

Table of Contents

1 User Guide	1
1.1 Introduction	1
1.2 How to Logon to the Cluster	1
1.3 Home Directories	1
1.4 How to Submit Jobs to GE	1
1.5 How to Submit MPI jobs to GE	1
1.6 How to View Status and Delete Jobs from GE	1
1.7 How to use Modules Command/Tool	1
1.8 How to Compile and Build a MPI Application	1
1.9 UserGuide How to View Status of Cluster	1
1.9.1 Ganglia	1
1.9.2 Cacti	1
1.10 Grid Engine Reference Guide	2
2 UserGuide Grid Engine Cheat Sheet	3
3 UserGuide How to Compile and Build a MPI Application	6
3.1 Introduction	6
3.2 Supported Implementations	6
3.3 Scope	6
3.4 MPI Programs	6
3.5 C	6
3.5.1 OpenMPI	6
3.5.2 MPICH1	6
3.5.3 MPICH2	6
3.5.4 MVAPICH1	7
3.5.5 MVAPICH2	7
3.6 C++	7
3.6.1 OpenMPI	7
3.6.2 MPICH1	7
3.6.3 MPICH2	7
3.6.4 MVAPICH1	7
3.6.5 MVAPICH2	7
3.7 Fortran 77	7
3.7.1 OpenMPI	8
3.7.2 MPICH1	8
3.7.3 MPICH2	8
3.7.4 MVAPICH1	8
3.7.5 MVAPICH2	8
3.8 Fortran 90	8
3.8.1 OpenMPI	8
3.8.2 MPICH1	8
3.8.3 MPICH2	8
3.8.4 MVAPICH1	9
3.8.5 MVAPICH2	9
4 UserGuide How to Submit Jobs to GE	10
4.1 Introduction	10
4.2 Environment Setup	10
4.3 Job Submission Possibilities	10
4.3.1 qsub - Submit Job	10
4.3.2 qsh - Submit an Interactive X-windows Session	11
4.3.3 qlogin - Submit an Interactive Login Session	11
4.3.4 qrsh - Submit an Interactive RSH Session	11
4.3.5 qresub - Submit a Copy of an Existing Job	11
4.3.6 qmake - Distributed Parallel Make	11
4.3.7 qtchsh - TCSH with Transparent Remote Execution	11
4.3.8 qmon - Graphical User Interface	11
4.3.9 qalter - Modify Pending Job	11
4.4 Environment Variables	11
4.5 Script-Embedded Options (Active Comments)	13
5 UserGuide How to Submit MPI jobs to GE	14
5.1 Assumptions	14
5.2 OpenMPI	14
5.2.1 Parallel Environment	14
5.2.2 Wrapper Script	14
5.2.3 Job Submission	14
5.2.4 Example	14
5.3 MPICH1	14

Table of Contents

5 UserGuide How to Submit MPI jobs to GE	
5.3.1 Parallel Environment.....	15
5.3.2 Wrapper Script.....	15
5.3.3 Job Submission.....	15
5.4 MPICH2.....	15
5.4.1 Parallel Environment.....	15
5.4.2 Wrapper Script.....	15
5.4.3 Job Submission.....	16
5.5 MVAPICH1.....	16
5.5.1 Infiniband Resources.....	16
5.5.2 Parallel Environment.....	16
5.5.3 Wrapper Script.....	16
5.5.4 Job Submission.....	16
5.6 MVAPICH2.....	16
5.6.1 Infiniband Resources.....	17
5.6.2 Parallel Environment.....	17
5.6.3 Wrapper Script.....	17
5.6.4 Job Submission.....	17
6 UserGuide How to View Status and Delete Jobs from GE.....	18
6.1 Job Status.....	18
6.1.1 Resource Restriction.....	18
6.2 Suspending a Job.....	18
6.3 Applying a User Hold on a Job.....	18
6.4 Deleting a Job.....	18
7 Tool.....	19
7.1 Overview.....	19
7.2 Example.....	19
7.3 Usage.....	19

1 User Guide

1.1 Introduction

This Users Guide will provide an overview of the steps necessary to logon to the cluster and use SGE to submit and monitor jobs and also demonstrate other means of monitoring the cluster.

1.2 How to Logon to the Cluster

NOTE: By default, UniCluster 4.0 uses standard `/etc/passwd` for user accounts. Your system administrator may have changed that. If so, please contact your administrator for details.

The controlling point of the cluster is the installer node. Any account that you have on that machine is synchronized to all of the compute nodes in the cluster. You should be able to user secure shell (SSH) to connect to any of the nodes.

1.3 Home Directories

The directories in `/home` are NFS mounted from the installer node by default. Therefore, any files that you create in your home directory will automatically appear on all other nodes.

1.4 How to Submit Jobs to GE

How to Submit Jobs to GE

1.5 How to Submit MPI jobs to GE

How to Submit MPI jobs to GE

1.6 How to View Status and Delete Jobs from GE

How to View Status and Delete Jobs from GE

1.7 How to use Modules Command/Tool

How to use Modules Command/Tool

1.8 How to Compile and Build a MPI Application

How to Compile and Build a MPI Application

1.9 UserGuide How to View Status of Cluster

1.9.1 Ganglia

To view the status of a cluster with Ganglia, the Ganglia kit must be installed and deployed within the cluster (see the Administrator Guide for details on deploying kits). Once it has been deployed, pointing a web browser at the node with the gmetad component installed on it (likely the installer node). For example, if the component is installed on "localhost", the URL for Ganglia will be <http://localhost/ganglia>. Also, if the gmetad component is installed on the Installer node there will be a link from the UniCluster homepage (<http://localhost>) on the Installer node directly to the Ganglia web interface.

