

TCC SGI Altix ICE Cluster User's Guide [DRAFT]



Bryan Hughes <khan@nmt.edu>

16 Feb 2011



Table of Contents

1. Introduction	1
2. Hardware Overview	1
3. Software Overview	2
4. Access	2
4.1. Requesting Time On The Cluster	2
4.2. Accounts	2
4.3. Login	2
5. Requesting Resources	2
6. Software	3
6.1. Selecting Software Using Module	3
6.2. MPI Libraries	4
7. E-Mail	7
8. Monitoring	7
8.1. Monitoring Job Status	7
8.2. Monitoring Cluster Status	7
9. Examples	8
9.1. MPI with MVAPICH-2 - Interactive Batch Session - Walkthrough	8

1. Introduction

SGI Altix ICE Cluster is a SGI Altix ICE integrated compute environment for high performance computing.

2. Hardware Overview

- 22 Compute Nodes
- Dual Four-Core Intel Xeon X5365 Processors Per Node (8 Logical Processors)
 - CPU Speed: 3.00 GHz
 - Bus Speed: 1333 MHz
 - L2 Cache: 8MB
 - Arch: x86_64
- 16 GB Memory per node

- 4x DDR Infiniband Interconnect.
- 1 TB User File Storage (1 GB User Quota)
- 8.8 TB Scratch Temporary File Storage

3. Software Overview

Operating System

SUSE Linux Enterprise Server 10.1 (x86_64)

MPI

mvapich2-1.0-1

mvapich-0.9.9

openmpi-1.2-2

Compilers

Intel 10.1

GCC 4.1.2

Libraries

Intel Math Kernel Library 10.0 Update 3

4. Access

4.1. Requesting Time On The Cluster

4.2. Accounts

SGI Altix ICE Cluster accounts are completely separate from your TCC account. User ID, password, and file storage are not shared between accounts. You can transfer data between your TCC account and SGI Altix ICE Cluster account through scp or sftp described below.

Your SGI Altix ICE Cluster account has 1 GB of user space on `/home/`. There is 8.8 TB of shared temporary scratch space on `/scratch/`

4.3. Login

Login using secure-shell (ssh):

```
ssh [user]@icel.nmt.edu
```

Transfer files to the Linux cluster from your home machine or TCC account using secure copy(scp) or secure ftp (sftp):

```
scp [file] [user]@icel.nmt.edu:[dest]
```

```
sftp [user]@icel.nmt.edu
```

5. Requesting Resources

Warning

Do not run jobs on service0. Please use it for compilation and small debugging runs only.

Resources are requested through qsub.

A required option for all resource requests is walltime. By default all jobs have default walltime of 0 minutes. You can specify a walltime for your job with a -l option to qsub in the form of -l walltime=1:00 for one minute, or -l walltime=1:00:00 for one hour. There is currently no upper limit for walltime, the system is in place to help the scheduler.

6. Software

6.1. Selecting Software Using Module

The cluster has several software packages that serve the same purpose and have the same executable name. To ensure the software package you want to use is being executed when you run the command, you will need to set up your shell environment to prefer the package you want. This is done through the Modules package and the *module* command.

Using The Module Command

Module Command

If the *module* command is not found, you need to manually source its configuration by entering :

```
#For Bash enter:
source /etc/profile.d/modules.sh
#For CSH enter
source /etc/profile.d/modules.csh
```

Listing Modules

module avail

The *avail* subcommand lists the modules that are available to load into your environment.

```
khan@service0:~> module avail
----- /usr/share/modules -----
3.1.6                modulefiles/mvapich_gcc
modulefiles/dot      modulefiles/mvapich_intel
modulefiles/module-cvs  modulefiles/null
modulefiles/module-info  modulefiles/openmpi_gcc
modulefiles/modules    modulefiles/openmpi_intel
modulefiles/mvapich2_gcc  modulefiles/perfcatcher
modulefiles/mvapich2_intel  modulefiles/use.own

----- /usr/share/modules/modulefiles -----
dot                mvapich2_intel  openmpi_intel
module-cvs         mvapich_gcc    perfcatcher
module-info       mvapich_intel  use.own
modules           null
mvapich2_gcc     openmpi_gcc

----- /opt/intel/intel_modules -----
cc/10.1.015  cce/10.1.015  fc/10.1.015  mpi/3.1.026
```

module list

The *list* subcommand lists the modules you have currently loaded.

```
khan@service0:~> module list
1) mvapich2_gcc
```

