



ANU MIS “Bogong” Cluster

Administrator/User Guide

Version 1.0

Contents

- Contents 2
- Introduction 3
- Technical Specifications 4
 - Support & Service 4
 - Online Resources 4
- Using the Bogong Cluster 5
 - Installing the Operating System on Nodes 5
 - Creating User Accounts 5
 - Cluster Security and Node Updates 6
 - Using TORQUE 6
 - How TORQUE works 7
 - Checking Node Health 7
 - Bogong Queues 7
 - Submitting Jobs to a Queue 7
 - Running Interactive Jobs 10
 - View TORQUE Queues with qstat 11
 - Deleting Jobs with qdel 11
 - Deleting Jobs when qdel does not work 11

Introduction

The ANU MIS “Bogong” Cluster has been configured for useability and performance, using the latest computer hardware, operating system and cluster utilities. With two dual-core AMD Opteron processors in each node and a PathScale InfiniPath HTX interconnect adaptor, the Bogong Cluster provides the best performance low latency parallel computing platform for the price. Initially the cluster has been provided with 4 nodes, and can be upgraded to 24 nodes without requiring any additional InfiniBand switches.

Using the latest remote installation techniques, the cluster operating system on each node can be easily updated from the server when a new version is required. This provides an easy upgrade path as new Linux kernel enhancements become available. The cluster is supplied with Open SuSE Linux 10.0.

Cluster resource management is handled by TORQUE – a highly scalable resource manager which allows multiple users to run multiple jobs simultaneously, derived from OpenPBS. TORQUE includes a number of desirable features including arbitration of conflicting requests for resources, basic fault tolerance, and is simple enough for technically skilled users to understand its source and add functionality. MPI programs can be run through TORQUE in an interactive way for debugging etc, or in batch mode for production purposes. PathScale’s MPI has been installed and configured to use with TORQUE via mpirun.

NOTE: Users often want to run their jobs right away and get their results as quickly as possible. Scheduling systems can slow this down for all but the highest priority jobs, and users’ perception is often that the scheduler slows the cluster down. The reality is that using a scheduling system usually allows you to run a significantly higher number of total jobs on the cluster in a given time frame, thus dramatically increasing the overall performance of the system. When viewed objectively, this increased use of the cluster as a computing resource can outweigh the performance drop for individual jobs.

The cluster can be used to run jobs requiring a single processor or up to 16 processors. It is easily scalable by adding additional components and is designed so that modules are serviceable without the need for any tools. Cluster usage has been simplified with an easy to install, easy to manage and easy to run, integrated software solution which provides a single point of installation, login and administration.

When running jobs on the cluster, the following directories are accessed from the nodes by an NFS export on the head node:

```
/opt/cluster
```

```
/home
```

You can use local disk on the nodes with the directory:

```
/tmp
```

Technical Specifications

Physical Specifications	
Number of Nodes	<ul style="list-style-type: none"> • 4
Interconnect	<ul style="list-style-type: none"> • PathScale InfiniPath HTX • Voltaire ISR 9024M 24 port InfiniBand Switch
Node Architecture	<ul style="list-style-type: none"> • Two AMD Opteron Dual-core 2.2GHz Processors • HyperTransport Bus • 4GB ECC PC3200 DDR Memory • 160GB SATA Hard Disk
Server Architecture	<ul style="list-style-type: none"> • Two AMD Opteron Dual-core 2.2GHz Processors • HyperTransport Bus • 4GB ECC PC3200 DDR Memory • 400GB RAID 1 Storage

Software Specifications	
Operating System	<ul style="list-style-type: none"> • At time of commissioning, configured with Open SuSE Linux 10.0 (with kernel 2.6.13-15-smp)
Resource Manager	<ul style="list-style-type: none"> • TORQUE (Version 2.0.0p2)
MPI	<ul style="list-style-type: none"> • PathScale MPI 1.1-1972.684 • mpirun
Cluster Management Shell	<ul style="list-style-type: none"> • PDSH (Parallel Distributed Shell – version 2.8-1)

Environmental Specifications	
Power Requirements	<ul style="list-style-type: none"> • ~ 2,500W
Operating Temperature	<ul style="list-style-type: none"> • 18° – 25° C

Warranty & Support	
Warranty	<ul style="list-style-type: none"> • 36 months onsite parts and labour on hardware (next business day) • On-site emergency calls at standard callout fee
Support	<ul style="list-style-type: none"> • Ex-gratia support for 12 months is provided for hardware, operating system and cluster configuration issues • Email support - support@alexandertechnology.com

Support & Service

The system administrator should always be the first point of contact for any support or service issues with the cluster. If necessary, the system administrator can contact Alexander Technology for support.