1.9.2 Cacti

To view the status of a cluster with Cacti, the Cacti kit must be installed and deployed within the cluster (see the Administrator Guide for details on deploying kits). Once it has been deployed, pointing a web browser at the node with the cacti component installed on it (likely the installer node). For example, if the component is installed on "localhost", the URL for Cacti will be <http://localhost/cacti>. The default username for cacti is "admin" and the default password is also "admin" although your administrator may have changed these.

Also, if the cacti component is installed on the Installer node there will be a link from the UniCluster homepage (<http://localhost>) on the Installer node directly to the Cacti web interface.

1.10 Grid Engine Reference Guide

UserGuide Grid Engine Cheat Sheet

[UniCluster Documentation]

2 UserGuide Grid Engine Cheat Sheet

Action	Target	Command	Switch	Notes
Create	Admin Host	qconf	-ah <i>name</i>	
	Calendar		-acal <i>name</i>	
	Checkpointing Environment		-ackpt <i>name</i>	
	Complex Variable		-mc	
	Host Configuration		-aconf <i>name</i>	
	Execution Host Configuration		-ae [<i>config to copy</i>]	If a name is given, it is treated as an already existing host configuration to copy.
	Host Group		-ahgrp @ <i>name</i>	
	Manager		-am <i>name</i>	
	Operator		-ao <i>name</i>	
	Parallel Environment		-ap <i>name</i>	
	Project		-aprj <i>name</i>	
	Queue		-aq <i>name</i>	
	Resource Quota Set		-arqs [<i>name</i>]	
	Submit Tree		-astree	
	Submit Host		-as <i>name</i>	
	User		-auser	
User List	-au <i>user name list name</i>	If the given user list does not already exist, it will be created		
Delete	Admin Host	qconf	-dh <i>name</i>	
	Calendar		-dcal <i>name</i>	If name is ?all?, all execution hosts will be killed.
	Checkpointing Environment		-dckpt <i>name</i>	
	Complex Variable		-mc	
	Host Configuration		-dconf <i>name</i>	The host configuration named "global" cannot be deleted.
	Execution Host Configuration		-ae <i>name</i>	
	Host Group		-dhgrp @ <i>name</i>	
	Manager		-dm <i>name</i>	
	Operator		-do <i>name</i>	
	Parallel Environment		-ap <i>name</i>	
	Project		-dprj <i>name</i>	
	Queue		-dq <i>name</i>	
	Resource Quota Set		-drqs <i>name</i>	
	Submit Tree		-dstree	
	Submit Host		-ds <i>name</i>	
	User		-duser <i>name</i>	
User List	-dul <i>list name</i>			
User From User List	-du <i>user name list name</i>			

Modify	Calendar	-mcal <i>name</i>	
	Checkpointing Environment	-mckpt <i>name</i>	
	Complex Variable	-mc	
	Host Configuration	-mconf [<i>name</i>]	If a name is not given, the name is assumed to be "global"
	Execution Host Configuration	-me <i>name</i>	
	Host Group	-mhgrp @ <i>name</i>	
	Parallel Environment	-mp <i>name</i>	
	Project	-mprj <i>name</i>	
	Queue	-mq <i>name</i>	
	Resource Quota Set	-mrqs <i>name</i>	
	Scheduler Configuration	-msconf	
	Share Tree	-mstree	
	User	-muser <i>name</i>	
	User List	-mu <i>list name</i>	
User From User List	-au <i>user name list name</i>		
Show	Calendar	-scal	
	Checkpointing Environment	-sckpt <i>name</i>	
	Execution Host Configuration	-se <i>name</i>	
	Host Configuration	-sconf [<i>name</i>]	If a name is not given, the name is assumed to be "global"
	Host Group	-shgrp @ <i>name</i>	
	Parallel Environment	-sp <i>name</i>	
	Project	-sprj <i>name</i>	
	Queue	-sq <i>name</i>	
	Resource Quota Set	-srqs <i>name</i>	
	Scheduler	-sss	
	Scheduler Configuration	-ssconf	
	Share Tree	-sstree	
	User	-suser <i>name</i>	
	User List	-su <i>list name</i>	
Show List	Admin Host	-sh	
	Calendar	-scall	
	Checkpointing Environment	-sckptl	
	Complex Variable	-sc	
	Event Client	-secl	
	Execution Host Configuration	-sel	
	Host Configuration	-sconfl	
	Host Group	-shgrppl	