Loading Modules

module load *modulename*

module load sets up software into your environment.

```
khan@service0:~> module load mvapich2_gcc
khan@service0:~> module list
1) mvapich2_gcc
khan@service0:~> which mpicc
/usr/mpi/mvapich2-1.0-1/gcc/bin/mpicc
```

Unloading Modules

module unload *modulename*

module unload removes that software from your environment.

```
khan@service0:~> module list
1) mvapich2_gcc
khan@service0:~> module unload mvapich2_gcc
khan@service0:~> which mpicc
which: no mpicc in (/usr/local/bin:/usr/bin:...)
```

Note

If you want to have these modules be available to you each time you log in, place the commands to load those modules in a file called `.bashrc` in your home folder.

6.2. MPI Libraries

6.2.1. MPT: SGI Message Passing Toolkit

Note

MPT is the default MPI installation. It is available without using *module*

Message Passing Toolkit (MPT) is a software package that supports interprocess data exchange for applications that use concurrent, cooperating processes on a single host or on multiple hosts. Data exchange is done through message passing, which is the use of library calls to request data delivery from one process to another or between groups of processes.

For more information on MPT, see SGI TechPub: 007-3773-007 [<http://techpubs.sgi.com/library/tpl/cgi-bin/download.cgi?docnumber=007-3773-007>]

The MPT package contains the following components and the appropriate accompanying documentation:

- Message Passing Interface (MPI). MPI is a standard specification for a message passing interface, allowing portable message passing programs in Fortran and C languages.
- the SHMEM programming model. The SHMEM programming model is a distributed, shared-memory model that consists of a set of SGI-proprietary message-passing library routines. These routines help distributed applications efficiently transfer data between cooperating processes. The model is based on multiple processes having separate address spaces, with the ability for one process to access data

in another process' address space without interrupting the other process. The SHMEM programming model is not a standard like MPI, so SHMEM applications developed on other vendors' hardware might or might not work with the SGI SHMEM implementation.

Procedure 1. Compiling and Linking MPI Programs with MPT

1. To compile using GNU compilers, choose one of the following commands
 - **g++ -o myprog myprog.cpp -lmpi++ -lmpi**
 - **gcc -o myprog myprog.c -lmpi**
 - **gfortan -I/usr/include -o myprog myprog.f -lmpi**
2. To compile programs with the Intel compiler, use the following commands:
 - **ifort -o myprog myprog.f -lmpi**
 - **icc -o myprog myprog.c -lmpi**

Procedure 2. Running MPT MPI Jobs using Portable Batch System (PBS)

1. Schedule a session with PBS using *qsub*.
2. Each MPI application is executed with the *mpiexec* command that is delivered with the PBS Pro software packages. This is a wrapper script that assembles the correct host list and corresponding *mpirun* command before executing the assembled *mpirun* command. The basic syntax is, as follows:

```
mpiexec -n P ./a.out
```

where P is the total number of MPI processes in the application. This syntax applies whether running on a single host or a clustered system. See the *mpiexec(8)* man page for more details.

6.2.2. MVAPICH2: MPI over InfiniBand

MVAPICH is open source software developed largely by the Network-Based Computing Laboratory (NBCL) at Ohio State University. MVAPICH develops the Message Passing Interface (MPI) style of process-to-process communications for computing systems employing Infiniband and other Remote Direct Memory Access (RDMA) interconnects.

For more descriptions including the MVAPICH User Guide and other MVAPICH publications, see <http://mvapich.cse.ohio-state.edu>.

