1 (exclusive)
	
	```qgpumode -H node01 -g 0 -m 0```
	
	This puts the first GPU on node01 into mode 0 (shared)

Related Topics

[qgpureset](#)

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.

Option	Description
-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

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).

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 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 an object.

Commands are:

Command	Description
active	Sets the active objects. If the active objects are specified, and the name is not given in a qmgr cmd the active object names will be used.
create	Is to create a new object, applies to queues and nodes.
delete	Is to destroy an existing object, applies to queues and nodes.
set	Is to define or alter attribute values of the object.
unset	Is to clear the value of attributes of the object. 🛈 This form does not accept an OP and value, only the attribute name.
list	Is to list the current attributes and associated values of the object.
print	Is to print all the queue and server attributes in a format that will be usable as input to the qmgr command.
names	Is a list of one or more names of specific objects The name list is in the form: `[name][@server][,queue_name[@server]...]` with no intervening white space. The name of an object is declared when the object is first created. If the name is @server, then all the objects of specified type at the server will be affected.
attr	Specifies the name of an attribute of the object which is to be set or modified. If the attribute is one which consist of a set of resources, then the attribute is specified in the form: `attribute_name.resource_name`

Command	Description
OP	Operation to be performed with the attribute and its value: • "=" – set the value of the attribute. If the attribute has an existing value, the current value is replaced with the new value. • "+=" – increase the current value of the attribute by the amount in the new value. • "-=" – decrease the current value of the attribute by the amount in the new value.
value	The value to assign to an attribute. If the value includes white space, commas or other special characters, such as the "#" character, the value string must be enclosed in quote marks (").

The following are examples of qmgr directives:

```
create queue fast priority=10,queue_type=e,enabled = true,max_running=0
set queue fast max_running +=2
create queue little
set queue little resources_max.mem=8mw,resources_max.cput=10
unset queue fast max_running
set node state = "down,offline"
active server s1,s2,s3
list queue @server1
set queue max_running = 10        - uses active queues
```

Exit status

Upon successful processing of all the operands presented to the qmgr command, the exit status will be a value of zero.

If the qmgr command fails to process any operand, the command exits with a value greater than zero.

Related Topics

pbs_server(8B)

Non-Adaptive Computing topics

- pbs_queue_attributes (7B)
- pbs_server_attributes (7B)
- qstart (8B), qstop (8B)
- qenable (8B), qdisable (8)
- PBS External Reference Specification

qmove

Move PBS batch jobs.

Synopsis

```
qmove destination jobId [jobId ...]
```

Description

To move a job is to remove the job from the queue in which it resides and instantiate the job in another queue. The qmove command issues a Move Job batch request to the batch server that currently owns each job specified by *jobId*.

A job in the **Running**, **Transiting**, or **Exiting** state cannot be moved.

Operands

The first operand, the new *destination*, is one of the following:

```
queue
```

```
@server
```

```
queue@server
```

If the *destination* operand describes only a queue, then qmove will move jobs into the queue of the specified name at the job's current server. If the *destination* operand describes only a batch server, then qmove will move jobs into the default queue at that batch server. If the *destination* operand describes both a queue and a batch server, then qmove will move the jobs into the specified queue at the specified server.

All following operands are *jobId*s which specify the jobs to be moved to the new *destination*. The qmove command accepts one or more *jobId* operands of the form: `sequenceNumber[.serverName][@server]`

Standard error

The qmove command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qmove command, the exit status will be a value of zero.

If the qmove command fails to process any operand, the command exits with a value greater than zero.

Related Topics

qsub

Related Topics(non-Adaptive Computing topics)

- pbs_movejob(3B)

qorder

Exchange order of two PBS batch jobs in any queue.

Synopsis

`qorder job1_identifier job2_identifier`

Description

To order two jobs is to exchange the jobs' positions in the queue(s) in which the jobs reside. The two jobs must be located on the same server. No attribute of the job, such as priority, is changed. The impact of changing the order in the queue(s) is dependent on local job schedule policy. For information about your local job schedule policy, contact your systems administrator.

> ℹ A job in the **running** state cannot be reordered.

Operands

Both operands are `job_identifier`s that specify the jobs to be exchanged. The `qorder` command accepts two `job_identifier` operands of the following form:
`sequence_number[.server_name][@server]`

The two jobs must be in the same location, so the server specification for the two jobs must agree.

Standard error

The `qorder` command will write diagnostic messages to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the `qorder` command, the exit status will be a value of zero.

If the `qorder` command fails to process any operand, the command exits with a value greater than zero.

Related Topics

qsub

qmove

Related Topics(non-Adaptive Computing topics)

- pbs_orderjob(3B)
- pbs_movejob(3B)

qrerun

(Rerun a batch job)

Synopsis

```
qrerun [{-f}] <JOBID>[ <JOBID>] ...
```

Description

The qrerun command directs that the specified jobs are to be rerun if possible. To rerun a job is to terminate the session leader of the job and return the job to the queued state in the execution queue in which the job currently resides.

If a job is marked as not rerunable then the rerun request will fail for that job. If the mini-server running the job is down, or it rejects the request, the Rerun Job batch request will return a failure unless **-f** is used.

Using **-f** violates IEEE Batch Processing Services Standard and should be handled with great care. It should only be used under exceptional circumstances. The best practice is to fix the problem mini-server host and let qrerun run normally. The nodes may need manual cleaning (see the -r option on the **qsub** and **qalter** commands).

Options

Option	Description
-f	Force a rerun on a job

```
qrerun -f 15406
```

> The qrerun all command is meant to be run if all of the compute nodes go down. If the machines have actually crashed, then we know that all of the jobs need to be restarted. The behavior if you don't run this would depend on how you bring up the pbs_mom daemons, but by default would be to cancel all of the jobs.
>
> Running the command makes it so that all jobs are requeued without attempting to contact the moms on which they should be running.

Operands

The qrerun command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Standard error

The qrerun command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qrerun command, the exit status will be a value of zero.

If the qrerun command fails to process any operand, the command exits with a value greater than zero.

Examples

```
> qrerun 3233
```

(Job 3233 will be re-run.)

Related Topics

qsub(1B)

qalter(1B)

Non-Adaptive Computing topics

- pbs_alterjob(3B)
- pbs_rerunjob(3B)

qrls

(Release hold on PBS batch jobs)

Synopsis

```
qrls [{-h <HOLD LIST>|-t <array_range>}] <JOBID>[ <JOBID>] ...
```

Description

The qrls command removes or releases holds which exist on batch jobs.

A job may have one or more types of holds which make the job ineligible for execution. The types of holds are USER, OTHER, and SYSTEM. The different types of holds may require that the user issuing the qrls command have special privileges. A user may always remove a USER hold on their own jobs, but only privileged users can remove OTHER or SYSTEM holds. An attempt to release a hold for which the user does not have the correct privilege is an error and no holds will be released for that job.

If no **-h** option is specified, the USER hold will be released.

If the job has no execution_time pending, the job will change to the queued state. If an execution_time is still pending, the job will change to the waiting state.

> ℹ If you run qrls on an array sub-job, pbs_server will correct the slot limit holds for the array to which it belongs.

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

qsub(1B)
qalter(1B)
qhold(1B)

Non-Adaptive Computing topics)

- pbs_alterjob(3B)
- pbs_holdjob(3B)
- pbs_rlsjob(3B)

qrun

(Run a batch job)

Synopsis

qrun [{-H <HOST>|-a}] <JOBID>[<JOBID>] ...

Overview

The qrun command runs a job.

Format

-H	
Format	<STRING> Host Identifier
Default	---
Description	Specifies the host within the cluster on which the job(s) are to be run. The host argument is the name of a host that is a member of the cluster of hosts managed by the server. If the option is not specified, the server will select the "worst possible" host on which to execute the job.
Example	```qrun -H hostname 15406```

-a	
Format	---
Default	---
Description	Run the job(s) asynchronously.
Example	```qrun -a 15406```

Command details

The qrun command is used to force a batch server to initiate the execution of a batch job. The job is run regardless of scheduling position or resource requirements.

In order to execute qrun, the user must have PBS Operation or Manager privileges.

Examples

```
> qrun 3233
```

(Run job 3233.)

qsig

(Signal a job)

Synopsis

```
qsig [{-s <SIGNAL>}] <JOBID>[ <JOBID>] ...
[-a]
```

Description

The qsig command requests that a signal be sent to executing batch jobs. The signal is sent to the session leader of the job. If the **-s** option is not specified, SIGTERM is sent. The request to signal a batch job will be rejected if:

- The user is not authorized to signal the job.
- The job is not in the running state.
- The requested signal is not supported by the system upon which the job is executing.

The qsig command sends a Signal Job batch request to the server which owns the job.

Options

Option	Name	Description
-s	signal	Declares which signal is sent to the job. The signal argument is either a signal name, e.g. SIGKILL, the signal name without the SIG prefix, e.g. KILL, or an unsigned signal number, e.g. 9. The signal name SIGNULL is allowed; the server will send the signal 0 to the job which will have no effect on the job, but will cause an obituary to be sent if the job is no longer executing. Not all signal names will be recognized by qsig. If it doesn't recognize the signal name, try issuing the signal number instead. Two special signal names, "suspend" and "resume", are used to suspend and resume jobs. Cray systems use the Cray-specific suspend()/resume() calls. On non-Cray system, suspend causes a SIGTSTP to be sent to all processes in the job's top task, wait 5 seconds, and then send a SIGSTOP to all processes in all tasks on all nodes in the job. This differs from Torque 2.0.0 which did not have the ability to propagate signals to sister nodes. Resume sends a SIGCONT to all processes in all tasks on all nodes. When suspended, a job continues to occupy system resources but is not executing and is not charged for walltime. The job will be listed in the "S" state. Manager or operator privilege is required to suspend or resume a job. ℹ Interactive jobs may not resume properly because the top-level shell will background the suspended child process.
-a	asynchronously	Makes the command run asynchronously.

Operands

The qsig command accepts one or more job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

Examples

```
> qsig -s SIGKILL 3233    send a SIGKILL to job 3233
> qsig -s KILL 3233       send a SIGKILL to job 3233
> qsig -s 9 3233          send a SIGKILL to job 3233
```

Standard error

The qsig command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qsig command, the exit status will be a value of zero.

If the qsig command fails to process any operand, the command exits with a value greater than zero.

Related Topics

   qsub(1B)

Non-Adaptive Computing topics

- pbs_sigjob(3B)
- pbs_resources_*(7B) where * is system type
- PBS ERS

qstat

Show status of PBS batch jobs.

Synopsis

```
qstat [-c] [-C] [-f [-1]] [-W site_specific] [job_
identifier... | destination...] [time]
qstat [-a|-i|-r|-e] [-c] [-n [-1]] [-s] [-G|-M] [-R] [-u user_
list]
[job_identifier... | destination...]
qstat -Q [-f [-1]] [-c] [-W site_specific] [destination...]
qstat -q [-c] [-G|-M] [destination...]
qstat -B [-c] [-f [-1]][-W site_specific] [server_name...]
qstat -t [-c] [-C]
```

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
-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.
-a	All jobs are displayed in the alternative format (see Standard output). 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.
-B	Specifies that the request is for batch server status and that the operands are the names of servers.
-c	Completed jobs are not displayed in the output. If desired, you can set the PBS_QSTAT_NO_COMPLETE environment variable to cause all qstat requests to not show completed jobs by default.
-C	Specifies that Torque will provide only a condensed output (job name, resources used, queue, state, and job owner) for jobs that have not changed recently. See job_full_report_time on page 318. Jobs that have recently changed will continue to send a full output.
-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.
-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
-G	Show size information in giga-bytes.
-i	Job status is displayed in the alternative format. For a destination id operand, statuses for jobs at that destination which are not running are displayed. This includes jobs which are queued, held or waiting. If an operand is a job id, status for that job is displayed regardless of its state.
-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.
-M	Show size information, disk or memory in mega-words. A word is considered to be 8 bytes.
-n	In addition to the basic information, nodes allocated to a job are listed.
-q	Specifies that the request is for queue status which should be shown in the alternative format.
-Q	Specifies that the request is for queue status and that the operands are destination identifiers.
-r	If an operand is a job id, status for that job is displayed. For a destination id operand, statuses for jobs at that destination which are running are displayed; this includes jobs which are suspended. Note that if there is no walltime given for a job, then elapsed time does not display.
-R	In addition to other information, disk reservation information is shown. Not applicable to all systems.
-s	In addition to the basic information, any comment provided by the batch administrator or scheduler is shown.
-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, statuses for jobs at that destination which are owned by the user(s) listed in user_list are displayed. The syntax of the user_list is: `user_name[@host][,user_name[@host],...]` Host names may be wild carded on the left end, e.g. "*.nasa.gov". User_name without a "@host" is equivalent to "user_name@*", that is at any host.

Operands

If neither the **-q** nor the **-B** option is given, the operands on the qstat command must be either job identifiers or destinations identifiers.

If the operand is a job identifier, it must be in the following form:

`sequence_number[.server_name][@server]`

where *sequence_number.server_name* is the job identifier assigned at submittal time (see **qsub**). If the *.server_name* is omitted, the name of the default server will be used. If *@server* is supplied, the request will be for the job identifier currently at that Server.

If the operand is a destination identifier, it is one of the following three forms:

- queue
- @server
- queue@server

If queue is specified, the request is for status of all jobs in that queue at the default server. If the @server form is given, the request is for status of all jobs at that server. If a full destination identifier, queue@server, is given, the request is for status of all jobs in the named queue at the named server.

If the **-Q** option is given, the operands are destination identifiers as specified above. If queue is specified, the status of that queue at the default server will be given. If queue@server is specified, the status of the named queue at the named server will be given. If @server is specified, the status of all queues at the named server will be given. If no destination is specified, the status of all queues at the default server will be given.

If the **-B** option is given, the operand is the name of a server.

Standard output

Displaying job status

If job status is being displayed in the default format and the **-f** option is not specified, the following items are displayed on a single line, in the specified

order, separated by white space:

- the job identifier assigned by PBS.
- the job name given by the submitter.
- the job owner.
- the CPU time used.
- the job state:

Item	Description
C	Job is completed after having run.
E	Job is exiting after having run.
H	Job is held.
Q	Job is queued, eligible to run or routed.
R	Job is running.
T	Job is being moved to new location.
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 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)

When any of the above options or the **-r** option is used to request an alternative display format, a column with the requested memory for the job is displayed. If more than one type of memory is requested for the job, either through server or queue parameters or command line, only one value can be displayed. The value displayed depends on the order the memory types are evaluated with the last type evaluated being the value displayed. The order of evaluation is dmem, mem, pmem, pvmem, vmem.

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 consists of the header line:

```
Server: server name
```

Followed by one line per server attribute of the form:

```
attribute_name = value
```

Standard error

The qstat command will write a diagnostic message to standard error for each error occurrence.

Configuration

If qstat is compiled with an option to include a Tcl interpreter, using the **-f** flag to get a full display causes a check to be made for a script file to use to output the requested information. The first location checked is `$HOME/.qstatrc`. If this does not exist, the next location checked is administrator configured. If one of these is found, a Tcl interpreter is started and the script file is passed to it along with three global variables. The command line arguments are split into two variable named flags and operands . The status information is passed in a variable named objects . All of these variables are Tcl lists. The flags list contains the name of the command (usually "qstat") as its first element. Any other elements are command line option flags with any options they use, presented in the order given on the command line. They are broken up individually so that if two flags are given together on the command line, they are separated in the list. For example, if the user typed:

```
qstat -QfWbigdisplay
```

the flags list would contain

```
qstat -Q -f -W bigdisplay
```

The operands list contains all other command line arguments following the flags. There will always be at least one element in operands because if no operands are typed by the user, the default destination or server name is used. The objects list contains all the information retrieved from the server(s) so the Tcl interpreter can run once to format the entire output. This list has the same

number of elements as the operands list. Each element is another list with two elements.

The first element is a string giving the type of objects to be found in the second. The string can take the values "server", "queue", "job" or "error".

The second element will be a list in which each element is a single batch status object of the type given by the string discussed above. In the case of "error", the list will be empty. Each object is again a list. The first element is the name of the object. The second is a list of attributes.

The third element will be the object text.

All three of these object elements correspond with fields in the structure batch_ status which is described in detail for each type of object by the man pages for pbs_statjob(3), pbs_statque(3), and pbs_statserver(3). Each attribute in the second element list whose elements correspond with the attrl structure. Each will be a list with two elements. The first will be the attribute name and the second will be the attribute value.

Exit status

Upon successful processing of all the operands presented to the qstat command, the exit status will be a value of zero.

If the qstat command fails to process any operand, the command exits with a value greater than zero.

Related Topics

> qalter(1B)
> qsub(1B)

Non-Adaptive Computing topics

- pbs_alterjob(3B)
- pbs_statjob(3B)
- pbs_statque(3B)
- pbs_statserver(3B)
- pbs_submit(3B)
- pbs_job_attributes(7B)
- pbs_queue_attributes(7B)
- pbs_server_attributes(7B)
- qmgr query_other_jobs parameter (allow non-admin users to see other users' jobs)
- pbs_resources_*(7B) where * is system type
- PBS ERS

qsub

Submit PBS job.

Synopsis

```
qsub [-a date_time][-A account_string][-b secs][-c checkpoint_
options][-C directive_prefix][-d path][-D path][-e path][-f][-
F][-h][-i idle_slot_limit][-I][-j join][-k keep][-K kill_
delay][-l resource_list][-L NUMA resource_list][-m mail_
options][-M user_list][-n node_exclusive][-N name][-o path][-p
priority][-P user[:group]][-q destination] [-r][-S path_to_
shell(s)][-t array_request][-u user_list][-v variable_list][-
V][-w path][-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). 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.