	Manager		-sm	
	Operator		-so	
	Parallel Environment		-spl	
	Project		-sprjl	
	Queue		-sql	
	Resource Quota Set		-srqsl	
	Submit Host		-ss	
	User		-suserl	
	User List		-sul	
Terminate	Execution Host		-ke <i>name</i>	
	Qmaster		-km	
	Scheduler		-ks	
	Job	qdel	<i>job ID</i>	The given job id can also be a job name or a wildcard pattern
Hold	Job	qhold	-h u <i>job ID</i>	The given job ID can also be a job name or a wildcard pattern.
		qalter	-h u <i>job ID</i>	The given job ID can also be a job name or a wildcard pattern.
Alter	Job		<i>see man page</i>	
Release	Job		-h n <i>job ID</i>	The given job ID can also be a job name or a wildcard pattern.
		qrfs	<i>job ID</i>	The given job ID can also be a job name or a wildcard pattern.
Reschedule	Job		-rj <i>job ID</i>	The given job ID can also be a job name or a wildcard pattern.
Suspend	Job	qmod	-sj <i>job ID</i>	The given job ID can also be a job name or a wildcard pattern.
	Queue		-sq <i>name</i>	
Release	Job		-usj <i>job ID</i>	The given job ID can also be a job name or a wildcard pattern.
	Queue		-usq <i>name</i>	
Clear Error	Job		-cj <i>job ID</i>	The given job ID can also be a job name or a wildcard pattern.
	Queue		-cq <i>name</i>	
Submit	Job	qsub	<i>see documentation</i>	
		qrsh	<i>see documentation</i>	

Adapted from DantT's Cheat Sheet

[Back to User Guide](#)

3 UserGuide How to Compile and Build a MPI Application

[Return to the User Guide]

3.1 Introduction

Message Passing Interface (MPI) is a specification for an API that allows an application running on many processors to communicate with one another irrespective of the number of individual machines on which the program is running. Both point-to-point and collective interprocess communication is supported. MPI is only necessary in programs that need to communicate between active processes.

3.2 Supported Implementations

Currently OpenMPI is the preferred implementation of the MPI-2 standard. That said, we have also done limited testing of the following:

- MPICH1
- MPICH2
- MVAPICH1/2

To ensure that your PATH and LD_LIBRARY_PATH variables are configured for the desired MPI environment, be sure to run the Man:module command.

3.3 Scope

This document will describe how to compile simple example programs. It is beyond the scope to describe how to develop an MPI application.

3.4 MPI Programs

Every MPI program has an initialization and a finalization phase. Just about anything can happen in between including the good, the bad, and the ugly. We have included a simple "Hello World" example for four languages.

3.5 C

```
#include <stdio.h>
#include "mpi.h"
```

```
int main(int argc, char* argv[])
{
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello, world, I am %d of %d\n", rank, size);
    MPI_Finalize();

    return 0;
}
```

3.5.1 OpenMPI

```
export PATH=/opt/openmpi/gnu/bin:$PATH
mpicc -Wl,-rpath=/opt/openmpi/gnu/lib hello_c.c -o hello_c
```

3.5.2 MPICH1

```
export PATH=/opt/mpich1/gnu/bin:$PATH
mpicc hello_c.c -o hello_c
```

3.5.3 MPICH2

```
export PATH=/opt/mpich2/gnu/bin:$PATH
mpicc hello_c.c -o hello_c
```

3.5.4 MVAICH1

```
export PATH=/opt/mvapich1/gnu/bin:$PATH
mpicc -WI,-rpath=/opt/mvapich1/gnu/lib/shared/ hello_c.c -o hello_c
mpirun -np <slot count> -hostfile <host file name> hello_c
```

3.5.5 MVAICH2

```
export PATH=/opt/mvapich2/gnu/bin:$PATH
mpicc -WI,-rpath=/opt/mvapich2/gnu/lib/ hello_c.c -o hello_c
mpiexec -machinefile <host file name> -np <slot count> hello_c
```

3.6 C++

```
#include <iostream.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
    MPI::Init(argc, argv);

    int rank = MPI::COMM_WORLD.Get_rank();
    int size = MPI::COMM_WORLD.Get_size();

    cout << "Hello World! I am " << rank << " of " << size << endl;

    MPI::Finalize();
    return(0);
}
```

3.6.1 OpenMPI

```
export PATH=/opt/openmpi/gnu/bin:$PATH
mpic++ -WI,-rpath=/opt/openmpi/gnu/lib hello_cxx.cc -o hello_cxx
```

3.6.2 MPICH1

```
export PATH=/opt/mpich1/gnu/bin:$PATH
mpicxx hello_cxx.cc -o hello_cxx
```

3.6.3 MPICH2

```
export PATH=/opt/mpich2/gnu/bin:$PATH
mpicxx hello_cxx.cc -o hello_cxx
```

3.6.4 MVAICH1

```
export PATH=/opt/mvapich1/gnu/bin:$PATH
mpicxx -WI,-rpath=/opt/mvapich1/gnu/lib/shared/ hello_cxx.cc -o hello_cxx
mpirun -np <slot count> -hostfile <host file name> hello_cxx
```

3.6.5 MVAICH2

```
export PATH=/opt/mvapich2/gnu/bin:$PATH
mpicxx -WI,-rpath=/opt/mvapich2/gnu/lib/ hello_cxx.cc -o hello_cxx
mpiexec -machinefile <host file name> -np <slot count> hello_cxx
```

3.7 Fortran 77

```
program main
implicit none
include 'mpif.h'
integer ierr, rank, size
```

```

call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
print *, "Hello, world, I am ", rank, " of ", size
call MPI_FINALIZE(ierr)

end

```

3.7.1 OpenMPI

```

export PATH=/opt/openmpi/gnu/bin:$PATH
mpif77 -Wl,-rpath=/opt/openmpi/gnu/lib hello_f77.f -o hello_f77