Online Resources

Information in this handbook is a guide for doing many of the common administrative tasks required to set up and maintain the Bogong cluster. More extensive documentation on each of the major software component can be found at the following URL:

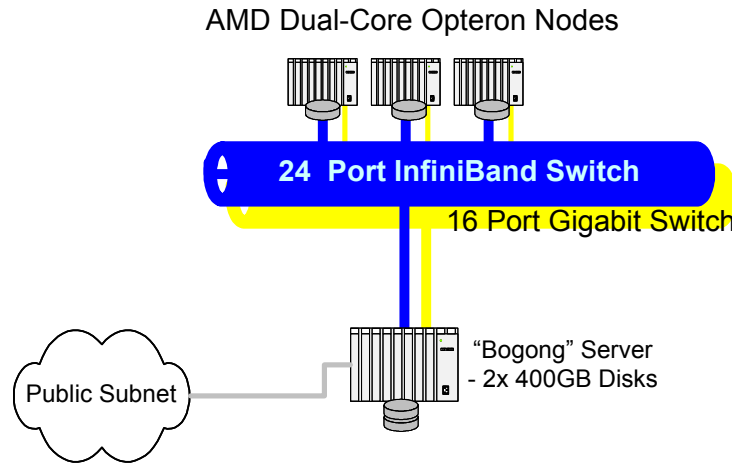
TORQUE admin manual <http://www.clusterresources.com/products/torque/docs/torqueadmin.shtml>

PDSH documentation <http://www.llnl.gov/linux/pdsh/pdsh.html>

Using the Bogong Cluster

The cluster architecture is very simple, consisting of a head node (cluster server) named “Bogong”, a 24 port InfiniBand switch, and 3 dedicated compute nodes (initially). Users log in to the head node, and execute their jobs using scripts through the resource manager TORQUE (PBS).

Alexander Technology™ PathFinder Cluster Architecture



Bogong can be used for running many single processor jobs simultaneously, and for running parallel jobs using MPI on up to 16 processors. Bogong uses the PathScale InfiniPath interconnect, which has latencies as low as 1.3 microseconds and bi-directional bandwidth up to 1.8GB/s.

Installing the Operating System on Nodes

The operating system on the cluster nodes is automatically installed over the network from the server when required. The procedure for this is as follows:

1. The node is switched on manually and boots initially from the network interface card PXE.
2. The node is allocated its IP address from the DHCP server on the head node.
3. PxeLinux boot starts on the node and checks the existence of a file within: `"/opt/cluster/nodeloader/tftboot/pxelinux.cfg"` on the server. If its IP address filename (in hexadecimal format) exists, then the node loads the operating system over the network.
4. Once the operating system is loaded and mounted, the node communicates this to the head node, which creates the node's IP address in the `pxelinux.cfg` directory.
5. The node then reboots from the hard disk drive.

Thus, if you require the node to reimage itself you need to remove the file from the `pxelinux.cfg` directory and reboot the node.

Creating User Accounts

All user accounts must be created using YAST to ensure all nodes are updated with user information. Do not use other methods as it will most likely break the cluster!

User information and other network information are obtained from the NIS server on the head node.

In the first instance that it appears that newly added users information has not propagated, first check what the NIS server is broadcasting. From one of the compute nodes, run `ypcat passwd | grep <username>` and check that the passwd entry is returned in the output.

If it is not, try manually updating the NIS cache on the headnode by running `make -C /var/yp` before contacting Alexander Technology Support.

Cluster Security and Node Updates

We recommend using YOU (YAST online update) to keep the server security current.