Variable	Description
PBS_O_WORKDIR	The absolute path of the current working directory of the qsub command.
PBS_ARRAYID	Each member of a job array is assigned a unique identifier (see -t option).
PBS_ENVIRONMENT	Set to PBS_BATCH to indicate the job is a batch job, or to PBS_INTERACTIVE to indicate the job is a PBS interactive job (see -I option).
PBS_GPUFILE	The name of the file containing the list of assigned GPUs. For more information about how to set up Torque with GPUS, see Accelerators in the Moab Workload Manager *Administrator Guide*.
PBS_JOBID	The job identifier assigned to the job by the batch system. It can be used in the stdout and stderr paths. Torque replaces $PBS_JOBID with the job's jobid (for example, #PBS -o /tmp/$PBS_JOBID.output).
PBS_JOBNAME	The job name supplied by the user.
PBS_NODEFILE	The name of the file contains the list of nodes assigned to the job (for parallel and cluster systems).
PBS_QUEUE	The name of the queue from which the job is executed.

Options

Option	Name	Description
-a	date_time	Declares the time after which the job is eligible for execution.
		The date_time argument is in the form:
		`[[[[CC]YY]MM]DD]hhmm[.SS]`
		where *CC* is the first two digits of the year (the century), *YY* is the second two digits of the year, *MM* is the two digits for the month, *DD* is the day of the month, *hh* is the hour, *mm* is the minute, and the optional *SS* is the seconds.
		If the month (MM) is not specified, it will default to the current month if the specified day (DD) is in the future. Otherwise, the month will be set to next month. Likewise, if the day (DD) is not specified, it will default to today if the time (hhmm) is in the future. Otherwise, the day will be set to tomorrow.
		For example, if you submit a job at 11:15 am with a time of `-a 1110`, the job will be eligible to run at 11:10 am tomorrow.

Option	Name	Description
-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`).
-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.) 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.

Option	Name	Description
-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. > ⓘ When specifying a directory for the location you need to include a trailing slash. The argument will be interpreted as follows: • *path_name* – where *path_name* is not an absolute path name, then the qsub command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component. • *hostname:path_name* – where *path_name* is not an absolute path name, then the qsub command will not expand the path name relative to the current working directory of the command. On delivery of the standard error, the path name will be expanded relative to the user's home directory on the hostname system. • *path_name* – where *path_name* specifies an absolute path name, then the qsub will supply the name of the host on which it is executing for the hostname. • *hostname:path_name* – where *path_name* specifies an absolute path name, the path will be used as specified. If the −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.
-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.

Option	Name	Description
-F	---	Specifies the arguments that will be passed to the job script when the script is launched. The accepted syntax is: `qsub -F "myarg1 myarg2 myarg3=myarg3value" myscript2.sh` ⓘ Quotation marks are required. qsub will fail with an error message if the argument following −F is not a quoted value. The pbs_mom server will pass the quoted value as arguments to the job script when it launches the script.
-h	---	Specifies that a user hold be applied to the job at submission time.
-i	idle_slot_ limit	Sets an idle slot limit for the job array being submitted. If this parameter is set for a non-array job, it will be rejected. Additionally, if the user requests an idle slot limit that exceeds the server parameter's default, the job will be rejected. See also the idle_slot_ limit server parameter. `$ qsub -t 0-99 -i 10 script.sh` *The submitted array will only instantiate 10 idle jobs; instead of all 100 jobs 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 for additional information of interactive jobs.
-j	join	Declares if the standard error stream of the job will be merged with the standard output stream of the job. An option argument value of *oe* directs that the two streams will be merged, intermixed, as standard output. An option argument value of *eo* directs that the two streams will be merged, intermixed, as standard error. If the join argument is *n* or the option is not specified, the two streams will be two separate files. ⓘ If using either the -e or the -o option and the −j eo|oe option, the −j option takes precedence and all standard error and output messages go to the chosen output file.

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 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.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.
		
		• *n* – Neither stream is retained.

-K	kill_delay	When set on a job, overrides server and queue kill_delay settings. The kill_delay value is a positive integer. The default is 0. See kill_delay on page 321 for more information.

Option	Name	Description
-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. See Requesting Resources on page 73 for more information. 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. > ⓘ -l supports some Moab-only extensions. See Requesting Resources on page 73 for more information on native Torque resources. qsub -W x= is recommended instead (supports more options). See -W for more information. For information on specifying multiple types of resources for allocation, see Multi-Req Support in the *Moab Workload Manager Administrator Guide*.
-L	req_information	> ⓘ Available with Torque 6.0 and later. This uses a different syntax than the -l resource_list option. Defines the NUMA-aware resource requests for NUMA hardware. This option will work with non-NUMA hardware. See -L NUMA Resource Request on page 185 for the syntax and valid values.
-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 "p", or one or more of the characters "a", "b", "e", and "f". 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. If the character "p" is specified, mail will never be sent for the job. For the characters "a", "b", "e" and "f": • *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. • *f* – Mail is sent when the job terminates with a non-zero exit code. If the `-m` option is not specified, mail will be sent if the job is aborted.

Option	Name	Description
-M	user_list	Declares the list of users to whom mail is sent by the execution server when it sends mail about the job. The user_list argument is of the form: `user[@host][,user[@host],...]` If unset, the list defaults to the submitting user at the qsub host, i.e. the job owner.

-n	node_exclusive	Allows a user to specify an exclusive-node access/allocation request for the job. This will set `node_exclusive = True` in the output of `qstat -f <job ID>`.
		
		ⓘ For Moab, the following options are equivalent to "-n":
		```> qsub -l naccesspolicy=singlejob jobscript.sh```
		```# OR```
		```> qsub -W x=naccesspolicy:singlejob jobscript.sh```
		
		By default, this only applies for cpusets, and only for compatible schedulers (see Linux Cpuset Support on page 118).
		
		For systems that use Moab *and* have cgroups enabled, the recommended manner for assigning all cores is to use NUMA syntax: "`-L tasks=<count>:lprocs=all:place=node`".
		
		With cgroups, the ("-l") syntax (lowercase L) will, by default, restrict to the number of cores requested, or to the `resources_default.procs` value (i.e., 1 core, typically). In order to override this behavior and have Moab assign all the cores on a node while using "`-l...singlejob`" and/or "`-n`" (in other words, without "`-L ...lprocs=all...`"), you must also set `RMCFG[<torque>] FLAGS=MigrateAllJobAttributes` in moab.cfg.

-N	name	Declares a name for the job. The name specified may be an unlimited number of characters in length. It must consist of printable, nonwhite space characters with the first character alphabetic. If the `-N` option is not specified, the job name will be the base name of the job script file specified on the command line. If no script file name was specified and the script was read from the standard input, then the job name will be set to STDIN.

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. ℹ️ When specifying a directory for the location you need to include a trailing slash. The argument will be interpreted as follows: • *path_name* – where *path_name* is not an absolute path name, then the qsub command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component. • *hostname:path_name* – where *path_name* is not an absolute path name, then the qsub command will not expand the path name relative to the current working directory of the command. On delivery of the standard output, the path name will be expanded relative to the user's home directory on the hostname system. • *path_name* – where *path_name* specifies an absolute path name, then the qsub will supply the name of the host on which it is executing for the hostname. • *hostname:path_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 and the PBS ERS section 2.7.4, "Default Server".) If the -q option is specified, it is in one of the following three forms: • queue • @server • queue@server If the destination argument names a queue and does not name a server, the job will be submitted to the named queue at the default server. If the destination argument names a server and does not name a queue, the job will be submitted to the default queue at the named server. If the destination argument names both a queue and a server, the job will be submitted to the named queue at the named server.
-r	y/n	Declares whether the job is rerunable (see the **qrerun** command). The option argument is a single character, either y or n. If the argument is "y", the job is rerunable. If the argument is "n", the job is not rerunable. The default value is y, rerunable.

-S	path_list	Declares the path to the desires shell for this job.
		`qsub script.sh -S /bin/tcsh`
		If the shell path is different on different compute nodes, use the following syntax:
		`path[@host][,path[@host],...]`
		`qsub script.sh -S /bin/tcsh@node1,/usr/bin/tcsh@node2`
		Only one path may be specified for any host named. Only one path may be specified without the corresponding host name. The path selected will be the one with the host name that matched the name of the execution host. If no matching host is found, then the path specified without a host will be selected, if present.
		
		If the −S option is not specified, the option argument is the null string, or no entry from the path_list is selected, the execution will use the user's login shell on the execution host.

Option	Name	Description
-t	array_request	Specifies the task ids of a job array. Single task arrays are allowed.
		The array_request argument is an integer id or a range of integers. Multiple ids or id ranges can be combined in a comma delimited list. Examples: `-t 1-100` or `-t 1,10,50-100`
		An optional *slot limit* can be specified to limit the amount of jobs that can run concurrently in the job array. The default value is unlimited. The slot limit must be the last thing specified in the array_request and is delimited from the array by a percent sign (*%*).
		```\nqsub script.sh -t 0-299%5\n```
		This sets the slot limit to 5. Only 5 jobs from this array can run at the same time.
		You can use **qalter** to modify slot limits on an array. The server parameter **max_slot_limit** can be used to set a global slot limit policy.
-u	user_list	Defines the user name under which the job is to run on the execution system.
		The user_list argument is of the form:
		`user[@host][,user[@host],...]`
		Only one user name may be given per specified host. Only one of the user specifications may be supplied without the corresponding host specification. That user name will used for execution on any host not named in the argument list. If unset, the user list defaults to the user who is running qsub.
-v	variable_list	Expands the list of environment variables that are exported to the job.
		In addition to the variables described in the "Description" section above, variable_list names environment variables from the qsub command environment which are made available to the job when it executes. The variable_list is a comma separated list of strings of the form `variable` or `variable=value`. These variables and their values are passed to the job. Note that `-v` has a higher precedence than `-V`, so identically named variables specified via `-v` will provide the final value for an environment variable in the job.
-V	---	Declares that all environment variables in the qsub commands environment are to be exported to the batch job.
-w	path	Defines the working directory path to be used for the job. If the -w option is not specified, the default working directory is the current directory. This option sets the environment variable PBS_O_WORKDIR.

Option	Name	Description

-W	additional_
	attributes
		ℹ Use "-W x=" as pass-through for scheduler-only job extensions. See Resource Manager Extensions in the *Moab Workload ManagerAdministrator Guide* for a list of scheduler-only job extensions.
		
		For legacy purposes, qsub -l will continue to support some scheduler-only job extensions. However, when in doubt, use "-W x=".
		
		The `-W` option allows for the specification of additional job attributes. The general syntax of `-W` is in the form:
		
		`-W attr_name=attr_value.`
		
		You can use multiple `-W` options with this syntax:
		
		`-W attr_name1=attr_value1 -W attr_name2=attr_value2.`
		
		ℹ 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.
		• *group_list=g_list* – Defines the group name under which the job is to run on the execution system. The g_list argument is of the form:
		
		  `group[@host][,group[@host],...]`
		
		  Only one group name may be given per specified host. Only one of the group specifications may be supplied without the corresponding host specification. That group name will used for execution on any host not named in the argument list. If not set, the group_list defaults to the primary group of the user under which the job will be run.
		• *interactive=true* – If the interactive attribute is specified, the job is an interactive job. The -I option is an alternative method of specifying this attribute.
		• *job_radix=<int>* – To be used with parallel jobs. It directs the Mother Superior of the job to create a distribution radix of size *<int>* between sisters. See Managing Multi-Node Jobs.
		• *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

Option	Name	Description

		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) call on the execution system in the follow manner:
		   - For stagein: `rcp hostname:remote_file local_file` - For stageout: `rcp local_file hostname:remote_file` Data staging examples:
		`-W stagein=/tmp/input.txt@headnode:/home/user/input.txt`
		`-W stageout=/tmp/output.txt@headnode:/home/user/output.txt`
		
		If Torque has been compiled with wordexp support, then variables can be used in the specified paths. Currently only `$PBS_JOBID`, `$HOME`, and `$TMPDIR` are supported for stagein.
		    umask=XXX – Sets umask used to create stdout and stderr spool files in pbs_mom spool directory. Values starting with 0 are treated as octal values, otherwise the value is treated as a decimal umask value.

-x	---	By default, if you submit an interactive job with a script, the script will be parsed for PBS directives but the rest of the script will be ignored since it's an interactive job. The `-x` option allows the script to be executed in the interactive job and then the job completes. For example:
		`script.sh`
		`#!/bin/bash`
		`ls`
		`---end script---`
		`qsub -I script.sh`
		`qsub: waiting for job 5.napali to start`
		`dbeer@napali:#`
		`<displays the contents of the directory, because of the ls command>`
		`qsub: job 5.napali completed`

-X	---	Enables X11 forwarding. The DISPLAY environment variable must be set.
-z	---	Directs that the qsub command is not to write the job identifier assigned to the job to the commands standard output.

depend=dependency_list valid dependencies

> **ℹ** For job dependencies to work correctly, you must set the keep_completed server parameter.

Dependency	Description
synccount:count	This job is the first in a set of jobs to be executed at the same time. Count is the number of additional jobs in the set.
syncwith:jobid	This job is an additional member of a set of jobs to be executed at the same time. In the above and following dependency types, jobid is the job identifier of the first job in the set.
after:jobid[:jobid...]	This job may be scheduled for execution at any point after jobs jobid have started execution.
afterok:jobid[:jobid...]	This job may be scheduled for execution only after jobs jobid have terminated with no errors. See the csh warning under Extended description.
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 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.
beforenotok:jobid[:jobid...]	If this job terminates execution with errors, then jobs jobid... may begin. See the csh warning under Extended description.

Dependency	Description
`beforeany:jobid[:jobid...]`	When this job terminates execution, jobs jobid… may begin.
	If any of the before forms are used, the jobs referenced by jobid must have been submitted with a dependency type of on.
	If any of the before forms are used, the jobs referenced by jobid must have the same owner as the job being submitted. Otherwise, the dependency is ignored.

ℹ️ 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.

Dependency	Description
`afterstartarray:arrayid[count]`	After this many jobs have started from arrayid, this job may start.
`afterokarray:arrayid[count]`	This job may be scheduled for execution only after jobs in arrayid have terminated with no errors.
`afternotokarray:arrayid[count]`	This job may be scheduled for execution only after jobs in arrayid have terminated with errors.
`afteranyarray:arrayid[count]`	This job may be scheduled for execution after jobs in arrayid have terminated, with or without errors.
`beforestartarray:arrayid[count]`	Before this many jobs have started from arrayid, this job may start.
`beforeokarray:arrayid[count]`	If this job terminates execution without errors, then jobs in arrayid may begin.
`beforenotokarray:arrayid[count]`	If this job terminates execution with errors, then jobs in arrayid may begin.
`beforeanyarray:arrayid[count]`	When this job terminates execution, jobs in arrayid may begin.
	If any of the before forms are used, the jobs referenced by arrayid must have been submitted with a dependency type of on.
	If any of the before forms are used, the jobs referenced by arrayid must have the same owner as the job being submitted. Otherwise, the dependency is ignored.

Dependency	Description

> ⓘ Error processing of the existence, state, or condition of the job on which the newly submitted job is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the new job will be deleted by the server. Mail will be sent to the job submitter stating the error.

> ⓘ Jobs can depend on single job dependencies and array dependencies at the same time.

Dependency	Description
`afterok:jobid [:jobid...],afterokarray:arrayid [count]`	This job may be scheduled for execution only after jobs jobid and jobs in arrayid have terminated with no errors.

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.)

```
qsub script.sh -W depend=afterok:360976,afterokarray:360977[]
```

(Job 360976 and all jobs in array 360977 have to successfully finish 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. See <u>"torque.cfg" Configuration File on page 391</u> for more information on these parameters.

Extended description

Script Processing:

A job script may consist of PBS directives, comments and executable statements. A PBS directive provides a way of specifying job attributes in addition to the command line options. For example:

```
:
#PBS -N Job_name
#PBS -l walltime=10:30,mem=320kb
#PBS -m be
#
step1 arg1 arg2
step2 arg3 arg4
```

The qsub command scans the lines of the script file for directives. An initial line in the script that begins with the characters "#!" or the character ":" will be ignored and scanning will start with the next line. Scanning will continue until the first executable line, that is a line that is not blank, not a directive line, nor a line whose first nonwhite space character is "#". If directives occur on subsequent lines, they will be ignored.

A line in the script file will be processed as a directive to qsub if and only if the string of characters starting with the first nonwhite space character on the line and of the same length as the directive prefix matches the directive prefix.

The remainder of the directive line consists of the options to qsub in the same syntax as they appear on the command line. The option character is to be preceded with the "-" character.

If an option is present in both a directive and on the command line, that option and its argument, if any, will be ignored in the directive. The command line takes precedence.