```

3.7.2 MPICH1

```

export PATH=/opt/mpich1/gnu/bin:$PATH
mpif77 hello_f77.f -o hello_f77

```

3.7.3 MPICH2

```

export PATH=/opt/mpich2/gnu/bin:$PATH
mpif77 hello_f77.f -o hello_f77

```

3.7.4 MVAICH1

```

export PATH=/opt/mvapich1/gnu/bin:$PATH
mpif77 -Wl,-rpath=/opt/mvapich1/gnu/lib/shared/ hello_f77.f -o hello_f77
mpirun -np <slot count> -hostfile <host file name> hello_f77

```

3.7.5 MVAICH2

```

export PATH=/opt/mvapich2/gnu/bin:$PATH
mpif77 -Wl,-rpath=/opt/mvapich2/gnu/lib/ hello_f77.f -o hello_f77
mpiexec -machinefile <host file name> -np <slot count> hello_f77

```

3.8 Fortran 90

```

program main
  use mpi
  implicit none
  integer :: ierr, rank, size

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
  print *, "Hello, world, I am ", rank, " of ", size
  call MPI_FINALIZE(ierr)
end

```

3.8.1 OpenMPI

```

export PATH=/opt/openmpi/gnu/bin:$PATH
mpif90 -Wl,-rpath=/opt/openmpi/gnu/lib hello_f90.f90 -o hello_f90

```

3.8.2 MPICH1

```

export PATH=/opt/mpich1/gnu/bin:$PATH
mpif90 hello_f90.f90 -o hello_f90

```

3.8.3 MPICH2

```

export PATH=/opt/mpich2/gnu/bin:$PATH
mpif90 hello_f90.f90 -o hello_f90

```

3.8.4 MVAPICH1

```
export PATH=/opt/mvapich1/gnu/bin:$PATH
mpif90 -Wl,-rpath=/opt/mvapich1/gnu/lib/shared/ hello_f90.f90 -o hello_f90
mpirun -np <slot count> -hostfile <host file name> hello_f90
```

3.8.5 MVAPICH2

```
export PATH=/opt/mvapich2/gnu/bin:$PATH
mpif90 -Wl,-rpath=/opt/mvapich2/gnu/lib/ hello_f90.f90 -o hello_f90
mpiexec -machinefile <host file name> -np <slot count> hello_f90
```

[Back to User Guide](#)

4 UserGuide How to Submit Jobs to GE

[Return to the User Guide]

4.1 Introduction

Grid Engine Documentation

4.2 Environment Setup

The first thing you must do is connect to a submission host.

set-up your environment to use grid engine. If you are using Bourne-shell:

```
./opt/sge/default/common/settings.sh
```

However, if you are using C-shell then the set-up is slightly different:

```
source /opt/sge/default/common/settings.csh
```

Now you are ready to submit a job using either *qsub* or *qmon*

4.3 Job Submission Possibilities

4.3.1 *qsub* - Submit Job

The *qsub* command submits batch jobs to the Grid Engine queuing system. Grid Engine supports single and multiple-node jobs. The command can be a path to a binary or a script.

Grid Engine comes with several examples in */opt/sge/examples*. We will use those for instructive purposes.

4.3.1.1 Simple Example

Copy */opt/sge/examples/jobs/simple.sh* into a test directory of your choice. This program will print the date and time, sleep for twenty seconds, print the date and time once again, and then exit. From the test directory, submit the program:

```
qsub simple.sh
```

The job will create two files in your home directory: *simple.sh.e<job ID>* for the "error" output and *simple.sh.o<job ID>* for the standard output. If all goes well, the standard output file will contain something like this:

```
Wed Jul 9 17:38:39 EDT 2008
```

```
Wed Jul 9 17:38:59 EDT 2008
```

4.3.1.2 Array Job Example

There are times when you may wish to execute a particular set of operations repeatedly with the same input parameters. Rendering an animated movie frame-by-frame is one example of this usage. Job such as these can be submitted as a single array job rather than one job per frame.

The tasks of an array job are referenced through an array index number. The index range is defined during submission of the array job using a single *qsub* command.

Copy */opt/sge/examples/jobs/step_B_array_submitter.sh* into a test directory of your choice. This program will print the job ID and the task ID for each job in the array. The job ID should remain constant while the task ID will vary depending on the range you provided. From the test directory, submit the program:

```
qsub -t <start index>-<end index> step_B_array_submitter.sh
```

The *-t* option defines the task index range. Choose an index range with at least two possible values. For example, if you choose the range 2 to 10, 2 specifies the lowest index number while 10 is the highest.

You can also specify the increment for the array job by adding a third parameter to the *-t* argument:

```
qsub -t <start index>-<end index> :<step> step_B_array_submitter.sh
```

In this situation, if you choose the range 10 to 20 with a step of 2, 10 specifies the lowest index number while 20 is the highest. Only every second index, the *:2* part of the specification, is used. Thus the array job is made up of 6 tasks with indices 10, 12, 14, 16, 18, and 20.

The job will create two files for each index into your home directory: one for errors and one for standard output. The error files will take the form *step_B_array_submitter.sh.e<job ID>.<task ID>* while the output files will take the form *step_B_array_submitter.sh.o<job ID>.<task ID>*. If all goes well, the each of the standard output files will contain both the job ID and the task ID.

4.3.2 *qsh* - Submit an Interactive X-windows Session

The *qsh* command submits an interactive X-windows session to Grid Engine. An x-terminal (*xterm*) is brought up from the executing machine with the display directed either to the X-server indicated by the `DISPLAY` environment variable, or as specified with the `-display qsh` option. Interactive jobs are not spooled if no resource is available to execute them. They are either dispatched to a suitable machine for execution immediately, or the user submitting the job is notified by *qsh* that appropriate resources to execute the job are not available.