The nodes can be installed to a clean image from the server, in which case any individual updates are lost. To update the uncompressed node image, which is in:

```
/opt/cluster/nodeloader/nodeimage/node.img
```

1. First back it up!
2. Mount the image to a loopback device
3. chroot to the mounted directory
4. apply patches etc
5. exit chroot shell
6. sync
7. unmount the image

1. To backup the image just copy it to a different name:

```
cp /opt/cluster/nodeloader/nodeimage/node.img /opt/cluster/nodeloader/nodeimage/node.backup.img
```

2. To mount it:

```
mount -o loop /opt/cluster/nodeloader/nodeimage/node.img /mnt/image
```

and then:

```
cd /mnt/image
```

and you are inside the node image. You can edit files, move them around etc. So, you should be able to do:

```
chroot /mnt/image
```

and then you will get a shell inside the node image. Chroot jails are not perfect, please take note of any odd behaviour or errors and assess if they might cause a problem with the update.

When finished, exit the node image shell and do:

```
sync  
umount /mnt/image
```

to unmount the image.

To install the new image to all the nodes:

1. Delete all the files from `/opt/cluster/nodeloader/tftboot/pxelinux.cfg` (which lets the nodes know to load the image when they reboot)
2. Once all jobs on the cluster are ended/killed...
3. Reboot all the nodes
4. Check the nodes with `pbsnodes -a`

Using TORQUE

TORQUE (Terascale Open-Source Resource and Queue manager) is a cluster resource manager for providing control over batch jobs, and is derived from OpenPBS. TORQUE has been developed by many government and educational institutions for use in production clusters. Its features include:

- Run serial and parallel batch jobs remotely (create, route, execute, modify, delete)
- Define resource policies to determine resources for jobs
- Manage node availability

How TORQUE works

There are three essential components to TORQUE:

- pbs_server
- pbs_mom
- pbs_sched

The SERVER daemon runs on the head node (Bogong server) and handles all TORQUE commands such as qsub, qstat etc.

The MOM daemons (known as the MOMs) run on each nodes and the head node, monitor the nodes' health, restrict resources on nodes for job execution, and handle the jobs for the server.

The SCHEDULER runs on the head node for now and handles the order of job execution for jobs submitted to all the PBS queues.

Checking Node Health

To check the health and status of nodes, use the `pbsnodes` command for node query and control.

Common uses of the `pbsnodes` command are as follows:

To diagnose one node and report its health information, use:

```
pbsnodes -d nodeXX
```

To query all nodes and their attributes, use:

```
pbsnodes -ap
```

where the flag “p” forces a ping of all nodes to update the pbsnodes record, followed by:

```
pbsnodes -a
```

which then lists all the nodes and their attributes.

Bogong Queues

To submit a job you need to be logged into the cluster server (head node) and use the TORQUE `qsub` command. There is a default queue called `batch` which is used if no other queue is specified. The available queues are:

```
batch          (default queue – walltime = 01:00:00)
```

Submitting Jobs to a Queue

To submit a job you need to be logged into the cluster server (head node) and use the TORQUE `qsub` command. For example, so submit a simple serial job:

```
> qsub -l nodes=1 job_to_run.sh
```

To use a different queue, use the `-q` flag:

```
> qsub -l nodes=1 -q low job_to_run.sh
```

In the above example the job script only contains the commands to run the job. You cannot use a binary file as the job script.

You will generally pass options to TORQUE from a jobs script like this example:

```
#!/bin/bash

#PBS -l nodes=4
#PBS -l walltime=4:00:00

mpirun -machinefile $(PBS_NODEFILE) -np 16 -ppn 2 ~/jobs/mpi_job
exit
```

Any line that begins #PBS -l will pass options to TORQUE.

NOTE: Only use bash shell in your scripts as other shells may not work properly

We are using **epilogue** scripts with TORQUE which writes error output to the directory which was current when you submitted your job, with file name:

```
myscript.sh.eJOBID
```

and writes details of the job to the file:

```
myscript.sh.oJOBID
```

Which includes these details:

```
Jobid:                604.cluster
Job name:             myscript.sh
Resources requested:  neednodes=2,nodes=2,walltime=01:00:00
Resources used:      cput=00:00:00,mem=596kb,vmem=4992kb,walltime=00:00:10
```

You can find out which nodes you have been assigned by TORQUE using the command:

```
qstat -u username -n
```

on the head node, which outputs the list of your host nodes.