If an option is present in a directive and not on the command line, that option and its argument, if any, will be processed as if it had occurred on the command line.

The directive prefix string will be determined in order of preference from:

- The value of the **-C** option argument if the option is specified on the command line.
- The value of the environment variable PBS_DPREFIX if it is defined.
- The four character string #PBS.

If the **-C** option is found in a directive in the script file, it will be ignored.

User Authorization:

When the user submits a job from a system other than the one on which the PBS Server is running, the name under which the job is to be executed is selected according to the rules listed under the **-u** option. The user submitting

the job must be authorized to run the job under the execution user name. This authorization is provided if:

- The host on which qsub is run is trusted by the execution host (see `/etc/hosts.equiv`).
- The execution user has an `.rhosts` file naming the submitting user on the submitting host.

C-Shell .logout File:

The following warning applies for users of the c-shell, csh. If the job is executed under the csh and a `.logout` file exists in the home directory in which the job executes, the exit status of the job is that of the `.logout` script, not the job script. This may impact any inter-job dependencies. To preserve the job exit status, either remove the `.logout` file or place the following line as the first line in the `.logout` file:

```
set EXITVAL = $status
```

and the following line as the last executable line in `.logout`:

```
exit $EXITVAL
```

Interactive Jobs:

If the **-I** option is specified on the command line or in a script directive, or if the "interactive" job attribute declared true via the **-w** option, `-W interactive=true`, either on the command line or in a script directive, the job is an interactive job. The script will be processed for directives, but will not be included with the job. When the job begins execution, all input to the job is from the terminal session in which qsub is running.

When an interactive job is submitted, the qsub command will not terminate when the job is submitted. qsub will remain running until the job terminates, is aborted, or the user interrupts qsub with an SIGINT (the control-C key). If qsub is interrupted prior to job start, it will query if the user wishes to exit. If the user response "yes", qsub exits and the job is aborted.

One the interactive job has started execution, input to and output from the job pass through qsub. Keyboard generated interrupts are passed to the job. Lines entered that begin with the tilde (~) character and contain special sequences are escaped by qsub. The recognized escape sequences are:

Sequence	Description
~.	qsub terminates execution. The batch job is also terminated.
~susp	Suspend the qsub program if running under the C shell. "susp" is the suspend character (usually CNTL-Z).

Sequence	Description
~asusp	Suspend the input half of qsub (terminal to job), but allow output to continue to be displayed. Only works under the C shell. "asusp" is the auxiliary suspend character, usually CNTL-Y.

Exit status

Upon successful processing, the qsub exit status will be a value of zero.

If the qsub command fails, the command exits with a value greater than zero.

Related Topics

qalter(1B)
qdel(1B)
qhold(1B)
qrls(1B)
qsig(1B)
qstat(1B)
pbs_server(8B)

Non-Adaptive Computing topics

- pbs_connect(3B)
- pbs_job_attributes(7B)
- pbs_queue_attributes(7B)
- pbs_resources_irix5(7B)
- pbs_resources_sp2(7B)
- pbs_resources_sunos4(7B)
- pbs_resources_unicos8(7B)
- pbs_server_attributes(7B)
- qselect(1B)
- qmove(1B)
- qmsg(1B)
- qrerun(1B)

qterm

Terminate processing by a PBS batch server.

Synopsis

```
qterm [-t type] [server...]
```

Description

The qterm command terminates a batch server. When a server receives a terminate command, the server will go into the "Terminating" state. No new jobs will be allowed to be started into execution or enqueued into the server. The impact on jobs currently being run by the server depends

In order to execute qterm, the user must have PBS Operation or Manager privileges.

Options

Option	Name	Description
-t	type	Specifies the type of shut down. The types are: • *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. ⓘ The immediate and delay types are deprecated.

Operands

The server operand specifies which servers are to shut down. If no servers are given, then the default server will be terminated.

Standard error

The qterm command will write a diagnostic message to standard error for each error occurrence.

Exit status

Upon successful processing of all the operands presented to the qterm command, the exit status will be a value of zero.

If the qterm command fails to process any operand, the command exits with a value greater than zero.

Related Topics(non-Adaptive Computing topics)

> pbs_server(8B)
>
> qmgr(8B)
>
> pbs_resources_aix4(7B)
>
> pbs_resources_irix5(7B)
>
> pbs_resources_sp2(7B)

pbs_resources_sunos4(7B)

pbs_resources_unicos8(7B)

trqauthd

(Torque authorization daemon)

Synopsis

trqauthd <u>-D</u>

trqauthd <u>-d</u>

trqauthd <u>-F</u>

trqauthd <u>-n</u>

Description

The trqauthd daemon, introduced in Torque 4.0.0, replaced the pbs_iff authentication process. When users connect to pbs_server by calling one of the Torque utilities or by using the Torque APIs, the new user connection must be authorized by a trusted entity which runs as root. The advantage of trqauthd's doing this rather than pbs_iff is that trqauthd is resident, meaning you do not need to be loaded every time a connection is made; multi-threaded; scalable; and more easily adapted to new functionality than pbs_iff.

Beginning in Torque 4.2.6, trqauthd can remember the currently active pbs_server host, enhancing high availability functionality. Previously, trqauthd tried to connect to each host in the TORQUE_HOME/*<server_name>* file until it could successfully connect. Because it now remembers the active server, it tries to connect to that server first. If it fails to connect, it will go through the *<server_name>* file and try to connect to a host where an active pbs_server is running.

Beginning in Torque 6.1.0, you have the option when starting trqauthd to disable trqauthd from logging anything. In addition, the -F (don't fork) option is available when running under systemd.

Options

-D — Debug	
Format	---
Default	---

-D — Debug

Description	Run trqauthd in debug mode.
| **Example** | ```
trqauthd -D
``` |

-d — Terminate

Format	---
Default	---
Description	Terminate trqauthd.
| **Example** | ```
trqauthd -d
``` |

-F — Fork

Format	---
Default	---
Description	Prevents the system from forking. Useful when running under systemd (Red Hat 7-based or SUSE 12-based systems).
| **Example** | ```
trqauthd -F
``` |

-n — No Logging

Format	---
Default	---
Description	Disables trqauthd from logging anything.
| **Example** | ```
trqauthd -n
``` |

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_group_hosts	email_batch_seconds	lock_file_check_time	node_ping_rate
acl_hosts	exit_code_canceled_job	log_events	node_submit_exceptions
acl_host_enable	ghost_array_recovery	log_file_max_size	no_mail_force
acl_logic_or	gres_modifiers	log_file_roll_depth	np_default
acl_user_hosts	idle_slot_limit	log_keep_days	operators
allow_node_submit	interactive_jobs_can_roam	log_level	pass_cpuclock
allow_proxy_user	job_exclusive_on_use	mail_body_fmt	poll_jobs
auto_node_np	job_force_cancel_time	mail_domain	query_other_jobs
automatic_requeue_exit_code	job_full_report_time	mail_from	record_job_info
cgroup_per_task	job_log_file_max_size	mail_subject_fmt	record_job_script
checkpoint_defaults	job_log_file_roll_depth	managers	resources_available
clone_batch_delay	job_log_keep_days	max_job_array_size	scheduling
clone_batch_size	job_nanny	max_slot_limit	submit_hosts
copy_on_rerun	job_stat_rate	max_threads	tcp_incoming_timeout
cray_enabled	job_start_timeout	max_user_queuable	tcp_timeout
default_gpu_mode	job_suffix_alias	max_user_run	thread_idle_seconds
default_queue	job_sync_timeout	min_threads	timeout_for_job_delete
disable_automatic_requeue	keep_completed	moab_array_compatible	timeout_for_job_requeue
disable_server_id_check	kill_delay	mom_job_sync	use_jobs_subdirs
display_job_server_suffix	legacy_vmem	next_job_number	
dont_write_nodes_file	lock_file	node_check_rate	
down_on_error	lock_file_update_time	node_pack	

acl_group_hosts

Format	group@host[.group@host]...
Default	---

acl_group_hosts	
Description	Users who are members of the specified groups will be able to submit jobs from these otherwise untrusted hosts. Users who aren't members of the specified groups will not be able to submit jobs unless they are specified in acl_user_hosts.

acl_hosts	
Format	<HOST>[,<HOST>]... *or* <HOST>[range] *or* <HOST*> where the asterisk (*) can appear anywhere in the host name
Default	Not set.

Description	Specifies a list of hosts which can have access to pbs_server when acl_host_enable is set to TRUE. This does not enable a node to submit jobs. To enable a node to submit jobs use submit_hosts.
	
	ℹ️ Hosts which are in the TORQUE_HOME/server_priv/nodesfile do not need to be added to this list.
	
	```Qmgr: set queue batch acl_hosts="hostA,hostB"\nQmgr: set queue batch acl_hosts+=hostC\nQmgr: set server acl_hosts="hostA,hostB"\nQmgr: set server acl_hosts+=hostC```
	
	ℹ️ 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"\nQmgr: set server acl_hosts += "galaxy[0-50].tom.org"```

acl_host_enable	
Format	<BOOLEAN>
Default	**FALSE**
Description	When set to TRUE, hosts not in the pbs_server nodes file must be added to the **acl_hosts** list in order to get access to pbs_server.

acl_logic_or	
Format	<BOOLEAN>

acl_logic_or

Default	FALSE
Description	When set to TRUE, the user and group queue ACLs are logically OR'd. When set to FALSE, they are AND'd.

acl_user_hosts

Format	group@host[.group@host]...
Default	---
Description	The specified users are allowed to submit jobs from otherwise untrusted hosts. By setting this parameter, other users at these hosts will not be allowed to submit jobs unless they are members of specified groups in acl_group_hosts.

allow_node_submit

Format	<BOOLEAN>
Default	FALSE
Description	When set to TRUE, allows all hosts in the PBSHOME/server_priv/nodes file (MOM nodes) to submit jobs to pbs_server. To only allow qsub from a subset of all MOMs, use submit_hosts.

allow_proxy_user

Format	<BOOLEAN>
Default	FALSE
Description	When set to TRUE, specifies that users can proxy from one user to another. Proxy requests will be either validated by ruserok() or by the scheduler (see Job Submission).

auto_node_np

Format	<BOOLEAN>

auto_node_np

Default	**DISABLED**
Description	When set to TRUE, automatically configures a node's np (number of processors) value based on the ncpus value from the status update. Requires full manager privilege to set or alter.

automatic_requeue_exit_code

Format	<LONG>
Default	---
Description	This is an exit code, defined by the admin, that tells pbs_server to requeue the job instead of considering it as completed. This allows the user to add some additional checks that the job can run meaningfully, and if not, then the job script exits with the specified code to be requeued.

cgroup_per_task

Format	<BOOLEAN>
Default	**FALSE**
Description	When set to FALSE, jobs submitted with the -L syntax will have *one* cgroup created per host unless they specify otherwise at submission time. This behavior is similar to the pre-6.0 cpuset implementation.
	When set to TRUE, jobs submitted with the -L syntax will have *one* cgroup created per task unless they specify otherwise at submission time.
	ⓘ Some MPI implementations are not compatible with using one cgroup per task.
	See -L NUMA Resource Request for more information.

checkpoint_defaults

Format	<STRING>
Default	---

checkpoint_defaults

Description	Specifies for a queue the default checkpoint values for a job that does not have checkpointing specified. The checkpoint_defaults parameter only takes effect on execution queues. ```set queue batch checkpoint_defaults="enabled, periodic, interval=5"```

clone_batch_delay

Format	<INTEGER>
Default	1
Description	Specifies the delay (in seconds) between clone batches (see **clone_batch_size**).

clone_batch_size

Format	<INTEGER>
Default	256
Description	Job arrays are created in batches of size X. X jobs are created, and after the **clone_batch_delay**, X more are created. This repeats until all are created.

copy_on_rerun

Format	<BOOLEAN>
Default	FALSE
Description	When set to TRUE, Torque will copy the output and error files over to the user-specified directory when the grerun command is executed (i.e. a job preemption). Output and error files are only created when a job is in running state before the preemption occurs. ⓘ pbs_server and pbs_mom need to be on the same version. ⓘ When you change the value, you must perform a pbs_server restart for the change to effect.

cray_enabled

Format   <BOOLEAN>

<div align="center">**cray_enabled**</div>

Default	**FALSE**
Description	When set to TRUE, specifies that this instance of pbs_server has Cray hardware that reports to it. See Installation Notes for Moab and Torque for Cray in the *Moab Workload Manager Administrator Guide*.

<div align="center">**default_queue**</div>

Format	<STRING>
Default	---
Description	Indicates the queue to assign to a job if no queue is explicitly specified by the submitter.

default_gpu_mode

Format	<STRING>
Default	**exclusive_thread**
Description	Determines what GPU mode will be used for jobs that request GPUs but do not request a GPU mode. Valid entries are exclusive_thread, exclusive, exclusive_process, default, and shared. ℹ️ If you are using CUDA 8, the default of exclusive_thread is no longer supported. We recommend that you set the default to exclusive_process.

disable_automatic_requeue

Format	<BOOLEAN>
Default	**FALSE**
Description	Normally, if a job cannot start due to a transient error, the MOM returns a special exit code to the server so that the job is requeued instead of completed. When this parameter is set, the special exit code is ignored and the job is completed.

disable_server_id_check

Format	<BOOLEAN>

disable_server_id_check

Default	**FALSE**
Description	When set to TRUE, makes it so the user for the job doesn't have to exist on the server. The user must still exist on all the compute nodes or the job will fail when it tries to execute.
	⚠️ If you have **disable_server_id_check** set to TRUE, a user could request a group to which they do not belong. Setting VALIDATEGROUP to TRUE in the torque.cfg file prevents such a scenario (see "torque.cfg" Configuration File).

display_job_server_suffix

Format	<BOOLEAN>
Default	**TRUE**
Description	When set to TRUE, Torque will display both the job ID and the host name. When set to FALSE, only the job ID will be displayed.
	ℹ️ If set to *FALSE*, the environment variable *NO_SERVER_SUFFIX* must be set to *TRUE* for pbs_track to work as expected.
	ℹ️ display_job_server_suffix should not be set unless the server has no queued jobs. If it is set while the server has queued jobs, it will cause problems correctly identifying job ids with all existing jobs.

dont_write_nodes_file

Format	<BOOLEAN>
Default	**FALSE**
Description	When set to TRUE, the nodes file cannot be overwritten for any reason; qmgr commands to edit nodes will be rejected.

down_on_error

Format	<BOOLEAN>
Default	**TRUE**

down_on_error

Description	When set to `TRUE`, nodes that report an error from their node health check to pbs_server will be marked down and unavailable to run jobs.

email_batch_seconds

Format	<INTEGER>
Default	**0**
Description	If set to a number greater than 0, emails will be sent in a batch every specified number of seconds, per addressee. For example, if this is set to 300, then each user will only receive emails every 5 minutes in the most frequent scenario. The addressee would then receive one email that contains all of the information which would've been sent out individually before. If it is unset or set to 0, then emails will be sent for every email event.

exit_code_canceled_job

Format	<INTEGER>
Default	---
Description	When set, the exit code provided by the user is given to any job that is canceled, regardless of the job's state at the time of cancellation.

ghost_array_recovery

Format	<BOOLEAN>
Default	**TRUE**
Description	When `TRUE`, array subjobs will be recovered regardless of whether the .AR file was correctly recovered. This prevents the loss of running and queued jobs. However, it may *no longer* enforce a per-job slot limit or handle array dependencies correctly, as some historical information will be lost. When `FALSE`, array subjobs will not be recovered if the .AR file is invalid or non-existent.

gres_modifiers

Format	<INTEGER>

gres_modifiers

Default	---
Description	List of users granted permission to modify the gres resource of their own running jobs. Note that users do not need special permission to modify the gres resource of their own queued jobs.

idle_slot_limit

Format	<INTEGER>
Default	300
Description	Sets a default idle slot limit that will be applied to all arrays submitted after it is set.
	The idle slot limit is the maximum number of sub jobs from an array that will be instantiated at once. For example, if this is set to 2, and an array with 1000 sub jobs is submitted, then only two will ever be idle (queued) at a time. Whenever an idle sub job runs or is deleted, then a new sub job will be instantiated until the array no longer has remaining sub jobs.
	If this parameter is set, and user during job submission (using qsub -i) requests an idle slot limit that exceeds this setting, that array will be rejected. See also the qsub -i option.
Example	`qmgr -c 'set server idle_slot_limit = 50'`

interactive_jobs_can_roam

Format	<BOOLEAN>
Default	**FALSE**
Description	By default, interactive jobs run from the login node that they submitted from. When TRUE, interactive jobs may run on login nodes other than the one where the jobs were submitted from. See Installation Notes for Moab and Torque for Cray in the *Moab Workload Manager Administrator Guide*.
	ⓘ With interactive_jobs_can_roam enabled, jobs will only go to nodes with the alps_login property set in the nodes file.

job_exclusive_on_use

Format	<BOOLEAN>

job_exclusive_on_use

Default	FALSE
Description	When job_exclusive_on_use is set to *TRUE*, pbsnodes will show job-exclusive on a node when there's at least one of its processors running a job. This differs with the default behavior which is to show job-exclusive on a node when all of its processors are running a job.
Example	`set server job_exclusive_on_use=TRUE`

job_force_cancel_time

Format	<INTEGER>
Default	Disabled
Description	If a job has been deleted and is still in the system after *x* seconds, the job will be purged from the system. This is mostly useful when a job is running on a large number of nodes and one node goes down. The job cannot be deleted because the MOM cannot be contacted. The qdel fails and none of the other nodes can be reused. This parameter can used to remedy such situations.

job_full_report_time

Format	<INTEGER>
Default	300 seconds
Description	Sets the time in seconds that a job should be fully reported after any kind of change to the job, even if condensed output was requested.

job_log_file_max_size

Format	<INTEGER>
Default	---

job_log_file_max_size

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 `<file-name.3>`. If a file is already at the specified depth, such as `<filename.3>`, the file is deleted so it can be replaced by the incoming file roll, `<filename.2>`.

job_log_keep_days

Format	<INTEGER>
Default	---
Description	This maintains logs for the number of days designated. If set to 4, any log file older than 4 days old is deleted.

job_nanny

Format	<BOOLEAN>
Default	**FALSE**
Description	When set to TRUE, enables the experimental "job deletion nanny" feature. All job cancels will create a repeating task that will resend KILL signals if the initial job cancel failed. Further job cancels will be rejected with the message "job cancel in progress." This is useful for temporary failures with a job's execution node during a job delete request.

job_stat_rate

Format	<INTEGER>
Default	300 (30 in Torque 1.2.0p5 and earlier)
Description	If the mother superior has not sent an update by the specified time, at the specified time pbs_server requests an update on job status from the mother superior.

job_start_timeout

Format	<INTEGER>
Default	---
Description	Specifies the pbs_server to pbs_mom TCP socket timeout in seconds that is used when the pbs_server sends a job start to the pbs_mom. It is useful when the MOM has extra overhead involved in starting jobs. If not specified, then the **tcp_timeout** parameter is used.

job_suffix_alias

Format	<STRING>
Default	---
Description	Allows the job suffix to be defined by the user. ℹ️ job_suffix_alias should not be set unless the server has no queued jobs. If it is set while the server has queued jobs, it will cause problems correctly identifying job ids with all existing jobs.
Example	```qmgr -c 'set server job_suffix_alias = biology'``` When a job is submitted after this, its jobid will have .biology on the end: 14.napali.biology. If display_job_server_suffix is set to false, it would be named 14.biology.

job_sync_timeout

Format	<INTEGER>
Default	60

job_sync_timeout

Description	When a stray job is reported on multiple nodes, the server sends a kill signal to one node at a time. This timeout determines how long the server waits between kills if the job is still being reported on any nodes.

keep_completed

Format	<INTEGER>
Default	300
Description	The amount of time (in seconds) a job will be kept in the queue after it has entered the completed state. keep_completed *must* be set for job dependencies to work. For more information, see Keeping Completed Jobs.

kill_delay

Format	<INTEGER>
Default	If using **qdel**, 2 seconds If using **qrerun**, 0 (no wait)
Description	Specifies the number of seconds between sending a SIGTERM and a SIGKILL to a job you want to cancel. It is possible that the job script, and any child processes it spawns, can receive several SIGTERM signals before the SIGKILL signal is received. ⓘ All MOMs must be configured with $exec_with_exec true in order for **kill_delay** to work, even when relying on default **kill_delay** settings. ⓘ If kill_delay is set for a queue, the queue setting overrides the server setting. See kill_delay in Queue Attributes on page 409.
Example	```qmgr -c "set server kill_delay=30"```

legacy_vmem

Format	<BOOLEAN>
Default	FALSE

legacy_vmem

Description	When set to true, the vmem request will be the amount of memory requested for each node of the job. When it is unset or false, vmem will be the amount of memory for the entire job and will be divided accordingly

lock_file

Format	`<STRING>`
Default	`torque/server_priv/server.lock`
Description	Specifies the name and location of the lock file used to determine which high availability server should be active. If a full path is specified, it is used verbatim by Torque. If a relative path is specified, Torque will prefix it with `torque/server_priv`.

lock_file_update_time

Format	`<INTEGER>`
Default	3
Description	Specifies how often (in seconds) the thread will update the lock file.

lock_file_check_time

Format	`<INTEGER>`
Default	9
Description	Specifies how often (in seconds) a high availability server will check to see if it should become active.

log_events

Format	Bitmap
Default	---

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 file size is checked every 5 minutes, and if the *current day* file size is greater than or equal to this value then it will be rolled from *X* to *X.1* and a new empty log will be opened. Any value less than or equal to 0 will be ignored by pbs_server (the log will not be rolled).

log_file_roll_depth

Format	<INTEGER>
Default	1
Description	Controls how deep the current day log files will be rolled, if log_file_max_size is set, before they are deleted.

log_keep_days

Format	<INTEGER>

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_from

Format	<STRING>
Default	adm
Description	Specify the name of the sender whenTorquesends 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

managers	
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 modified with **qalter**. ```qmgr -c 'set server max_slot_limit=10'``` No array can request a slot limit greater than 10. Any array that does not request a slot limit receives a slot limit of 10. Using the example above, slot requests greater than 10 are rejected with the message: "Requested slot limit is too large, limit is 10."

max_user_run	
Format	\<INTEGER\>
Default	unlimited
Description	This limits the maximum number of jobs a user can have running for the given server.
Example	```qmgr -c "set server max_user_run=5"```

max_threads

Format	<INTEGER>
Default	The value of min_threads ((2 * the number of procs listed in `/proc/cpuinfo`) + 1) * 20
Description	This is the maximum number of threads that should exist in the thread pool at any time. See Setting min_threads and max_threads for more information.

max_user_queuable

Format	<INTEGER>
Default	Unlimited
Description	When set, `max_user_queuable` places a system-wide limit on the amount of jobs that an individual user can queue.