4.3.3 *qlogin* - Submit an Interactive Login Session

The *qlogin* command is similar to *qsh* in that it submits an interactive job to the queueing system. However, it does not open an *xterm* window on the X display, but uses the current terminal for user I/O. Usually, *qlogin* establishes a telnet connection with the remote host, using standard client- and server-side commands. These commands can be configured with the *qlogin* daemon (server-side, Grid Engine *telnetd* if not set, otherwise something like `/usr/sbin/in.telnetd`) and *qlogin* command (client-side, Grid Engine *telnet* if not set, otherwise something like `/usr/bin/telnet`) parameters in the global and local configuration settings. The *qlogin* command is invoked exactly like *qsh* and its jobs can only run on interactive queues. Also, *qlogin* jobs can only be used if the SGE execution daemon is running under the root account.

4.3.4 *qrsh* - Submit an Interactive RSH Session

The *qrsh* command is similar to *qlogin* and *qsh* in that it submits an interactive job to the queueing system. It uses the current terminal for user I/O. Usually, *qrsh* establishes an rsh connection with the remote host. If no command is given to *qrsh*, an *rlogin* session is established. The server-side commands used can be configured with the *rsh_daemon* and *rlogin_daemon* parameters in the global and local configuration settings. Grid Engine *rshd* or *rlogind* is used if the parameters are not set. If the parameters are set, they should point to something like `/usr/sbin/in.rshd` or `/usr/sbin/in.rlogind`. On the client-side, the *rsh_command* and *rlogin_command* parameters can be set in the global and local configuration settings. If they are not set, special Grid Engine *rsh* and *rlogin* binaries are used. The *qrsh* jobs can only run in interactive queues unless the option "-now no" is used.

4.3.5 *qresub* - Submit a Copy of an Existing Job

The *qresub* command allows users to create jobs as copies of existing pending or running jobs. The copied jobs will have exactly the same attributes as the ones from which they were copied, except with a new job ID. The only modification to the copied jobs supported by the *qresub* command is assignment of a hold state using the `?-h?` option.

4.3.6 *qmake* - Distributed Parallel Make

This is a parallel, distributed make utility. Scheduling of the parallel make tasks is done by Grid Engine. The *qmake* utility is based on *gmake* (GNU make).

4.3.7 *qtcsh* - TCSH with Transparent Remote Execution

This is an extension to *tcsh*, popular csh-derivative command line interpreter. It allows the transparent remote execution of commands entered in *qtcsh* and controlled via Grid Engine. It can be used for processing of *tcsh* shell scripts.

4.3.8 *qmon* - Graphical User Interface

The *qmon* utility is an X-Windows OSF/Motif graphical user interface for Grid Engine. It allows administrators and users to monitor and manipulate the system from an X-Window environment. After invoking *qmon*, users/administrators are presented with various dialogues that enable users to accomplish certain tasks (e.g., job submission/monitoring, queue configuration, scheduler configuration, host configuration, user configuration, etc.).

4.3.9 *qalter* - Modify Pending Job

The *qalter* command can be used to change the attributes of pending jobs. For array jobs, modifications only affects pending tasks. Note that TU has implemented a custom wrapper tool (`tug_modify_job_priority.sh`) for modifying job priority.

4.4 Environment Variables

When a job runs, a number of variables are preset into the it's environment:

- ARC
The architecture name of the node on which the job is running. The name is compiled into the `sge_execd` binary.
- SGE_ROOT
The root directory of the grid engine system as set for `sge_execd` before startup, or the default `/usr/SGE`.
- SGE_BINARY_PATH
The directory in which the grid engine system binaries are installed.
- SGE_CELL
The cell in which the job runs.

SGE_JOB_SPOOL_DIR
The directory used by sge_shepherd to store job-related data while the job runs.

SGE_O_HOME
The path to the home directory of the job owner on the host from which the job was submitted.

SGE_O_HOST
The host from which the job was submitted.

SGE_O_LOGNAME
The login name of the job owner on the host from which the job was submitted.

SGE_O_MAIL
The content of the MAIL environment variable in the context of the job submission command.

SGE_O_PATH
The content of the PATH environment variable in the context of the job submission command.

SGE_O_SHELL
The content of the SHELL environment variable in the context of the job submission command.

SGE_O_TZ
The content of the TZ environment variable in the context of the job submission command.

SGE_O_WORKDIR
The working directory of the job submission command.

SGE_CKPT_ENV
The checkpointing environment under which a checkpointing job runs. The checkpointing environment is selected with the qsub -ckpt command.

SGE_CKPT_DIR
The path ckpt_dir of the checkpoint interface. Set only for checkpointing jobs. For more information, see the checkpoint(5) man page.

SGE_STDERR_PATH
The path name of the file to which the standard error stream of the job is diverted. This file is commonly used for enhancing the output with error messages from prolog, epilog, parallel environment start and stop scripts, or checkpointing scripts.

SGE_STDOUT_PATH
The path name of the file to which the standard output stream of the job is diverted. This file is commonly used for enhancing the output with messages from prolog, epilog, parallel environment start and stop scripts, or checkpointing scripts.

SGE_TASK_ID
The task identifier in the array job represented by this task.

ENVIRONMENT
Always set to BATCH. This variable indicates that the script is run in batch mode.

HOME
The user's home directory path from the passwd file.