For best performance of the cluster, it is important that **walltimes** are set as accurately as possible, to ensure jobs are scheduled in the right queues.

Here are the #PBS options you can use (from the Torque website):

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
file	size*	The amount of total disk requested for the job
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
nice	integer between -20 (highest priority) and 19 (lowest priority)	Adjust the process’ execution priority

nodes	{<node_count> or <hostname>[:ppn=<ppn>][:<feature>[:<feature>]...] [+ ...]}	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 '+' character, "node_spec[+node_spec...]". Each node_spec is an number of nodes required of the type declared in the node_spec and a name or one or more property or 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=# - specifying the number of processors per node requested. Defaults to 1. * features - a string assigned by the system administrator specify a node's features. Check with your administrator as to the node names and properties available to you.
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.
pccput	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
pvmem	size*	Maximum amount of virtual memory used by any single process in the job
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.
vmem	size*	Maximum amount of virtual memory used by all concurrent processes in the job
walltime	seconds, or [[HH:]MM:]SS	Maximum amount of real time during which the job can be in the running state

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

Variable	Description
PBS_JOBNAME	user specified job name
PBS_O_WORKDIR	user specified job directory
PBS_TASKNUM	number of tasks requested
PBS_O_HOME	home directory of submitting user
PBS_MOMPORT	active port for mom daemon
PBS_O_LOGNAME	name of submitting user
PBS_O_LANG	language variable for job
PBS_JOBCOOKIE	job cookie
PBS_NODENUM	node offset number
PBS_O_SHELL	script shell
PBS_O_JOBID	unique PBS job id
PBS_O_HOST	host on which job script is currently running
PBS_QUEUE	job queue

PBS_NODEFILE	file containing line delimited list of nodes allocated to the job
PBS_O_PATH	path variable used to locate executables within job script

Job submission is accomplished using the **qsub** command. This command takes a number of command line arguments and integrates this into the specified *PBS command file*. The PBS command file is specified as a filename on the **qsub** command line.

- 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.

Parallel jobs are distributed to nodes using the **pbsdsh** command, which can be used with the following flag options:

```
pbsdsh [-s] [-v] program [args]
pbsdsh [-c copies] [-s] [-v] program [args]
pbsdsh [-n node] [-s] [-v] program [args]
```

pbsdsh executes (spawns) programs on one or more nodes under TORQUE, using the Task Manager API to distribute the program to the nodes.

When run without the **-c** or **-n** option, **pbsdsh** spawns the program on all the allocated nodes at about the same time. **pbsdsh** may be launched from within an interactive or non-interactive job.

pbsdsh options:

- c** Copies – the program is spawned on the first node allocated. This option is mutually exclusive with **-n**
- n** Node – the program is spawned on one node.
- s** Serial execution – the program is run in turn on each node, one after the other.
- v** Verbose – output of error conditions and task exit status is produced.

Here is an example using **pbsdsh**

```
#!/bin/bash

# All nodes to print "Hello World"
pbsdsh echo Hello World

exit
```

Running Interactive Jobs

You can use TORQUE to run interactive jobs using **qsub** with the **-I** option as in this example [assuming there are nodes available!, otherwise it will sleep until they are.]:

```
> qsub -I -l nodes=2
qsub: waiting for job 112.cluster to start
```

```
qsub: job 112.cluster ready
```

```
username@nodeXX:~>
```

The above example provides you with a shell on the first node assigned to you. You can then use all the nodes assigned to you as if there was no batch system running, except that when your walltime expires, your shells will all be killed. If you exit the shell assigned to you, your job will end.

From the shell on your first node (the execution node) run the following to find out which nodes you can use:

```
username@nodeXX:~> cat $PBS_NODEFILE
```

View TORQUE Queues with `qstat`

Run `qstat` on the head node to see what jobs are queued (Q), running (R), holding (H) or exiting (E):

```
> qstat
```

Job id	Name	User	Time Use	S	Queue
112.cluster	STDIN	alex	00:00:00	R	short

See the `qstat` man page for other options.

Deleting Jobs with `qdel`

Simply use `qdel` with the job number as argument:

```
qdel 112
```

Deleting Jobs when `qdel` does not work

If you need to delete stale jobs from the queue in TORQUE and the `qdel` command doesn't work, ask the administrator for help.