```
qmgr -c 'set server max_user_queuable=500'
```

min_threads

Format	<INTEGER>
Default	(2 * the number of procs listed in `/proc/cpuinfo`) + 1. If Torque is unable to read `/proc/cpuinfo`, the default is 10.
Description	This is the minimum number of threads that should exist in the thread pool at any time. See Setting min_threads and max_threads for more information.

moab_array_compatible

Format	<BOOLEAN>
Default	TRUE
Description	This parameter places a hold on jobs that exceed the slot limit in a job array. When one of the active jobs is completed or deleted, one of the held jobs goes to a queued state.

mom_job_sync

Format	<BOOLEAN>
Default	**TRUE**
Description	When set to TRUE, specifies that the pbs_server will synchronize its view of the job queue and resource allocation with compute nodes as they come online. If a job exists on a compute node, it will be automatically cleaned up and purged. (Enabled by default in Torque 2.2.0 and higher.) Jobs that are no longer reported by the mother superior are automatically purged by pbs_server. Jobs that pbs_server instructs the MOM to cancel have their processes killed in addition to being deleted (instead of leaving them running as in versions of Torque prior to 4.1.1).

next_job_number

Format	<INTEGER>
Default	---
Description	Specifies the ID number of the next job. If you set your job number too low and Torque repeats a job number that it has already used, the job will fail. Before setting next_job_number to a number lower than any number that Torque has already used, you must clear out your .e and .o files. ℹ️ If you use Moab Workload Manager (and have configured it to synchronize job IDs with Torque), then Moab will generate the job ID and next_job_number will have no effect on the job ID. See Resource Manager Configuration in the *Moab Workload Manager Administrator Guide* for more information.

node_check_rate

Format	<INTEGER>
Default	600
Description	Specifies the minimum duration (in seconds) that a node can fail to send a status update before being marked down by the pbs_server daemon.

node_pack

Description	This is deprecated.

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.

node_submit_exceptions	
Format	String
Default	---
Description	When set in conjunction with allow_node_submit, these nodes will not be allowed to submit jobs.

no_mail_force	
Format	<BOOLEAN>
Default	**FALSE**
Description	When set to TRUE, eliminates all e-mails when mail_options (see qsub) 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.
	ⓘ np_default sets a minimum number of np per node. Nodes with less than the np_default get additional execution slots.

operators

Format	`<user>@<host.sub.domain>[,<user>@<host.sub.domain>...]`
Default	root@localhost
Description	List of users granted batch operator privileges. Requires full manager privilege to set or alter.

pass_cpuclock

Format	`<BOOLEAN>`
Default	**TRUE**
Description	If set to *TRUE*, the pbs_server daemon passes the option and its value to the pbs_mom daemons for direct implementation by the daemons, making the CPU frequency adjustable as part of a resource request by a job submission. If set to *FALSE*, the pbs_server daemon creates and passes a `PBS_CPUCLOCK` job environment variable to the pbs_mom daemons that contains the value of the **cpuclock** attribute used as part of a resource request by a job submission. The CPU frequencies on the MOMs are not adjusted. The environment variable is for use by prologue and epilogue scripts, enabling administrators to log and research when users are making **cpuclock** requests, as well as researchers and developers to perform CPU clock frequency changes using a method outside of that employed by the Torque pbs_mom daemons.

poll_jobs

Format	`<BOOLEAN>`
Default	**TRUE** (FALSE in Torque 1.2.0p5 and earlier)
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**

query_other_jobs	
Description	When set to TRUE, specifies whether or not non-admin users may view jobs they do not own.

record_job_info	
Format	<BOOLEAN>
Default	**FALSE**
Description	This must be set to TRUE in order for job logging to be enabled.

record_job_script	
Format	<BOOLEAN>
Default	**FALSE**
Description	If set to TRUE, this adds the contents of the script executed by a job to the log. ℹ For record_job_script to take effect, record_job_info must be set to TRUE.

resources_available	
Format	<STRING>
Default	---
Description	Allows overriding of detected resource quantities (see Assigning Queue Resource Limits). pbs_server must be restarted for changes to take effect. Also, resources_available is constrained by the smallest of queue.resources_available and the server.resources_available.

scheduling	
Format	<BOOLEAN>
Default	---

scheduling

Description	Allows `pbs_server` to be scheduled. When `FALSE`, `pbs_server` is a resource manager that works on its own. When `TRUE`, Torque allows a scheduler, such as Moab or Maui, to dictate what `pbs_server` should do.

submit_hosts

Format	<HOSTNAME>[,<HOSTNAME>]...
Default	Not set.
Description	Hosts in this list are able to submit jobs. This applies to any node whether within the cluster or outside of the cluster.
	If acl_host_enable is set to `TRUE` and the host is not in the PBSHOME/server_priv/nodes file, then the host must also be in the acl_hosts list.
	To allow qsub from all compute nodes instead of just a subset of nodes, use allow_node_submit.

tcp_incoming_timeout

Format	<INTEGER>
Default	600
Description	Specifies the timeout for incoming TCP connections to pbs_server. Functions exactly the same as tcp_timeout, but governs incoming connections while tcp_timeout governs only outgoing connections (or connections initiated by pbs_server).
	ⓘ If you use Moab Workload Manager, prevent communication errors by giving **tcp_incoming_timeout** at least twice the value of the Moab RMPOLLINTERVAL. See RMPOLLINTERVAL for more information.

tcp_timeout

Format	<INTEGER>
Default	300

tcp_timeout

Description	Specifies the timeout for idle outbound 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 for additional information.).
	Use tcp_incoming_timeout to specify the timeout for idle inbound TCP connections.

thread_idle_seconds

Format	<INTEGER>
Default	300
Description	This is the number of seconds a thread can be idle in the thread pool before it is deleted. If threads should not be deleted, set to −1. Torque will always maintain at least **min_threads** number of threads, even if all are idle.

timeout_for_job_delete

Format	<INTEGER> (seconds)
Default	120
Description	The specific timeout used when deleting jobs because the node they are executing on is being deleted.

timeout_for_job_requeue

Format	<INTEGER> (seconds)
Default	120
Description	The specific timeout used when requeuing jobs because the node they are executing on is being deleted.

use_jobs_subdirs	
Format	<BOOLEAN>
Default	Not set (FALSE).

Description	Lets an administrator direct the way pbs_server will store its job-related files.
	
	• When use_jobs_subdirs is unset (or set to FALSE), job and job array files will be stored directly under $PBS_HOME/server_priv/jobs and $PBS_HOME/server_priv/arrays.
	
	• When use_jobs_subdirs is set to TRUE, job and job array files will be distributed over 10 subdirectories under their respective parent directories. This method helps to keep a smaller number of files in a given directory.
	
	ⓘ This setting does not automatically move existing job and job array files into the respective subdirectories. If you choose to use this setting (TRUE), you must first
	    ○ set use_jobs_subdirs to TRUE,
	    ○ shutdown the Torque server daemon,
	    ○ in the contrib directory, run the "use_jobs_subdirs_setup" python script with -m option,
	    ○ start the Torque server daemon.

Node Manager (MOM) Configuration

Under Torque, MOM configuration is accomplished using the `mom_priv/config` file located in the PBS directory on each execution server. You must create this file and insert any desired lines in a text editor (blank lines are allowed). When you modify the `mom_priv/config` file, you must restart pbs_mom.

The following examples demonstrate two methods of modifying the `mom_priv/config` file:

```
> echo "\$loglevel 3" > /var/spool/torque/mom_priv/config
```

```
> vim /var/spool/torque/mom_priv/config
...
$loglevel 3
```

For details, see these topics:

- Parameters
- Node Features and Generic Consumable Resource Specification
- Command-line Arguments

Related Topics

 Commands Overview
 Prologue and Epilogue Scripts

Parameters

These parameters go in the `mom_priv/config` file. They control various behaviors for the MOMs.

arch	$ignwalltime	$memory_pressure_	$restricted
$attempt_to_make_	$jobdirectory_sticky	threshold	$restricted
dir	$job_exit_wait_time	$mom_hierarchy_retry_time	$rpp_throttle
$clienthost	$job_output_file_unmask	$mom_host	size[fs=<FS>]
$check_poll_time	$job_starter	$node_check_script	$source_login_batch
$configversion	$job_starter_run_priviledged	$node_check_interval	$source_login_
$cputmult	$log_directory	$nodefile_suffix	interactive
$cray_check_rur	$log_file_suffix	$nospool_dir_list	$spool_as_final_name
$cuda_visible_	$logevent	opsys	$status_update_time
devices	$loglevel	$pbsclient	$thread_unlink_calls
$down_on_error	$log_file_max_size	$pbsserver	$timeout
$enablemomrestart	$log_file_roll_depth	$presetup_prologue	$tmpdir
$exec_with_exec	$log_keep_days	$prologalarm	$usecp
$ext_pwd_retry	$max_conn_timeout_micro_	$rcpcmd	$use_smt
$force_overwrite	sec	$remote_reconfig	$varattr
$ideal_load	$max_join_job_wait_time	$remote_checkpoint_dirs	$wallmult
$igncput	$max_load	$reduce_prolog_checks	$xauthpath
$ignmem	$memory_pressure_duration	$reject_job_submission	
$ignvmem			

arch

Format	<STRING>
Description	Specifies the architecture of the local machine. This information is used by the scheduler only.
Example	`arch ia64`

$attempt_to_make_dir

Format	<BOOLEAN>
Description	When set to *TRUE*, specifies that you want Torque to attempt to create the output directories for jobs if they do not already exist. Default is FALSE. ⓘ Torque uses this parameter to make the directory as the *user* and not as *root*. Torque will create the directory (or directories) ONLY if the user has permissions to do so.
Example	`$attempt_to_make_dir true`

$clienthost

Format	<STRING>
Description	Specifies the machine running pbs_server. ⚠️ This parameter is deprecated. Use **$pbsserver**.
Example	`$clienthost node01.teracluster.org`

$check_poll_time

Format	<STRING>
Description	Amount of time between checking running jobs, polling jobs, and trying to resend obituaries for jobs that haven't sent successfully. Default is 45 seconds.
Example	`$check_poll_time 90`

$configversion

Format	<STRING>
Description	Specifies the version of the config file data.
Example	`$configversion 113`

$cputmult

Format	<FLOAT>
Description	CPU time multiplier. ℹ️ If set to 0.0, MOM level cputime enforcement is disabled.
Example	`$cputmult 2.2`

$cray_check_rur

Format	\<BOOLEAN\>
Default	**TRUE**
Description	When set to *FALSE*, login MOMs (Cray only) will not look at the energy resource information used for each job. Bypassing Resource Utilization Reporting (RUR) checking may improve performance.
Example	`$cray_check_rur false`

$cuda_visible_devices

Format	\<BOOLEAN\>
Default	**TRUE**
Description	When set to *TRUE*, the MOM will set the CUDA_VISIBLE_DEVICES environment variable for jobs using NVIDIA GPUs. If set to *FALSE*, the MOM will not set CUDA_VISBLE_DEVICES for any jobs.
Example	`$cuda_visible_devices true`

$down_on_error

Format	\<BOOLEAN\>
Description	Causes the MOM to report itself as state "down" to pbs_server in the event of a failed health check. See Health check on page 235 for more information.
Example	`$down_on_error true`

$enablemomrestart

Format	\<BOOLEAN\>

$enablemomrestart

Description	Enables automatic restarts of the MOM. If enabled, the MOM will check if its binary has been updated and restart itself at a safe point when no jobs are running; thus making upgrades easier. The check is made by comparing the mtime of the pbs_mom executable. Command-line args, the process name, and the PATH env variable are preserved across restarts. It is recommended that this not be enabled in the config file, but enabled when desired with momctl (see Parameters for more information.)
Example	`$enablemomrestart true`

$exec_with_exec

Format	\<BOOLEAN\>
Description	pbs_mom uses the exec command to start the job script rather than the Torque default method, which is to pass the script's contents as the input to the shell. This means that if you trap signals in the job script, they will be trapped for the job. Using the default method, you would need to configure the shell to also trap the signals. Default is FALSE.
Example	`$exec_with_exec true`

$ext_pwd_retry

Format	\<INTEGER\>
Description	(*Available in Torque 2.5.10, 3.0.4, and later.*) Specifies the number of times to retry checking the password. Useful in cases where external password validation is used, such as with LDAP. The default value is 3 retries.
Example	`$ext_pwd_retry = 5`

$force_overwrite

Format	\<BOOLEAN\>
Description	(*Available in Torque 6.0.3 and later.*) When set to true, forces the output files to be overwritten each time a job is started.
Example	`$force_overwrite true`

$ideal_load

Format	<FLOAT>
Description	Ideal processor load.
Example	`$ideal_load 4.0`

$igncput

Format	<BOOLEAN>
Description	Ignores limit violation pertaining to CPU time. Default is FALSE.
Example	`$igncput true`

$ignmem

Format	<BOOLEAN>
Description	Ignores limit violations pertaining to physical memory. Default is FALSE.
Example	`$ignmem true`

$ignvmem

Format	<BOOLEAN>
Description	Ignores limit violations pertaining to virtual memory. Default is FALSE.
Example	`$ignvmem true`

$ignwalltime

Format	<BOOLEAN>
Description	Ignore walltime (do not enable MOM based walltime limit enforcement).
Example	`$ignwalltime true`

$jobdirectory_sticky

Format	<BOOLEAN>
Description	When this option is set (`true`), the job directory on the MOM can have a sticky bit set. The default is `false`.
Example	`$jobdirectory_sticky true`

$job_exit_wait_time

Format	<INTEGER>
Description	This is the timeout to clean up parallel jobs after one of the sister nodes for the parallel job goes down or is otherwise unresponsive. The MOM sends out all of its kill job requests to sisters and marks the time. Additionally, the job is placed in the substate `JOB_SUBSTATE_EXIT_WAIT`. The MOM then periodically checks jobs in this state and if they are in this state for more than the specified time, death is assumed and the job gets cleaned up. Default is 10 minutes.
Example	`$job_exit_wait_time 300`

$job_output_file_unmask

Format	<STRING>
Description	Uses the specified umask when creating job output and error files. Values can be specified in base 8, 10, or 16; leading 0 implies octal and leading 0x or 0X hexadecimal. A value of "userdefault" will use the user's default umask. This parameter is in version 2.3.0 and later.
Example	`$job_output_file_umask 027`

$job_starter

Format	<STRING>
Description	Specifies the fully qualified pathname of the job starter. If this parameter is specified, instead of executing the job command and job arguments directly, the MOM will execute the job starter, passing the job command and job arguments to it as its arguments. The job starter can be used to launch jobs within a desired environment.

$job_starter

Example	`$job_starter /var/torque/mom_priv/job_starter.sh`
	`> cat /var/torque/mom_priv/job_starter.sh`
	`#!/bin/bash`
	`export FOOHOME=/home/foo`
	`ulimit -n 314`
	`$*`

$job_starter_priviledged

Format	\<BOOLEAN\>
Description	When set to TRUE, specifies that you want Torque to execute the $job_starter script with elevated privileges. The default is FALSE.
Example	`$job_starter_run_privileged true`

$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	<INTEGER>
Description	Creates an event mask enumerating which log events will be recorded in the MOM logs. By default all events are logged. These are the events which can be chosen: ``` ERROR 0x0001 internal errors SYSTEM 0x0002 system (server) & (trqauthd) events ADMIN 0x0004 admin events JOB 0x0008 job related events JOB_USAGE 0x0010 End of Job accounting SECURITY 0x0020 security violation events SCHED 0x0040 scheduler events DEBUG 0x0080 common debug messages DEBUG2 0x0100 less needed debug messages CLIENTAUTH 0X0200 TRQAUTHD login events SYSLOG 0x0400 pass this event to the syslog as well ``` ℹ️ The listed events are shown here with hexidecimal values; however, a decimal value must be used when setting $logevent.
Example	$logevent 1039 will log ERROR, SYSTEM, ADMIN, JOB and SYSLOG events. This has a hexidecimal value of 0x40F.

$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_join_job_wait_time

Format	<INTEGER>
Description	The interval to wait for jobs stuck in a prerun state before deleting them from the MOMs and requeueing them on the server. Default is 10 minutes. ⓘ If a MOM is completely idle, it can take as long as the next MOM-to-server update time to requeue a failed job.
Example	`$max_join_job_wait_time 300`

$max_load

Format	<FLOAT>
Description	Maximum processor load.
Example	`$max_load 4.0`

$memory_pressure_duration

Format	<INTEGER>
Description	(*Applicable in version 3.0 and later.*) Memory pressure duration sets a limit to the number of times the value of memory_pressure_threshold can be exceeded before a process is terminated. This can only be used with **$memory_pressure_threshold**.
Example	`$memory_pressure_duration 5`

$memory_pressure_threshold

Format	<INTEGER>
Description	(*Applicable in version 3.0 and later.*) The memory_pressure of a cpuset provides a simple per-cpuset running average of the rate that the processes in a cpuset are attempting to free up in-use memory on the nodes of the cpuset to satisfy additional memory requests. The memory_pressure_threshold is an integer number used to compare against the reclaim rate provided by the memory_pressure file. If the threshold is exceeded and memory_pressure_duration is set, then the process terminates after exceeding the threshold by the number of times set in memory_pressure_duration. If memory_pressure duration is not set, then a warning is logged and the process continues. Memory_pressure_threshold is only valid with memory_pressure enabled in the root cpuset. To enable, log in as the super user and execute the command `echo 1 >> /dev/cpuset/memory_pressure_enabled`. See the cpuset man page for more information concerning memory pressure.
Example	`$memory_pressure_threshold 1000`

$mom_hierarchy_retry_time

Format	<SECONDS>

$mom_hierarchy_retry_time

Description	Specifies the amount of time that a MOM waits to retry a node in the hierarchy path after a failed connection to that node. The default is 90 seconds.
Example	`$mom_hierarchy_retry_time 30`

$mom_host

Format	<STRING>
Description	Sets the local hostname as used by pbs_mom.
Example	`$mom_host node42`

$node_check_script

Format	<STRING>
Description	Specifies the fully qualified pathname of the health check script to run (see Compute Node Health Check for more information).
Example	`$node_check_script /opt/batch_tools/nodecheck.pl`

$node_check_interval

Format	<STRING>
Description	Specifies the number of MOM intervals between subsequent executions of the specified health check. This value defaults to 1 indicating the check is run every MOM interval (see Compute Node Health Check 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 (before the prologue script). *jobend* – makes the node health script run after each job has completed on a node (after the epilogue script). The node health check may be configured to run before or after the job with the "jobstart" and/or "jobend" options. However, the job environment variables do not get passed to node health check script, so it has no access to those variables at any time.

$node_check_interval

Example	`$node_check_interval 5`

$nodefile_suffix

Format	<STRING>
Description	Specifies the suffix to append to a host names to denote the data channel network adapter in a multi-homed compute node.
Example	`$nodefile_suffix i` with the suffix of "i" and the control channel adapter with the name *node01*, the data channel would have a hostname of *node01i*.

$nospool_dir_list

Format	<STRING>
Description	If this is configured, the job's output is spooled in the working directory of the job or the specified output directory. Specify the list in full paths, delimited by commas. If the job's working directory (or specified output directory) is in one of the paths in the list (or a subdirectory of one of the paths in the list), the job is spooled directly to the output location. $nospool_dir_list * is accepted. The user that submits the job must have write permission on the folder where the job is written, and read permission on the folder where the file is spooled. Alternatively, you can use the $spool_as_final_name parameter to force the job to spool directly to the final output. ⓘ This should generally be used only when the job can run on the same machine as where the output file goes, or if there is a shared filesystem. If not, this parameter can slow down the system or fail to create the output file.
Example	`$nospool_dir_list /home/mike/jobs/,/var/tmp/spool/`

opsys

Format	<STRING>
Description	Specifies the operating system of the local machine. This information is used by the scheduler only.
Example	`opsys RHEL3`

$pbsclient

Format	<STRING>
Description	Specifies machines which the MOM daemon will trust to run resource manager commands via **momctl**. This may include machines where monitors, schedulers, or admins require the use of this command.
Example	`$pbsclient node01.teracluster.org`

$pbsserver

Format	<STRING>
Description	Specifies the machine running pbs_server. ℹ This parameter replaces the deprecated parameter **$clienthost**.
Example	`$pbsserver node01.teracluster.org`

$presetup_prologue

Format	<STRING>
Description	A full path to the presetup prologue for all jobs on this node. If set, this script executes before any setup for the job occurs (such as becoming the user, creating the output files, or changing directories). As a result, no output from this script will appear in the job's output.
Example	`$presetup_prologue /opt/kerberos_integration.sh`

$prologalarm

Format	<INTEGER>
Description	Specifies maximum duration (in seconds) which the MOM will wait for the job prologue or job epilogue to complete. The default value is 300 seconds (5 minutes). When running parallel jobs, this is also the maximum time a sister node will wait for a job to start.
Example	`$prologalarm 60`

Node Manager (MOM) Configuration

$rcpcmd

Format	<STRING>
Description	Specifies the full path and optional additional command line args to use to perform remote copies.
Example	mom_priv/config: $rcpcmd /usr/local/bin/scp -i /etc/sshauth.dat

$remote_reconfig

Format	<STRING>
Description	Enables the ability to remotely reconfigure pbs_mom with a new config file. Default is disabled. This parameter accepts various forms of true, yes, and 1. For more information on how to reconfigure MOMs, see **momctl** -r.
Example	$remote_reconfig true

$remote_checkpoint_dirs

Format	<STRING>
Description	Specifies which server checkpoint directories are remotely mounted. It tells the MOM which directories are shared with the server. Using remote checkpoint directories eliminates the need to copy the checkpoint files back and forth between the MOM and the server. All entries must be on the same line, separated by a space.
Example	$remote_checkpoint_dirs /checkpointFiles /bigStorage /fast *This informs the MOM that the* /checkpointFiles, /bigStorage, *and* /fast *directories are remotely mounted checkpoint directories.*

$reduce_prolog_checks

Format	<BOOLEAN>
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

$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 true`

$resend_join_job_wait_time

Format	<INTEGER>
Description	This is the timeout for the Mother Superior to re-send the join job request if it didn't get a reply from all the sister MOMs. The resend happens only once. Default is 5 minutes.
Example	`$resend_join_job_wait_time 120`

$restricted

Format	<STRING>
Description	Specifies hosts which can be trusted to access MOM services as non-root. By default, no hosts are trusted to access MOM services as non-root.
Example	`$restricted *.teracluster.org`

$rpp_throttle

Format	<INTEGER>
Description	This integer is in microseconds and causes a sleep after every RPP packet is sent. It is for systems that experience job failures because of incomplete data.
Example	`$rpp_throttle 100` (will cause a 100 microsecond sleep)

size[fs=<FS>]

Format	N/A
Description	Specifies that the available and configured disk space in the <FS> filesystem is to be reported to the pbs_server and scheduler.
	ⓘ To request disk space on a per job basis, specify the file resource as in `qsub -l nodes=1,file=1000kb`.
	ⓘ Unlike most MOM config options, the *size* parameter is not preceded by a "*$*" character.
Example	`size[fs=/localscratch]` The available and configured disk space in the `/localscratch` filesystem will be reported.

$source_login_batch

Format	<BOOLEAN>
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	<BOOLEAN>
Description	Specifies whether or not MOM will source the `/etc/profile`, etc. type files for *interactive* jobs. Parameter accepts various forms of true, false, yes, no, 1 and 0. Default is TRUE. This parameter is in version 2.3.1 and later.
Example	`$source_login_interactive False` MOM will bypass the sourcing of `/etc/profile`, etc. type files.

$spool_as_final_name

Format	<BOOLEAN>

$spool_as_final_name

Description	This makes the job write directly to its output destination instead of a spool directory. This allows users easier access to the file if they want to watch the jobs output as it runs.
Example	`$spool_as_final_name true`

$status_update_time

Format	\<INTEGER\>
Description	Specifies the number of seconds between subsequent MOM-to-server update reports. Default is 45 seconds.
Example	status_update_time: `$status_update_time 120` MOM will send server update reports every 120 seconds.

$thread_unlink_calls

Format	\<BOOLEAN\>
Description	Threads calls to unlink when deleting a job. Default is false. If it is set to TRUE, pbs_mom will use a thread to delete the job's files.
Example	thread_unlink_calls: `$thread_unlink_calls true`

$timeout

Format	\<INTEGER\>
Description	Specifies the number of seconds before a TCP connection on the MOM will timeout. Default is 300 seconds. In version 3.x and earlier, this specifies the number of seconds before MOM-to-MOM messages will timeout if RPP is disabled. Default is 60 seconds.
Example	`$timeout 120` A TCP connection will wait up to 120 seconds before timing out. For 3.x and earlier, MOM-to-MOM communication will allow up to 120 seconds before timing out.

$tmpdir

Format	<STRING>
Description	Specifies a directory to create job-specific scratch space (see Creating Per-Job Temporary Directories).
Example	`$tmpdir /localscratch`

$usecp

Format	<HOST>:<SRCDIR> <DSTDIR>
Description	Specifies which directories should be staged (see NFS and Other Networked Filesystems)
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. ⓘ $use_smt is deprecated. Please use the -L NUMA Resource Request syntax to control whether or not threads or cores are used. If you use SMT, you will need to set the *np* attribute so that each logical processor is counted.
Example	`$use_smt false`

$varattr

Format	<INTEGER> <STRING>

$varattr	
Description	Provides a way to keep track of dynamic attributes on nodes. <INTEGER> is how many seconds should go by between calls to the script to update the dynamic values. If set to -1, the script is read only one time. <STRING> is the script path. This script should check for whatever dynamic attributes are desired, and then output lines in this format: `name=value` Include any arguments after the script's full path. These features are visible in the output of **pbsnodes** `-a` varattr=Matlab=7.1;Octave=1.0. For information about using $varattr to request dynamic features in Moab, see REQATTR in the *Moab Workload Manager Administrator Guide*.
Example	`$varattr 25 /usr/local/scripts/nodeProperties.pl arg1 arg2 arg3`

$wallmult	
Format	<FLOAT>
Description	Sets a factor to adjust walltime usage by multiplying a default job time to a common reference system. It modifies real walltime on a per-MOM basis (MOM configuration parameters). The factor is used for walltime calculations and limits in the same way that cputmult is used for cpu time. ⓘ If set to 0.0, MOM level walltime enforcement is disabled.
Example	`$wallmult 2.2`

$xauthpath	
Format	<STRING>
Description	Specifies the path to the xauth binary to enable X11 forwarding.
Example	`$xauthpath /opt/bin/xauth/`

Related Topics

 Node Manager (MOM) Configuration

Node Features and Generic Consumable Resource Specification

Node features (a.k.a. "node properties") are opaque labels which can be applied to a node. They are not consumable and cannot be associated with a value. (Use generic resources described below for these purposes). Node features are configured within the nodes file on the **pbs_server** head node. This file can be used to specify an arbitrary number of node features.

Additionally, per node consumable generic resources may be specified using the format "<ATTR> <VAL>" with no leading dollar ("$") character. When specified, this information is routed to the scheduler and can be used in scheduling decisions. For example, to indicate that a given host has two tape drives and one node-locked matlab license available for batch jobs, the following could be specified:

`mom_priv/config:`

```
$clienthost 241.13.153.7
tape 2
matlab 1
```

Dynamic consumable resource information can be routed in by specifying a path preceded by an exclamation point. (!) as in the example below. If the resource value is configured in this manner, the specified file will be periodically executed to load the effective resource value.

`mom_priv/config:`

```
$clienthost 241.13.153.7
tape !/opt/rm/gettapecount.pl
matlab !/opt/tools/getlicensecount.pl
```

Related Topics

Node Manager (MOM) Configuration

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.

Flag	Description
C <directory>	Checkpoint path.
d <directory>	Home directory.
L <file>	Log file.
M <integer>	MOM port to listen on.
p	Perform 'poll' based job recovery on restart (jobs persist until associated processes terminate).
P	On restart, deletes all jobs that were running on MOM (Available in 2.4.X and later).
q	On restart, requeues all jobs that were running on MOM (Available in 2.4.X and later).
r	On restart, kills all processes associated with jobs that were running on MOM, and then requeues the jobs.
R <integer>	MOM 'RM' port to listen on.
S <integer>	pbs_server port to connect to.
v	Display version information and exit.
x	Disable use of privileged port.
?	Show usage information and exit.

For more details on these command-line options, see pbs_mom.

Related Topics

 Node Manager (MOM) Configuration

Diagnostics and Error Codes

Torque has a diagnostic script to assist you in giving Torque Support the files they need to support issues. It should be run by a user that has access to run all Torque commands and access to all Torque directories (this is usually root).

The script (`contrib/diag/tdiag.sh`) is available in Torque 2.3.8, Torque 2.4.3, and later. The script grabs the node file, server and MOM log files, and captures the output of `qmgr -c 'p s'`. These are put in a tar file.

The script also has the following options (this can be shown in the command line by entering `./tdiag.sh -h`):