HOSTNAME
The host name of the node on which the job is running.

JOB_ID
A unique identifier assigned by the sge_qmaster when the job was submitted. The job ID is a decimal integer from 1 through 9,999,999.

JOB_NAME
The job name, which is built from the qsub script filename, a period, and the digits of the job ID. You can override this default with qsub -N.

LOGNAME
The user's login name from the passwd file.

NHOSTS
The number of hosts in use by a parallel job.

NQUEUES
The number of queues that are allocated for the job. This number is always 1 for serial jobs.

NSLOTS
The number of queue slots in use by a parallel job.

PATH
A default shell search path of: /usr/local/bin:/usr/ucb/bin:/usr/bin.

PE
The parallel environment under which the job runs. This variable is for parallel jobs only.

PE_HOSTFILE
The path of a file that contains the definition of the virtual parallel machine that is assigned to a parallel job by the grid engine system. This variable is used for parallel jobs only. See the description of the \$pe_hostfile parameter in sge_pe for details on the format of this file.

QUEUE
The name of the queue in which the job is running.

REQUEST
The request name of the job. The name is either the job script file name or is explicitly assigned to the job by the qsub -N command.

RESTARTED
Indicates whether a checkpointing job was restarted. If set to value 1, the job was interrupted at least once. The job is therefore restarted.

SHELL
The user's login shell from the passwd file. SHELL is not necessarily the shell that is used for the job.

TMPDIR
The absolute path to the job's temporary working directory.

TMP
The same as TMPDIR. This variable is provided for compatibility with NQS.

TZ
The time zone variable imported from sge_execd, if set.

USER
The user's login name from the passwd file.

4.5 Script-Embedded Options (Active Comments)

Grid engine recognizes special comment lines and uses these lines in a special way. By default, the special comment lines are identified by the "#\$" prefix string. This can be redefined using the `-C` option with the `qsub` command. The rest of any line with this prefix is treated as part of the command line argument list of the `qsub` command.

A key feature of these active comments is that they have access to the environmental variables listed previously.

Furthermore, because grid engine essentially ignores the `#!/` directive, it can be useful to force it to use the shell of your choice. If you add the following to your script it will force the system to use a Bourne-shell:

```
# Force sh if not Grid Engine default shell
```

```
#$ -S /bin/sh
```

Similarly, you can force grid engine to use C-shell:

```
# Force csh if not Grid Engine default shell
```

```
#$ -S /bin/csh
```

[Back to User Guide](#)

5 UserGuide How to Submit MPI jobs to GE

[Return to the User Guide]

5.1 Assumptions

- *OpenMPI* is installed in `/opt/openmpi/gnu`.
- *MPICH1* is installed in `/opt/mpich1/gnu`.
- *MPICH2* is installed in `/opt/mpich2/gnu`.
- *MVAPICH1* is installed in `/opt/mvapich1/gnu`.
- *MVAPICH2* is installed in `/opt/mvapich2/gnu`.
- Your grid engine queues and parallel environments are configured properly.
- Ensure that your `PATH` and `LD_LIBRARY_PATH` have the desired MPI environment selected by using the `Man:module` command.

5.2 OpenMPI

5.2.1 Parallel Environment

It is beyond the scope of this document to explain how the parallel environment for *OpenMPI* should be configured. If OpenMPI is configured properly, it can natively support GridEngine without the need for custom `start_proc_args` helper scripts. Having said that, here is an example of a round robin environment:

```
pe_name smallRoundRobin
slots 12
user_lists NONE
xuser_lists NONE
start_proc_args /bin/true
stop_proc_args /bin/true
allocation_rule $round_robin
control_slaves TRUE
job_is_first_task FALSE
urgency_slots avg
```

5.2.2 Wrapper Script

The first thing you must do is create a script to run your program. If the queue to which you are submitting uses `/bin/sh`, then this will work:

```
#!/bin/sh

export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/opt/openmpi/gnu/lib/
export PATH=/opt/openmpi/gnu/bin:$PATH
```

```
mpirun -np ${PROCS} <program name>
However, if you are using /bin/csh, then the script is slightly different:
```

```
#!/bin/csh

setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:/opt/openmpi/gnu/lib/
setenv PATH /opt/openmpi/gnu/bin:${PATH}
```

```
mpirun -np ${NSLOTS} <program name>
```

5.2.3 Job Submission

You are now able to submit your newly created script to grid engine:

```
qsub -pe <parallel environment name> <slot count> <script name>
```

5.2.4 Example

OpenMPI comes with several examples in `/opt/openmpi/gnu/examples/`. Any one of these examples should work using the process in this section. Please refer to the MPI compilation discussion for building one or more of these applications.

5.3 MPICH1

5.3.1 Parallel Environment

It is beyond the scope of this document to explain how the parallel environment for *MPICH1* should be configured. Having said that, here is an example:

```
pe_name mpi
slots 999
user_lists NONE
xuser_lists NONE
start_proc_args /opt/sge/mpi/startmpi.sh $pe_hostfile
stop_proc_args /opt/sge/mpi/stopmpi.sh
allocation_rule $round_robin
control_slaves FALSE
job_is_first_task TRUE
urgency_slots min
```

5.3.2 Wrapper Script

The first thing you must do is create a script to run your program. If the queue to which you are submitting uses `/bin/sh`, then this will work:

```
#!/bin/sh

export MPICH_HOME=/opt/mpich1/gnu
export PATH=${TMPDIR}:${MPICH_HOME}/bin:$PATH

mpirun -np $NSLOTS -machinefile $TMPDIR/machines <program name>
However, if you are using /bin/csh, then the script is slightly different:
```

```
#!/bin/csh

setenv MPICH_HOME /opt/mpich1/gnu
setenv PATH ${TMPDIR}:${MPICH_HOME}/bin:${PATH}

mpirun -np $NSLOTS -machinefile $TMPDIR/machines <program name>
```

5.3.3 Job Submission

You are now able to submit your newly created script to grid engine:

```
qsub -pe <parallel environment name> <slot count> <script name>
```

5.4 MPICH2

5.4.1 Parallel Environment

It is beyond the scope of this document to explain how the parallel environment for *MPICH2* should be configured. Having said that, here is an example:

```
pe_name mpich2_smpd_rsh
slots 999
user_lists NONE
xuser_lists NONE
start_proc_args /opt/sge/mpich2_smpd_rsh/startmpich2.sh -catch_rsh \
                $pe_hostfile
stop_proc_args /opt/sge/mpich2_smpd_rsh/stopmpich2.sh
allocation_rule $round_robin
control_slaves FALSE
job_is_first_task TRUE

urgency_slots min
```

5.4.2 Wrapper Script

The first thing you must do is create a script to run your program. If the queue to which you are submitting uses `/bin/sh`, then this will work:

```
#!/bin/sh

export MPICH_HOME=/opt/mpich2/gnu
export PATH=${TMPDIR}:${MPICH_HOME}/bin:$PATH

mpiexec -rsh -nopm -np $NSLOTS -machinefile $TMPDIR/machines <program name>
However, if you are using /bin/csh, then the script is slightly different:
```

```
#!/bin/csh
```

```
setenv MPICH_HOME /opt/mpich2/gnu  
setenv PATH ${TMPDIR}:${MPICH_HOME}/bin:${PATH}
```

```
mpiexec -rsh -nopr -np $NSLOTS -machinefile $TMPDIR/machines <program name>
```

5.4.3 Job Submission

You are now able to submit your newly created script to grid engine:

```
qsub -pe <parallel environment name> <slot count> <script name>
```

5.5 MVAPICH1

The main difference between *MVAPICH1* and *MPICH1* is its capability to use infiniband as well as standard TCP/IP based networks. Consequently, the MPI jobs should be very similar to those documented previously. However these have not been tested.

Note: these scripts have not been validated because working infiniband servers were not available at the time of this writing.

5.5.1 Infiniband Resources

You should either create a special queue or a resource flag for the machines that are infiniband capable. We will assume that you have created a queue for this document.

5.5.2 Parallel Environment

It is beyond the scope of this document to explain how the parallel environment for *MPICH1* should be configured. Having said that, here is an example:

```
pe_name mvapich1  
slots 999  
user_lists NONE  
xuser_lists NONE  
start_proc_args /opt/sge/mpi/startmvapich.sh $pe_hostfile  
stop_proc_args /opt/sge/mpi/stopmvapich.sh  
allocation_rule $round_robin  
control_slaves FALSE  
job_is_first_task TRUE  
urgency_slots min
```

5.5.3 Wrapper Script

The first thing you must do is create a script to run your program. If the queue to which you are submitting uses `/bin/sh`, then this will work:

```
#!/bin/sh
```

```
export MVAPICH_HOME=/opt/mvapich1/gnu  
export PATH=${TMPDIR}:${MVAPICH_HOME}/bin:$PATH
```

```
mpirun -np $NSLOTS -machinefile $TMPDIR/machines <program name>  
However, if you are using /bin/csh, then the script is slightly different:
```

```
#!/bin/csh
```

```
setenv MVAPICH_HOME /opt/mvapich1/gnu  
setenv PATH ${TMPDIR}:${MVAPICH_HOME}/bin:${PATH}
```

```
mpirun -np $NSLOTS -machinefile $TMPDIR/machines <program name>
```

5.5.4 Job Submission

You are now able to submit your newly created script to grid engine:

```
qsub -q <infiniband queue>-pe <parallel environment name> \  
<slot count> <script name>
```

5.6 MVAPICH2

The main difference between *MVAPICH2* and *MPICH2* is its capability to use infiniband as well as standard TCP/IP based networks. Consequently, the MPI jobs should be very similar to those documented previously. However these have not been tested.

Note: these scripts have not been validated because working infiniband servers were not available at the time of this writing.

5.6.1 Infiniband Resources

You should either create a special queue or a resource flag for the machines that are infiniband capable. We will assume that you have created a queue for this document.

5.6.2 Parallel Environment

It is beyond the scope of this document to explain how the parallel environment for *MVAPICH2* should be configured. Having said that, here is an example:

```
pe_name mvapich2
slots 999
user_lists NONE
xuser_lists NONE
start_proc_args /opt/sge/mvapich2/startmvapich2.sh -catch_rsh \
    $pe_hostfile
stop_proc_args /opt/sge/mvapich2/stopmvapich2.sh
allocation_rule $round_robin
control_slaves FALSE
job_is_first_task TRUE

urgency_slots min
```