```
USAGE: ./torque_diag [-d DATE] [-h] [-o OUTPUT_FILE] [-t
TORQUE_HOME]
```

- *DATE* should be in the format YYYYmmdd. For example, " 20091130" would be the date for November 30th, 2009. If no date is specified, today's date is used.
- *OUTPUT_FILE* is the optional name of the output file. The default output file is `torque_diag<today's_date>.tar.gz`. TORQUE_HOME should be the path to your Torque directory. If no directory is specified, `/var/spool/torque` is the default.

Table D-1: Torque error codes

Error code name	Number	Description
PBSE_FLOOR	15000	No error
PBSE_UNKJOBID	15001	Unknown job ID error
PBSE_NOATTR	15002	Undefined attribute
PBSE_ATTRRO	15003	Cannot set attribute, read only or insufficient permission
PBSE_IVALREQ	15004	Invalid request
PBSE_UNKREQ	15005	Unknown request
PBSE_TOOMANY	15006	Too many submit retries
PBSE_PERM	15007	Unauthorized Request

Error code name	Number	Description
PBSE_IFF_NOT_FOUND	15008	trqauthd unable to authenticate
PBSE_MUNGE_NOT_FOUND	15009	Munge executable not found, unable to authenticate
PBSE_BADHOST	15010	Access from host not allowed, or unknown host
PBSE_JOBEXIST	15011	Job with requested ID already exists
PBSE_SYSTEM	15012	System error
PBSE_INTERNAL	15013	PBS server internal error
PBSE_REGROUTE	15014	Dependent parent job currently in routing queue
PBSE_UNKSIG	15015	Unknown/illegal signal name
PBSE_BADATVAL	15016	Illegal attribute or resource value for
PBSE_MODATRRUN	15017	Cannot modify attribute while job running
PBSE_BADSTATE	15018	Request invalid for state of job
PBSE_UNKQUE	15020	Unknown queue
PBSE_BADCRED	15021	Invalid credential
PBSE_EXPIRED	15022	Expired credential
PBSE_QUNOENB	15023	Queue is not enabled
PBSE_QACESS	15024	Access to queue is denied
PBSE_BADUSER	15025	Bad UID for job execution
PBSE_HOPCOUNT	15026	Job routing over too many hops
PBSE_QUEEXIST	15027	Queue already exists
PBSE_ATTRTYPE	15028	Incompatible type

Error code name	Number	Description
PBSE_QUEBUSY	15029	Cannot delete busy queue
PBSE_QUENBIG	15030	Queue name too long
PBSE_NOSUP	15031	No support for requested service
PBSE_QUENOEN	15032	Cannot enable queue, incomplete definition
PBSE_PROTOCOL	15033	Batch protocol error
PBSE_BADATLST	15034	Bad attribute list structure
PBSE_NOCONNECTS	15035	No free connections
PBSE_NOSERVER	15036	No server specified
PBSE_UNKRESC	15037	Unknown resource type
PBSE_EXCQRESC	15038	Job exceeds queue resource limits
PBSE_QUENODFLT	15039	No default queue specified
PBSE_NORERUN	15040	Job is not rerunnable
PBSE_ROUTEREJ	15041	Job rejected by all possible destinations (check syntax, queue resources, ...)
PBSE_ROUTEEXPD	15042	Time in Route Queue Expired
PBSE_MOMREJECT	15043	Execution server rejected request
PBSE_BADSCRIPT	15044	(qsub) cannot access script file
PBSE_STAGEIN	15045	Stage-in of files failed
PBSE_RESCUNAV	15046	Resource temporarily unavailable
PBSE_BADGRP	15047	Bad GID for job execution

Error code name	Number	Description
PBSE_MAXQUED	15048	Maximum number of jobs already in queue
PBSE_CKPBSY	15049	Checkpoint busy, may retry
PBSE_EXLIMIT	15050	Resource limit exceeds allowable
PBSE_BADACCT	15051	Invalid Account
PBSE_ALRDYEXIT	15052	Job already in exit state
PBSE_NOCOPYFILE	15053	Job files not copied
PBSE_CLEANEDOUT	15054	Unknown job id after clean init
PBSE_NOSYNCMSTR	15055	No master found for sync job set
PBSE_BADDEPEND	15056	Invalid Job Dependency
PBSE_DUPLIST	15057	Duplicate entry in list
PBSE_DISPROTO	15058	Bad DIS based Request Protocol
PBSE_EXECTHERE	15059	Cannot execute at specified host because of checkpoint or stagein files
PBSE_SISREJECT	15060	Sister rejected
PBSE_SISCOMM	15061	Sister could not communicate
PBSE_SVRDOWN	15062	Request not allowed: Server shutting down
PBSE_CKPSHORT	15063	Not all tasks could checkpoint
PBSE_UNKNODE	15064	Unknown node
PBSE_UNKNODEATR	15065	Unknown node-attribute
PBSE_NONODES	15066	Server has no node list

Error code name	Number	Description
PBSE_NODENBIG	15067	Node name is too big
PBSE_NODEEXIST	15068	Node name already exists
PBSE_BADNDATVAL	15069	Illegal value for
PBSE_MUTUALEX	15070	Mutually exclusive values for
PBSE_GMODERR	15071	Modification failed for
PBSE_NORELYMOM	15072	Server could not connect to MOM
PBSE_NOTSNODE	15073	No time-share node available
PBSE_JOBTYPE	15074	Wrong job type
PBSE_BADACLHOST	15075	Bad ACL entry in host list
PBSE_MAXUSERQUED	15076	Maximum number of jobs already in queue for user
PBSE_BADDISALLOWTYPE	15077	Bad type in disallowed_types list
PBSE_NOINTERACTIVE	15078	Queue does not allow interactive jobs
PBSE_NOBATCH	15079	Queue does not allow batch jobs
PBSE_NORERUNABLE	15080	Queue does not allow rerunable jobs
PBSE_NONONRERUNABLE	15081	Queue does not allow nonrerunable jobs
PBSE_UNKARRAYID	15082	Unknown Array ID
PBSE_BAD_ARRAY_REQ	15083	Bad Job Array Request
PBSE_BAD_ARRAY_DATA	15084	Bad data reading job array from file
PBSE_TIMEOUT	15085	Time out
PBSE_JOBNOTFOUND	15086	Job not found

Error code name	Number	Description
PBSE_NOFAULTTOLERANT	15087	Queue does not allow fault tolerant jobs
PBSE_NOFAULTINTOLERANT	15088	Queue does not allow fault intolerant jobs
PBSE_NOJOBARRAYS	15089	Queue does not allow job arrays
PBSE_RELAYED_TO_MOM	15090	Request was relayed to a MOM
PBSE_MEM_MALLOC	15091	Error allocating memory - out of memory
PBSE_MUTEX	15092	Error allocating controling mutex (lock/unlock)
PBSE_THREADATTR	15093	Error setting thread attributes
PBSE_THREAD	15094	Error creating thread
PBSE_SELECT	15095	Error in socket select
PBSE_SOCKET_FAULT	15096	Unable to get connection to socket
PBSE_SOCKET_WRITE	15097	Error writing data to socket
PBSE_SOCKET_READ	15098	Error reading data from socket
PBSE_SOCKET_CLOSE	15099	Socket close detected
PBSE_SOCKET_LISTEN	15100	Error listening on socket
PBSE_AUTH_INVALID	15101	Invalid auth type in request
PBSE_NOT_IMPLEMENTED	15102	This functionality is not yet implemented
PBSE_QUENOTAVAILABLE	15103	Queue is currently not available
PBSE_TMPDIFFOWNER	15104	tmpdir owned by another user
PBSE_TMPNOTDIR	15105	tmpdir exists but is not a directory
PBSE_TMPNONAME	15106	tmpdir cannot be named for job

Error code name	Number	Description
PBSE_CANTOPENSOCKET	15107	Cannot open demux sockets
PBSE_CANTCONTACTSISTERS	15108	Cannot send join job to all sisters
PBSE_CANTCREATETMPDIR	15109	Cannot create tmpdir for job
PBSE_BADMOMSTATE	15110	Mom is down, cannot run job
PBSE_SOCKET_INFORMATION	15111	Socket information is not accessible
PBSE_SOCKET_DATA	15112	Data on socket does not process correctly
PBSE_CLIENT_INVALID	15113	Client is not allowed/trusted
PBSE_PREMATURE_EOF	15114	Premature End of File
PBSE_CAN_NOT_SAVE_FILE	15115	Error saving file
PBSE_CAN_NOT_OPEN_FILE	15116	Error opening file
PBSE_CAN_NOT_WRITE_FILE	15117	Error writing file
PBSE_JOB_FILE_CORRUPT	15118	Job file corrupt
PBSE_JOB_RERUN	15119	Job can not be rerun
PBSE_CONNECT	15120	Can not establish connection
PBSE_JOBWORKDELAY	15121	Job function must be temporarily delayed
PBSE_BAD_PARAMETER	15122	Parameter of function was invalid
PBSE_CONTINUE	15123	Continue processing on job. (Not an error)
PBSE_JOBSUBSTATE	15124	Current sub state does not allow trasaction.
PBSE_CAN_NOT_MOVE_FILE	15125	Error moving file
PBSE_JOB_RECYCLED	15126	Job is being recycled

Error code name	Number	Description
PBSE_JOB_ALREADY_IN_QUEUE	15127	Job is already in destination queue.
PBSE_INVALID_MUTEX	15128	Mutex is NULL or otherwise invalid
PBSE_MUTEX_ALREADY_ LOCKED	15129	The mutex is already locked by this object
PBSE_MUTEX_ALREADY_ UNLOCKED	15130	The mutex has already been unlocked by this object
PBSE_INVALID_SYNTAX	15131	Command syntax invalid
PBSE_NODE_DOWN	15132	A node is down. Check the MOM and host
PBSE_SERVER_NOT_FOUND	15133	Could not connect to batch server
PBSE_SERVER_BUSY	15134	Server busy. Currently no available threads

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 Resource Manager). 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), running jobs may be lost. In such cases, running jobs should be allowed to be completed or should be requeued before upgrading Torque.

- pbs_mom and pbs_server daemons of differing versions may be run together. However, not all combinations have been tested and unexpected failures may occur.

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

To upgrade

1. Build new release (do not install).
2. Stop all Torque daemons (see **qterm** and **momctl** `-s`).
3. Install new Torque (use *make install*).
4. Start all Torque daemons.

Rolling Upgrade

If you are upgrading to a new point release of your current version (for example, from 4.2.2 to 4.2.3) and not to a new major release from your current version (for example, from 4.1 to 4.2), you can use the following procedure to upgrade Torque without taking your nodes offline.

ℹ Because Torque version 4.1.4 changed the way that pbs_server communicates with the MOMs, it is not recommended that you perform a rolling upgrade of Torque from version 4.1.3 to 4.1.4.

To perform a rolling upgrade in Torque

1. Enable the pbs_mom flag on the MOMs you want to upgrade. The `enablemomrestart` option causes a MOM to check if its binary has been updated and restart itself at a safe point when no jobs are running. You can enable this in the MOM configuration file, but it is recommended that you use momctl instead.

   ```
   > momctl -q enablemomrestart=1 -h :ALL
   ```
   The `enablemomrestart` flag is enabled on all nodes.

2. Replace the pbs_mom binary, located in `/usr/local/bin` by default. pbs_mom will continue to run uninterrupted because the pbs_mom binary has already been loaded in RAM.

   ```
   > torque-package-mom-linux-x86_64.sh --install
   ```

   The next time pbs_mom is in an idle state, it will check for changes in the binary. If pbs_mom detects that the binary on disk has changed, it will restart automatically, causing the new pbs_mom version to load.

   After the pbs_mom restarts on each node, the `enablemomrestart` parameter will be set back to false (`0`) for that node.

ℹ If you have cluster with high utilization, you may find that the nodes never enter an idle state so pbs_mom never restarts. When this occurs, you must manually take the nodes offline and wait for the running jobs to complete before restarting pbs_mom. To set the node to an offline state, which will allow running jobs to complete but will not allow any new jobs to be scheduled on that node, use `pbsnodes -o <nodeName>`. After the new MOM has started, you must make the node active again by running `pbsnodes -c <nodeName>`.

Considerations Before Upgrading

Large Cluster Considerations

Torque has enhanced much of the communication found in the original OpenPBS project. This has resulted in a number of key advantages including support for:

- larger clusters.
- more jobs.
- larger jobs.
- larger messages.

In most cases, enhancements made apply to all systems and no tuning is required. However, some changes have been made configurable to allow site specific modification. The configurable communication parameters are: **node_check_rate**, **node_ping_rate**, and **tcp_timeout**.

For details, see these topics:

- Scalability Guidelines
- End-User Command Caching
- Moab and Torque Configuration for Large Clusters
- Starting Torque in Large Environments
- Other Considerations

Scalability Guidelines

In very large clusters (in excess of 1,000 nodes), it may be advisable to tune a number of communication layer timeouts. By default, PBS MOM daemons timeout on inter-MOM messages after 60 seconds. In Torque 1.1.0p5 and higher, this can be adjusted by setting the timeout parameter in the `mom_priv/config` file (see, Node Manager (MOM) Configuration). If 15059 errors (cannot receive message from sisters) are seen in the MOM logs, it may be necessary to increase this value.

Client-to-server communication timeouts are specified via the **tcp_timeout** server option using the **qmgr** command.

> ⓘ On some systems, *ulimit* values may prevent large jobs from running. In particular, the open file descriptor limit (i.e., `ulimit -n`) should be set to at least the maximum job size in procs + 20. Further, there may be value in setting the `fs.file-max` in `sysctl.conf` to a high value, such as:
>
> ```
> /etc/sysctl.conf:
> fs.file-max = 65536
> ```

Related Topics

[Large Cluster Considerations](#)

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

Moab and Torque Configuration for Large Clusters

There are a few basic configurations for Moab and Torque that can potentially improve performance on large clusters.

Moab configuration

In the `moab.cfg` file, add:

1. `RMPOLLINTERVAL 30,30` - This sets the minimum and maximum poll interval to 30 seconds.
2. `RMCFG[<name>] FLAGS=ASYNCSTART` - This tells Moab not to block until it receives a confirmation that the job starts.
3. `RMCFG[<name>] FLAGS=ASYNCDELETE` - This tells Moab not to block until it receives a confirmation that the job was deleted.

Torque configuration

1. Follow the Starting Torque in large environments recommendations.
2. Increase `job_start_timeout` on `pbs_server`. The default is *300* (5 minutes), but for large clusters the value should be changed to something like *600* (10

minutes). Sites running very large parallel jobs might want to set this value even higher.

3. Use a node health check script on all MOM nodes. This helps prevent jobs from being scheduled on bad nodes and is especially helpful for large parallel jobs.

4. Make sure that `ulimit -n` (maximum file descriptors) is set to *unlimited*, or a very large number, and not the default.

5. For clusters with a high job throughput it is recommended that the server parameter `max_threads` be increased from the default. The default is (2 * number of cores + 1) * 10.

6. Versions 5.1.3, 6.0.2, and later: if you have the server send emails, set `email_batch_seconds` appropriately. Setting this parameter will prevent `pbs_server` from forking too frequently and increase the server's performance. See email_batch_seconds for more information on this server parameter.

Related Topics

Large Cluster Considerations

Starting Torque in Large Environments

If running Torque in a large environment, use these tips to help Torque start up faster.

Fastest possible start up

1. Create a MOM hierarchy, even if your environment has a one-level MOM hierarchy (meaning all MOMs report directly to `pbs_server`), and copy the file to the `mom_priv` directory on the MOMs.

2. Start pbs_server with the -n option. This specifies that `pbs_server` won't send the hierarchy to the MOMs unless a MOM requests it.

3. Start the MOMs normally.

If no daemons are running

1. Start `pbs_server` with the -c option.

2. Start the MOMs without the `-w` option.

If MOMs are running and just restarting pbs_server

1. Start `pbs_server` without the `-c` option.

If restarting a MOM or all MOMs

1. Start `pbs_server` without the `-w` option. Starting it with `-w` causes the MOMs to appear to be down.

Related Topics

[Large Cluster Considerations]

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.

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 Moab Parameters in 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 (i.e. "Premature end of message") are reported within the scheduler, the "TIMEOUT" attribute of the RMCFG parameter may be set with a value of between 30 and 90 seconds.

File System

Torque can be configured to disable file system blocking until data is physically written to the disk by using the `--disable-filesync` argument with *configure*. While having filesync enabled is more reliable, it may lead to server delays for sites with either a larger number of nodes, or a large number of jobs. Filesync is enabled by default.

Network ARP Cache

For networks with more than 512 nodes it is mandatory to increase the kernel's internal ARP cache size. For a network of ~1000 nodes, we use these values in `/etc/sysctl.conf` on all nodes and servers:

```
/etc/sysctl.conf

# Don't allow the arp table to become bigger than this
net.ipv4.neigh.default.gc_thresh3 = 4096
# Tell the gc when to become aggressive with arp table cleaning.
# Adjust this based on size of the LAN.
net.ipv4.neigh.default.gc_thresh2 = 2048
# Adjust where the gc will leave arp table alone
net.ipv4.neigh.default.gc_thresh1 = 1024
# Adjust to arp table gc to clean-up more often
net.ipv4.neigh.default.gc_interval = 3600
# ARP cache entry timeout
net.ipv4.neigh.default.gc_stale_time = 3600
```

Use `sysctl -p` to reload this file.

The ARP cache size on other Unixes can presumably be modified in a similar way.

An alternative approach is to have a static `/etc/ethers` file with all hostnames and MAC addresses and load this by `arp -f /etc/ethers`. However, maintaining this approach is quite cumbersome when nodes get new MAC addresses (due to repairs, for example).

Related Topics

Large Cluster Considerations

Prologue and Epilogue Scripts

Torque provides administrators the ability to run scripts before and/or after each job executes. With such a script, a site can prepare systems, perform node health checks, prepend and append text to output and error log files, cleanup systems, and so forth.

The following table shows which MOM runs which script. All scripts must be in the `TORQUE_HOME/mom_priv/` directory and be available on every compute node. The "Mother Superior" is the pbs_mom on the first node allocated for a job. While it is technically a sister node, it is not a "Sister" for the purposes of the following table.

> ℹ The initial working directory for each script is `TORQUE_HOME/mom_priv/`.

Script	Execution location	Script Location	Execute as	File permissions
Prologue Scripts				
presetup.prologue	Mother Superior	8th argument	root	Readable and executable by root and NOT writable by anyone but root (e.g., `-r-x-----`)
prologue	Mother Superior	8th argument	root	Readable and executable by root and NOT writable by anyone but root (e.g., `-r-x-----`)
prologue.parallel	Sister	---	root	Readable and executable by root and NOT writable by anyone but root (e.g., `-r-x-----`)
prologue.user	Mother Superior	---	user	Readable and executable by root and other (e.g., `-r-x---r-x`)

Script	Execution location	Script Location	Execute as	File permissions
prologue.user.parallel	Sister	---	user	Readable and executable by root and other (e.g., `-r-x---r-x`)
Epilogue Scripts				
epilogue	Mother Superior	11th argument	root	Readable and executable by root and NOT writable by anyone but root (e.g., `-r-x-----`)
epilogue.parallel	Sister	---	root	Readable and executable by root and NOT writable by anyone but root (e.g., `-r-x-----`)
epilogue.precancel	Mother Superior This script runs after a job cancel request is received from pbs_server and before a kill signal is sent to the job process.	---	user	Readable and executable by root and other (e.g., `-r-x---r-x`)
epilogue.user	Mother Superior	---	user	Readable and executable by root and other (e.g., `-r-x---r-x`)
epilogue.user.parallel	Sister	---	user	Readable and executable by root and other (e.g., `-r-x---r-x`)

> ⓘ `epilogue.parallel` is available in version 2.1 and later.

This section contains these topics:

- [Script Order of Execution](#)
- [Script Environment](#)
- [Per Job Prologue and Epilogue Scripts](#)

- Prologue and Epilogue Scripts Time Out
- Prologue Error Processing

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 for more information.)

> ⓘ The node health check may be configured to run before or after the job with the "jobstart" and/or "jobend" options. However, the job environment variables do not get passed to node health check script, so it has no access to those variables at any time.

> ⓘ Root squashing is now supported for `epilogue` and `prologue` scripts.

Related Topics

Prologue and Epilogue Scripts

Script Environment

The `prologue` and `epilogue` scripts can be very simple. On most systems, the script must declare the execution shell using the `#!<SHELL>` syntax (for example, "`#!/bin/sh`"). In addition, the script may want to process context sensitive arguments passed by Torque to the script.

Prologue Environment

The following arguments are passed to the `presetup.prologue`, `prologue`, `prologue.user`, and `prologue.parallel` scripts:

Argument	Description
argv[1]	Job id.

Argument	Description
argv[2]	Job execution user name.
argv[3]	Job execution group name.
argv[4]	Job name (Torque 1.2.0p4 and higher only).
argv[5]	List of requested resource limits (Torque 1.2.0p4 and higher only).
argv[6]	Job execution queue (Torque 1.2.0p4 and higher only).
argv[7]	Job account (Torque 1.2.0p4 and higher only).
argv[8]	Job script location.
argv[9]	Comma-separated list of each host in the job. For example, if a job is using 10 cores on each of roshar, nalthis, elantris, and scadrial, this argument will have the value: roshar,n-althis,elantris,scadrial.

Epilogue Environment

Torque supplies the following arguments to the `epilogue`, `epilogue.user`, `epilogue.precancel`, and `epilogue.parallel` scripts:

Argument	Description
argv[1]	job id
argv[2]	job execution user name
argv[3]	job execution group name
argv[4]	job name
argv[5]	session id
argv[6]	list of requested resource limits
argv[7]	list of resources used by job

Argument	Description
argv[8]	job execution queue
argv[9]	job account
argv[10]	job exit code
argv[11]	job script location

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.

> ℹ The cancel job command (**qdel**) will take as long to return as the `epilogue.precancel` script takes to run. For example, if the script runs for 5 minutes, it takes 5 minutes for qdel to return.

For all scripts, the environment passed to the script is empty. However, if you submit the job using msub rather than qsub, some Moab environment variables are available in the Torque prologue and epilogue script environment: MOAB_CLASS, MOAB_GROUP, MOAB_JOBARRAYINDEX, MOAB_JOBARRAYRANGE, MOAB_JOBID, MOAB_JOBNAME, MOAB_MACHINE, MOAB_NODECOUNT, MOAB_NODELIST, MOAB_PARTITION, MOAB_PROCCOUNT, MOAB_QOS, MOAB_TASKMAP, and MOAB_USER. See **msub** on page 1 in the *Moab Workload Manager Administrator Guide* for more information.

Also, standard input for both scripts is connected to a system dependent file. Currently, for all systems this is `/dev/null`. Except for epilogue scripts of an interactive job, `prologue.parallel`, `epilogue.precancel`, and `epilogue.parallel`, the standard output and error are connected to output and error files associated with the job. For an interactive job, since the pseudo terminal connection is released after the job completes, the standard input and error point to `/dev/null`. For `prologue.parallel` and `epilogue.parallel`, the user will need to redirect `stdout` and `stderr` manually.

Related Topics

 Prologue and Epilogue Scripts

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

Prologue and Epilogue Scripts Time Out

Torque takes preventative measures against prologue and epilogue scripts by placing an alarm around the scripts execution. By default, Torque sets the alarm to go off after 5 minutes of execution. If the script exceeds this time, it will be terminated and the node will be marked down. This timeout can be adjusted by setting the **$prologalarm** parameter in the `mom_priv/config` file.

> ⓘ While Torque is executing the `epilogue`, `epilogue.user`, or `epilogue.precancel` scripts, the job will be in the *E* (exiting) state.

If an `epilogue.parallel` script cannot open the `.OU` or `.ER` files, an error is logged but the script is continued.

Related Topics

 Prologue and Epilogue Scripts

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-2:

Following are example prologue and epilogue scripts that write the arguments passed to them in the job's standard out file:

prologue	
Script	```#!/bin/sh
echo "Prologue Args:"
echo "Job ID: $1"
echo "User ID: $2"
echo "Group ID: $3"
echo ""

exit 0``` |
| **stdout** | ```Prologue Args:
Job ID: 13724.node01
User ID: user1
Group ID: user1``` |

epilogue	
Script	```#!/bin/sh
echo "Epilogue Args:"
echo "Job ID: $1"
echo "User ID: $2"
echo "Group ID: $3"
echo "Job Name: $4"
echo "Session ID: $5"
echo "Resource List: $6"
echo "Resources Used: $7"
echo "Queue Name: $8"
echo "Account String: $9"
echo ""

exit 0``` |
| **stdout** | ```Epilogue Args:
Job ID: 13724.node01
User ID: user1
Group ID: user1
Job Name: script.sh
Session ID: 28244
Resource List: neednodes=node01,nodes=1,walltime=00:01:00
Resources Used: cput=00:00:00,mem=0kb,vmem=0kb,walltime=00:00:07
Queue Name: batch
Account String:``` |

Example G-3:

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
```

> ℹ `prologue`, `prologue.user`, and `prologue.parallel` scripts can have dramatic effects on job scheduling if written improperly.

Related Topics

[Prologue and Epilogue Scripts](#)

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
- Configuring the Second Torque
- Bringing the First Torque Server online
- Bringing the Second Torque Server Online

Configuring the First Torque

```
./configure --with-server-home=/usr/spool/PBS1 --bindir=/usr/spool/PBS1/bin --
sbindir=/usr/spool/PBS1/sbin
```

Then make and make install will place the first Torque into `/usr/spool/PBS1` with the executables in their corresponding directories.

Configuring the Second Torque

```
./configure --with-server-home=/usr/spool/PBS2 --bindir=/usr/spool/PBS2/bin --
sbindir=/usr/spool/PBS2/sbin
```

Then make and make install will place the second Torque into `/usr/spool/PBS2` with the executables in their corresponding directories.

Bringing the First Torque Server online

Each command, including pbs_server and pbs_mom, takes parameters indicating which servers and ports to connect to or listen on (when appropriate). Each of these is documented in their corresponding man pages (configure with `--enable-docs`).

In this example the first Torque server will accept batch requests on port 35000, communicate with the MOMs on port 35001, and communicate via RPP on port 35002. The first Torque MOM will try to connect to the server on port 35000, it will listen for requests from the server on port 35001 and will communicate via RPP on port 35002. (Each of these command arguments is discussed in further details on the corresponding man page. In particular, `-t create` is only used the first time a server is run.)

```
> pbs_server -p 35000 -M 35001 -R 35002 -t create
> pbs_mom -S 35000 -M 35001 -R 35002
```

Afterwards, when using a client command to make a batch request it is
necessary to specify the server name and server port (35000):

```
> pbsnodes -a -s node01:35000
```

Submitting jobs can be accomplished using the `-q` option ([queue][@host
[:port]]):

```
> qsub -q @node01:35000 /tmp/script.pbs
```

Bringing the Second Torque Server Online

In this example the second Torque server will accept batch requests on port
36000, communicate with the MOMS on port 36002, and communicate via RPP
on port 36002. The second Torque MOM will try to connect to the server on port
36000, it will listen for requests from the server on port 36001 and will
communicate via RPP on port 36002.

```
> pbs_server -p 36000 -M 36001 -R 36002 -t create
> pbs_mom -S 36000 -M 36001 -R 36002
```

Afterward, when using a client command to make a batch request it is
necessary to specify the server name and server port (36002):

```
> pbsnodes -a -s node01:36000
> qsub -q @node01:36000 /tmp/script.pbs
```

Security Overview

The authorization model for Torque changed in version 4.0.0 from `pbs_iff` to a daemon called `trqauthd`. The job of the `trqauthd` daemon is the same as `pbs_iff`. The difference is that `trqauthd` is a resident daemon whereas `pbs_iff` is invoked by each client command. `pbs_iff` is not scalable and is prone to failure under even small loads. `trqauthd` is very scalable and creates the possibility for better security measures in the future.

trqauthd and pbs_iff Authorization Theory

The key to security of both `trqauthd` and `pbs_iff` is the assumption that any host which has been added to the Torque cluster has been secured by the administrator. Neither `trqauthd` nor `pbs_iff` do authentication. They only do authorization of users. Given that the host system is secure the following is the procedure by which `trqauthd` and `pbs_iff` authorize users to `pbs_server`.

1. Client utility makes a connection to `pbs_server` on a dynamic port.

2. Client utility sends a request to `trqauthd` with the user name and port.

3. `trqauthd` verifies the user ID and then sends a request to `pbs_server` on a privileged port with the user ID and dynamic port to authorize the connection.

4. `trqauthd` reports results of the server to client utility.

Both `trqauthd` and `pbs_iff` use Unix domain sockets for communication from the client utility. Unix domain sockets have the ability to verify that a user is who they say they are by using security features that are part of the file system.

Job Submission Filter ("qsub Wrapper")

When a "submit filter" exists, Torque will send the command file (or contents of STDIN if piped to qsub) to that script/executable and allow it to evaluate the submitted request based on specific site policies. The resulting file is then handed back to qsub and processing continues. Submit filters can check user jobs for correctness based on site policies. They can also modify user jobs as they are submitted. Some examples of what a submit filter might evaluate and check for are:

- Memory Request - Verify that the job requests memory and rejects if it does not.

- Job event notifications - Check if the job does one of the following and rejects it if it:

  - explicitly requests no notification.

  - requests notifications but does not provide an email address.

- Walltime specified - Verify that the walltime is specified.

- Global Walltime Limit - Verify that the walltime is below the global max walltime.

- Test Walltime Limit - If the job is a test job, this check rejects the job it if it requests a walltime longer than the testing maximum.

The script below reads the original submission request from STDIN and shows how you could insert parameters into a job submit request:

```
#!/bin/sh
# add default memory constraints to all requests
# that did not specify it in user's script or on command line
echo "#PBS -l mem=16MB"
while read i
do
echo $i
done
```

The same command line arguments passed to qsub will be passed to the submit filter and in the same order. Exit status of -1 will cause qsub to reject the submission with a message stating that it failed due to administrative policies.

The "submit filter" must be executable, must be available on each of the nodes where users may submit jobs, and by default must be located at `${libexecdir}/qsub_filter` (for version 2.1 and older: `/usr/local/sbin/torque_submitfilter`). At run time, if the file does not exist at this new preferred path then qsub will fall back to the old hard-coded path. The submit filter location can be customized by setting the *SUBMITFILTER* parameter inside the file (see "torque.cfg" Configuration File), as in the following example:

`torque.cfg`:

```
SUBMITFILTER /opt/torque/submit.pl
...
```

ℹ️ Initial development courtesy of Oak Ridge National Laboratories.

"torque.cfg" Configuration File

Administrators can configure the `torque.cfg` file (located in PBS_SERVER_ HOME (/var/spool/torque by default)) to alter the behavior of the qsub command on specific host machines where the file resides. This file contains a list of parameters and values separated by whitespace. This only affects qsub, and only on each specific host with the file.

Configuration Parameters

CLIENTRETRY	
Format	<INT>
Default	0
Description	Seconds between retry attempts to talk to pbs_server.
Example	```CLIENTRETRY 10``` *Torque waits 10 seconds after a failed attempt before it attempts to talk to pbs_server again.*

DEFAULTCKPT	
For mat	One of *None*, *Enabled*, *Shutdown*, *Periodic*, *Interval=minutes*, *depth=number*, or *dir=path*
Default	*None*
Description	Default value for job's checkpoint attribute. For a description of all possible values, see qsub ⓘ This default setting can be overridden at job submission with the qsub -c option.
Example	```DEFAULTCKPT Shutdown``` *By default, Torque checkpoints at pbs_mom shutdown.*

FAULT_TOLERANT_BY_DEFAULT

Format	<BOOLEAN>
Default	FALSE
Description	Sets all jobs to fault tolerant by default. (See `qsub -f` for more information on fault tolerance.)
| **Example** | ```
FAULT_TOLERANT_BY_DEFAULT TRUE
```<br>*Jobs are fault tolerant by default. They will not be canceled based on failed polling, no matter how many nodes fail to report.* |

HOST_NAME_SUFFIX

Format	<STRING>
Default	---
Description	Specifies a hostname suffix. When qsub submits a job, it also submits the username of the submitter and the name of the host from which the user submitted the job. Torque appends the value of HOST_NAME_SUFFIX to the hostname. This is useful for multi-homed systems that may have more than one name for a host.
| **Example** | ```
HOST_NAME_SUFFIX -ib
```<br>*When a job is submitted, the -ib suffix is appended to the host name.* |

QSUBHOST

Format	<HOSTNAME>
Default	---
Description	The hostname given as the argument of this option will be used as the PBS_O_HOST variable for job submissions. By default, PBS_O_HOST is the hostname of the submission host. This option allows administrators to override the default hostname and substitute a new name.
| **Example** | ```
QSUBHOST host1
```<br>*The default hostname associated with a job is host1.* |

QSUBSENDUID

Format	N/A
Default	---
Description	Integer for job's PBS_OUID variable. Specifying the parameter name anywhere in the config file enables the feature. Removing the parameter name disables the feature.
Example	```
QSUBSENDUID
```
*Torque assigns a unique ID to a job when it is submitted by qsub.* |

## QSUBSLEEP

| | |
|---|---|
| **Format** | <INT> |
| **Default** | 0 |
| **Description** | Specifies time, in seconds, to sleep between a user's submitting and Torque's starting a qsub command. Used to prevent users from overwhelming the scheduler. |
| **Example** | ```
QSUBSLEEP 2
```
When a job is submitted with qsub, it will sleep for 2 seconds. |

RERUNNABLEBYDEFAULT

Format	<BOOLEAN>
Default	TRUE
Description	Specifies if a job is re-runnable by default. Setting this to false causes the re-runnable attribute value to be false unless the users specifies otherwise with the **qsub** -r option. (New in Torque 2.4.)
Example	```
RERUNNABLEBYDEFAULT FALSE
```
*By default, qsub jobs cannot be rerun.* |

## SERVERHOST

| | |
|---|---|
| **Format** | \<STRING\> |
| **Default** | localhost |
| **Description** | If set, the qsub command will open a connection to the host specified by the SERVERHOST string. |
| **Example** | ```SERVERHOST orion15```<br><br>*The server will open socket connections and and communicate using serverhost* *orion15.* |

## SUBMITFILTER

| | |
|---|---|
| **Format** | \<STRING\> |
| **Default** | ${libexecdir}/qsub_filter (for version 2.1 and older: /usr/local/sbin/torque_submitfilter) |
| **Description** | Specifies the location of the submit filter (see Job Submission Filter ("qsub Wrapper") used to pre-process job submission. |
| **Example** | ```SUBMITFILTER /usr/local/sbin/qsub_filter```<br><br>*The location of the submit filter is specified as* `/usr/local/sbin/qsub_filter`. |

## TRQ_IFNAME

| | |
|---|---|
| **Format** | \<STRING\> |
| **Default** | null |
| **Description** | Allows you to specify a specific network interface to use for outbound Torque requests. The string is the name of a network interface, such as *eth0* or *eth1*, depending on which interface you want to use. |
| **Example** | ```TRQ_IFNAME eth1```<br><br>*Outbound Torque requests are handled by* *eth1.* |

## VALIDATEGROUP

| Format | <BOOLEAN> |
|---|---|
| Default | FALSE |
| Description | Validate submit user's group on **qsub** commands. For Torque builds released after 2/8/2011, *VALIDATEGROUP* also checks any groups requested in group_list at the submit host. Set *VALIDATEGROUP* to "TRUE" if you set **disable_server_id_check** to TRUE. |
| Example | ```VALIDATEGROUP TRUE```<br><br>*qsub* verifies the submitter's group ID. |

## VALIDATEPATH

| Format | <BOOLEAN> |
|---|---|
| Default | TRUE |
| Description | Validate local existence of '-d' working directory. |
| Example | ```VALIDATEPATH FALSE```<br><br>*qsub* does not validate the path. |

# Torque Quick Start Guide

## Initial Installation

Torque is now hosted at https://github.com under the adaptivecomputing organization. To download source, you will need to use the git utility. For example:

```
[root]# git clone https://github.com/adaptivecomputing.com/torque.git -b 6.1.0 6.1.0
```

To download a different version, replace each 6.1.0 with the desired version. After downloading a copy of the repository, you can list the current branches by typing `git branch -a` from within the directory of the branch you cloned.