5.6.3 Wrapper Script

The first thing you must do is create a script to run your program. If the queue to which you are submitting uses `/bin/sh`, then this will work:

```
#!/bin/sh

export MVAPICH_HOME=/opt/mvapich2/gnu
export PATH=${TMPDIR}:${MVAPICH_HOME}/bin:$PATH

mpiexec -machinefile $TMPDIR/machines -np $NSLOTS <program name>
However, if you are using /bin/csh, then the script is slightly different:
```

```
#!/bin/csh

setenv MVAPICH_HOME /opt/mvapich2/gnu
setenv PATH ${TMPDIR}:${MVAPICH_HOME}/bin:${PATH}

mpiexec -machinefile $TMPDIR/machines -np $NSLOTS <program name>
```

5.6.4 Job Submission

You are now able to submit your newly created script to grid engine:

```
qsub -q <infiniband queue> -pe <parallel environment name> \
    <slot count> <script name>
```

[Back to User Guide](#)

6 UserGuide How to View Status and Delete Jobs from GE

[Return to the User Guide]

6.1 Job Status

- All jobs for all queues:
qstat -f
- All jobs for all queues with additional resources:
qstat -F [optional resource name list]
- By job ID(s):
qstat -j job ID list
- By job name(s):
qstat -j job name list
- Using wildcards:
qstat -j wildcard set

If you specify any jobs in a list and they do not exist, the command will exit before attempting to query the next item.

6.1.1 Resource Restriction

The resources required by the jobs or granted by the queues on which information is requested can be specified either alone or with other options:

```
qstat -l resource=[value]
```

6.2 Suspending a Job

```
qmod -rj job_id
```

6.3 Applying a User Hold on a Job

A hold can be applied to a job so that one or more pending jobs will not be scheduled. As long as any type of hold is assigned to a job, the job is not eligible for scheduling.

- By job ID(s):
qhold -h u job ID list
- By job name(s):
qhold -h u job name list
- Using wildcards:
qhold -h u wildcard set

6.4 Deleting a Job

- By job ID(s):
qdel job ID list
- By job name(s):
qdel job name list
- Using wildcards:
qdel wildcard set

[Back to User Guide](#)

7 Tool

7.1 Overview

The "module" command toolset is a set of tools that allows for the configuration of an environment for a specific application. The name "module" is somewhat misleading, it is not a tool for manipulating kernel modules or binary object modules, rather it allows for sets of configuration to be created such that an environment or shell can be created with the specific needs of a particular application in mind. UniCluster 4.0 comes with module configuration for several applications already installed

7.2 Example

One instance where the module command is commonly used is with MPI libraries. There are a variety of MPI libraries that each have their own set of libraries, compilers and executables.

To see the list of available module configurations run "module avail":

```
[root@installer4 etc]# module avail

----- /usr/share/Modules/modulefiles -----
dot                module-cvs                mpi/openmpi-interconnects-gnu
hdf5/hdf5-mpich1-gnu  module-info              null
hdf5/hdf5-openmpi-gnu  modules                  use.own
linpack/linpack-mpich1-gnu  mpi/mpich1-ethernet-gnu
linpack/linpack-openmpi-gnu  mpi/mpich2-ethernet-gnu

----- /usr/share/Modules/modulefiles -----
dot                module-cvs                mpi/openmpi-interconnects-gnu
hdf5/hdf5-mpich1-gnu  module-info              null
hdf5/hdf5-openmpi-gnu  modules                  use.own
linpack/linpack-mpich1-gnu  mpi/mpich1-ethernet-gnu
linpack/linpack-openmpi-gnu  mpi/mpich2-ethernet-gnu
```

Here is an example of loading a module under bash:

```
% which mpicc
no mpicc in ...
% module load mpi/mpich2-ethernet-gnu
% which mpicc
/opt/mpich2/gnu/bin/mpicc
```

Now we'll switch to a different version of the module

```
% module switch mpi/mpich2-ethernet-gnu mpi/mpich1-ethernet-gnu
% which mpicc
/opt/mpich1/gnu/bin/mpicc
```

7.3 Usage

The "module" toolset is installed by default on the cluster as part of the UniCluster 4.0 framework. This help list can be displayed with "module --help":

```
[root@installer uc4]# module --help

Modules Release 3.2.5 2007-02-14 (Copyright GNU GPL v2 1991):

Usage: module [ switches ] [ subcommand ] [subcommand-args ]

Switches:
  -H|--help                this usage info
  -V|--version             modules version & configuration options
  -f|--force               force active dependency resolution
  -t|--terse               terse      format avail and list format
  -l|--long                long      format avail and list format
  -h|--human               readable format avail and list format
  -v|--verbose             enable  verbose messages
  -s|--silent              disable verbose messages
  -c|--create              create caches for avail and apropos
  -i|--icase               case insensitive
  -u|--userlvl <lvl>     set user level to (nov[ice],exp[ert],adv[anced])

Available SubCommands and Args:
+ add|load                modulefile [modulefile ...]
+ rm|unload               modulefile [modulefile ...]
+ switch|swap             [modulefile1] modulefile2
+ display|show            modulefile [modulefile ...]
+ avail                   [modulefile [modulefile ...]]
+ use [-a|--append]      dir [dir ...]
+ unuse                   dir [dir ...]
```

```
+ update
+ refresh
+ purge
+ list
+ clear
+ help          [modulefile [modulefile ...]]
+ whatis       [modulefile [modulefile ...]]
+ apropos|keyword string
+ initadd      modulefile [modulefile ...]
+ initprepend  modulefile [modulefile ...]
+ initrm       modulefile [modulefile ...]
+ initswitch   modulefile1 modulefile2
+ initlist
+ initclear
```

[Back to User Guide](#)