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

Extract and build the distribution on the machine that will act as the "Torque server" - the machine that will monitor and control all compute nodes by running the pbs_server daemon. See the example below:

```
> tar -xzvf torque.tar.gz
> cd torque
> ./configure
> make
> make install
```

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

> ℹ️ 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`.

> ℹ️ In this document, TORQUE_HOME corresponds to where Torque stores its configuration files. The default is `/var/spool/torque`.

Initialize/Configure Torque on the Server (pbs_server)

- Once installation on the Torque server is complete, configure the pbs_server daemon by executing the command `torque.setup <USER>` found packaged with the distribution source code, where *<USER>* is a username

that will act as the Torque admin. This script will set up a basic batch queue to get you started. If you experience problems, make sure that the most recent Torque executables are being executed, or that the executables are in your current PATH.

> ⓘ If you are upgrading from Torque 2.5.9, run pbs_server -u before running `torque.setup`.
>
> ```
> [root]# pbs_server -u
> ```

- If doing this step manually, be certain to run the command `pbs_server -t create` to create the new batch database. If this step is not taken, the pbs_server daemon will be unable to start.
- Proper server configuration can be verified by following the steps listed in Testing server configuration.

Install Torque on the Compute Nodes

To configure a compute node do the following on each machine (see page 19, Section 3.2.1 of *PBS 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 (i.e. xCAT users might do `prcp torque-package-linux-i686.sh main:/tmp/; psh main /tmp/torque-package-linux-i686.sh --install`). Optionally, distribute and install the clients package.

Configure Torque on the Compute Nodes

- For each compute host, the MOM daemon must be configured to trust the pbs_server daemon. In Torque 2.0.0p4 and earlier, this is done by creating the `TORQUE_HOME/mom_priv/config` file and setting the **$pbsserver** parameter. In Torque 2.0.0p5 and later, this can also be done by creating the `TORQUE_HOME/server_name` file and placing the server hostname inside.
- Additional config parameters may be added to `TORQUE_HOME/mom_priv/config` (see Node Manager (MOM) Configuration 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). This enables running client commands.

Verify 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.

Enable the Scheduler

Selecting the cluster scheduler is an important decision and significantly affects cluster utilization, responsiveness, availability, and intelligence. The default Torque scheduler, pbs_sched, is very basic and will provide poor utilization of your cluster's resources. Other options, such as Maui Scheduler or Moab Workload Manager are highly recommended. If using Maui/Moab, see **Moab-Torque Integration Guide** on page 1 in the *Moab Workload Manager Administrator Guide*. If using pbs_sched, start this daemon now.

> ℹ️ 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

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
- Test 1 - Basic Operation
- Test 2 - Persistence of Checkpoint Images
- Test 3 - Restart After Checkpoint
- Test 4 - Multiple Checkpoint/Restart
- Test 5 - Periodic Checkpoint
- Test 6 - Restart from Previous Image

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

Test 1 - Basic Operation

Introduction

This test determines if the proper environment has been established.

Test Steps

Submit a test job and the issue a hold on the job.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
```

Possible Failures

Normally the result of qhold is nothing. If an error message is produced saying that qhold is not a supported feature then one of the following configuration errors might be present.

- The Torque images may have not be configured with `--enable-blcr`
- BLCR support may not be installed into the kernel with insmod.
- The config script in mom_priv may not exist with `$checkpoint_script` defined.
- The config script in mom_priv may not exist with `$restart_script` defined.
- The config script in mom_priv may not exist with `$checkpoint_run_exe` defined.
- The scripts referenced in the config file may not exist.
- The scripts referenced in the config file may not have the correct permissions.

Successful Results

If no configuration was done to specify a specific directory location for the checkpoint file, the default location is off of the Torque directory, which in my case is `/var/spool/torque/checkpoint`.

Otherwise, go to the specified directory for the checkpoint image files. This was done by either specifying an option on job submission, i.e. `-c dir=/home/test` or by setting an attribute on the execution queue. This is done with the command `qmgr -c 'set queue batch checkpoint_dir=/home/test'`.

Doing a directory listing shows the following.

```
# find /var/spool/torque/checkpoint
/var/spool/torque/checkpoint
/var/spool/torque/checkpoint/999.xxx.yyy.CK
/var/spool/torque/checkpoint/999.xxx.yyy.CK/ckpt.999.xxx.yyy.1205266630
# find /var/spool/torque/checkpoint |xargs ls -l
-r-------- 1 root root 543779 2008-03-11 14:17
/var/spool/torque/checkpoint/999.xxx.yyy.CK/ckpt.999.xxx.yyy.1205266630

/var/spool/torque/checkpoint:
total 4
drwxr-xr-x 2 root root 4096 2008-03-11 14:17 999.xxx.yyy.CK

/var/spool/torque/checkpoint/999.xxx.yyy.CK:
total 536
-r-------- 1 root root 543779 2008-03-11 14:17 ckpt.999.xxx.yyy.1205266630
```

Doing a `qstat -f` command should show the job in a held state, *job_state* = *H*. Note that the attribute checkpoint_name is set to the name of the file seen above.

If a checkpoint directory has been specified, there will also be an attribute *checkpoint_dir* in the output of `qstat -f`.

```
$ qstat -f
Job Id: 999.xxx.yyy
    Job_Name = test.sh
    Job_Owner = test@xxx.yyy
    resources_used.cput = 00:00:00
    resources_used.mem = 0kb
    resources_used.vmem = 0kb
    resources_used.walltime = 00:00:06
    job_state = H
    queue = batch
    server = xxx.yyy
    Checkpoint = u
    ctime = Tue Mar 11 14:17:04 2008
    Error_Path = xxx.yyy:/home/test/test.sh.e999
    exec_host = test/0
    Hold_Types = u
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Tue Mar 11 14:17:10 2008
    Output_Path = xxx.yyy:/home/test/test.sh.o999
    Priority = 0
    qtime = Tue Mar 11 14:17:04 2008
    Rerunable = True
    Resource_List.neednodes = 1
    Resource_List.nodect = 1
    Resource_List.nodes = 1
    Resource_List.walltime = 01:00:00
    session_id = 9402 substate = 20
    Variable_List = PBS_O_HOME=/home/test,PBS_O_LANG=en_US.UTF-8,
        PBS_O_LOGNAME=test,
        PBS_O_PATH=/usr/local/perltests/bin:/home/test/bin:/usr/local/s
bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games,
        PBS_O_SHELL=/bin/bash,PBS_SERVER=xxx.yyy,
        PBS_O_HOST=xxx.yyy,PBS_O_WORKDIR=/home/test,
        PBS_O_QUEUE=batch
    euser = test
    egroup = test
    hashname = 999.xxx.yyy
    queue_rank = 3
    queue_type = E comment = Job started on Tue Mar 11 at 14:17
    exit_status = 271
    submit_args = test.sh
    start_time = Tue Mar 11 14:17:04 2008
    start_count = 1
    checkpoint_dir = /var/spool/torque/checkpoint/999.xxx.yyy.CK
    checkpoint_name = ckpt.999.xxx.yyy.1205266630
```

ⓘ The value of `Resource_List.*` is the amount of resources requested.

Related Topics

  BLCR Acceptance Tests

Test 2 - Persistence of Checkpoint Images

Introduction

This test determines if the checkpoint files remain in the default directory after the job is removed from the Torque queue.

Note that this behavior was requested by a customer but in fact may not be the right thing to do as it leaves the checkpoint files on the execution node. These will gradually build up over time on the node being limited only by disk space. The right thing would seem to be that the checkpoint files are copied to the user's home directory after the job is purged from the execution node.

Test Steps

Assuming the steps of Test 1 (see Test 1 - Basic Operation), 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

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), 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

Test 4 - Multiple Checkpoint/Restart

Introduction

This test determines if the checkpoint/restart cycle can be repeated multiple times.

Test Steps

Start a job and then while tailing the job output, do multiple **qhold**/**qrls** operations.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
> qrls 999
> qhold 999
> qrls 999
> qhold 999
> qrls 999
```

Successful results

After each qrls, the job's output should resume. Also tried "while true; do qrls 999; qhold 999; done" and this seemed to work as well.

Related Topics

BLCR Acceptance Tests

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

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

Queue Attributes

This appendix provides information on the different queue attributes.

This appendix also lists some queue resource limits. See .

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

Attributes

acl_groups	disallowed_types	max_user_queuable	resources_default
acl_group_enable	enabled	max_user_run	resources_max
acl_group_sloppy	features_required	priority	resources_min
acl_hosts	ghost_queue	queue_type	route_destinations
acl_host_enable	keep_completed	req_information_max	started
acl_logic_or	kill_delay	req_information_min	
acl_users	max_queuable	required_login_property	
acl_user_enable	max_running	resources_available	

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@h1" > qmgr -c "set queue batch acl_groups+=staff@h1" ``` > **ⓘ** Used in conjunction with **acl_group_enable**.

acl_group_enable

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

acl_group_sloppy

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

acl_hosts

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

acl_host_enable

Format	<BOOLEAN>
Default	FALSE

acl_host_enable

Description	If **TRUE**, constrains Torque to only allow jobs submitted from hosts specified by the <u>acl_hosts</u> parameter.
| **Example** | ```
qmgr -c "set queue batch acl_host_enable=true"
``` |

acl_logic_or

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

acl_users

Format	<USER>[@<HOST>][+<USER>[@<HOST>]]...
Default	---
Description	Specifies the list of users who may submit jobs to the queue. If <u>acl_user_enable</u> 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@h1"
> qmgr -c "set queue batch acl_users+=stevek@h1"
```<br><br>ⓘ Used in conjunction with <u>acl_user_enable</u>. |

acl_user_enable

Format	<BOOLEAN>
Default	FALSE

acl_user_enable

Description	If **TRUE**, constrains Torque to only allow jobs submitted from users specified by the **acl_users** parameter.
Example	```qmgr -c "set queue batch acl_user_enable=true"```

disallowed_types

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

enabled

Format	<BOOLEAN>
Default	FALSE
Description	Specifies whether the queue accepts new job submissions.
Example	```qmgr -c "set queue batch enabled=true"```

features_required

Format	feature1[,feature2[,feature3...]]
Default	---
Description	Specifies that all jobs in this queue will require these features in addition to any they may have requested. A feature is a synonym for a property.

features_required

Example	```qmgr -c 's q batch features_required=fast'```

ghost_queue

Format	<BOOLEAN>
Default	FALSE
Description	Intended for automatic, internal recovery (by the server) only. If set to **TRUE**, the queue rejects new jobs, but permits the server to recognize the ones currently queued and/or running. Unset this attribute in order to approve a queue and restore it to normal operation. See Automatic Queue and Job Recovery on page 199 for more information regarding this process.
Example	```qmgr -c "unset queue batch ghost_queue"```

keep_completed

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

kill_delay

Format	<INTEGER>
Default	2

kill_delay	
Description	Specifies the number of seconds between sending a SIGTERM and a SIGKILL to a job in a specific queue that you want to cancel. It is possible that the job script, and any child processes it spawns, can receive several SIGTERM signals before the SIGKILL signal is received. ⓘ All MOMs must be configured with $exec_with_exec true in order for **kill_delay** to work, even when relying on default **kill_delay** settings. ⓘ This setting overrides the server setting. See kill_delay in <u>Server Parameters on page 309</u>.
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

max_user_queueable

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_queueable=20"```

max_user_run

Format	<INTEGER>
Default	unlimited
Description	This limits the maximum number of jobs a user can have running from the given queue.
Example	```qmgr -c "set queue batch max_user_run=10"```

priority

Format	<INTEGER>
Default	0
Description	Specifies the priority value associated with the queue.
Example	```qmgr -c "set queue batch priority=20"```

queue_type

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

req_information_max

Format	<STRING>
Default	---
Description	Specifies the maximum resource limits allowed for jobs submitted to a queue.
	ⓘ These limits apply only to the qsub -L job submission option.
	Valid values are lprocs, node, socket, numachip, core, thread, memory, swap, and disk.
	ⓘ If a maximum core count is specified, jobs with usecores must have lprocs<= the maximum core count; jobs without usecores are rejected.
	If a maximum thread count is specified, lprocs must be <= the maximum thread count.
Example	`qmgr -c "set queue batch req_information_max.lprocs=8"`

req_information_min

Format	<STRING>
Default	---
Description	Specifies the minimum resource limits allowed for jobs submitted to a queue.
	ⓘ These limits apply only to the qsub -L job submission option.
	Valid values are lprocs, node, socket, numachip, core, thread, memory, swap, and disk.
	ⓘ If a minimum core count is specified, jobs with usecores must have lprocs>= the minimum core count; jobs without usecores are rejected.
	If a minimum thread count is specified, lprocs must be >= the minimum thread count.
Example	`qmgr -c "set queue batch req_information_min.lprocs=2"`

required_login_property

Format	<STRING>

required_login_property

Default	---
Description	Adds the specified login property as a requirement for all jobs in this queue.
Example	`qmgr -c 's q <queuename> required_login_property=INDUSTRIAL'`

resources_available

Format	<STRING>
Default	---
Description	Specifies to cumulative resources available to all jobs running in the queue. See qsub will not allow the submission of jobs requesting many processors for more information.
Example	`qmgr -c "set queue batch resources_available.nodect=20"` ⓘ You must restart pbs_server for changes to take effect. Also, resources_available is constrained by the smallest of queue.resources_available and server.resources_available.

resources_default

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

resources_max

Format	<STRING>
Default	---

resources_max

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

resources_min

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

route_destinations

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

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

started	
Description	Specifies whether jobs in the queue are allowed to execute.
Example	```qmgr -c "set queue batch started=true"```

Assigning Queue Resource Limits

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

Resource	Format	Description
arch	string	Specifies the administrator defined system architecture required.
mem	size	Amount of physical memory used by the job. (Ignored on Darwin, Digital Unix, Free BSD, HPUX 11, IRIX, NetBSD, and SunOS. Also ignored on Linux if number of nodes is not 1. Not implemented on AIX and HPUX 10.)
ncpus	integer	Sets the number of processors in one task where a task cannot span nodes. ⓘ You cannot request both ncpus and nodes in the same queue.
nodect	integer	Sets the number of nodes available. By default, Torque will set the number of nodes available to the number of nodes listed in the `TORQUE_HOME/server_priv/nodes` file. nodect can be set to be greater than or less than that number. Generally, it is used to set the node count higher than the number of physical nodes in the cluster.
nodes	integer	Specifies the number of nodes.
procct	integer	Sets limits on the total number of execution slots (procs) allocated to a job. The number of procs is calculated by summing the products of all node and ppn entries for a job. For example `qsub -l nodes=2:ppn=2+3:ppn=4 job.sh` would yield a procct of 16. 2*2 (2:ppn=2) + 3*4 (3:ppn=4).
pvmem	size	Amount of virtual memory used by any single process in a job.

Resource	Format	Description
vmem	<u>size</u>	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