MVAPICH applications use the Infiniband network of SGI Altix ICE 8200 systems for interprocess RDMA communications. SGI Altix ICE 8200 systems are configured with two Infiniband fabrics, designated as *ib0* and *ib1*. In order to maximize performance, SGI advises that the *ib0* fabric be used for all MPI traffic, including MVAPICH MPI. The *ib1* fabric is reserved for storage related traffic. The default configuration for MVAPICH MPI is to use only the *ib0* fabric.

Procedure 3. Compiling and Linking MPI Programs with MVAPICH2

1. To compile using GNU compilers
 - a. Load the *mvapich2 gcc* module
 - **module load mvapich2_gcc**
 - b. choose one of the following commands
 - **mpicxx -o myprog myprog.cpp**

- **mpicc -o myprog myprog.c**
 - **mpif77 -o myprog myprog.f**
 - **mpif90 -o myprog myprog.f**
2. To compile using Intel compilers
 - a. Load the mvapich2 intel and intel compiler modules
 - **module load mvapich2_intel**
 - **module load module load cce/10.1.015**
 - b. Choose one of the following compiler commands
 - **mpicxx -o myprog myprog.cpp**
 - **mpicc -o myprog myprog.c**
 - **mpif77 -o myprog myprog.f**
 - **mpif90 -o myprog myprog.f**

Procedure 4. Running MVAPICH2 MPI Jobs using Portable Batch System (PBS)

1. First configure mpd, create the \$HOME/.mpd.conf file with perms 0x600

```
# cat $HOME/.mpd.conf
MPD_SECRETWORD=secretword
```

Change "secretword" to something secret, it is your "password" for mpd.

2. Schedule a session with PBS using *qsub*.
3. Load the mvapich2 module you used to compile:

- **module load mvapich2_gcc**
- **module load mvapich2_intel**

4. Boot the MPD multiprocessing daemons.

mpdboot starts MPDs on the nodes you have access to. These MPDs make the nodes into a "virtual machine" that can run MPI programs. When you run an MPI program under mvapich2, requests are sent to MPD daemons to start up copies of the program.

mpdboot -n P -f \$PBS_NODEFILE

Where P is the number of nodes requested from PBS.

5. Launch the program with *mpiexec*

mpiexec -np P ./a.out

Where P is the total number of MPI processes in the application.

6. Clean up the mpi environment.

mpdallexit

6.2.3. OpenMPI

7. E-Mail

The batch system will notify you about your jobs via email to your local account on service0. Look at the manual page for `qsub`, specifically options `-M` and `-m` for more email options.

By default mail will be stored locally on service0. You can either read mail locally or forward it to another system.

Forwarding Mail

To forward mail to your TCC, department, or other email account use a `~/ .forward` file. Place addresses to forward to on separate lines. If you would like to keep a local copy of the mail insert a backslash followed by your local username on the last line of the file.

Example `~/ .forward` file that forwards to the TCC and stores a copy locally.

```
khan@nmt .edu
\khan
```

Checking Mail Locally

Use `mutt` to check your mail locally on service0. It is already configured to read from the local spool.

8. Monitoring

8.1. Monitoring Job Status

`qstat`

`qstat [jobid]`

The `qstat` utility allows users to display the status of jobs and list the batch jobs in queues.

The operands of the `qstat` utility may be either job identifiers, queues (specified as destination identifiers), or batch server names.

The other options of the `qstat` utility allow the user to control the amount of information displayed and the format in which it is displayed.

The `-f` option allows users to request a "full" display of job, queue, or server information.

`tracejob`

`tracejob jobid`

PBS includes the `tracejob` utility to extract daemon/service logfile messages for a particular job (from all log files available on the local host) and print them sorted into chronological order.

Note

Note that the third column of the display contains a single letter (S, M, A, or L) indicating the source of the log message (Server, MOM, Accounting, or scheduler log files).

8.2. Monitoring Cluster Status

`pbsnodes`

The `pbsnodes` command is used to query the status of hosts.

`pbsnodes -l`

pbsnodes -l will list the nodes that are currently down.

pbsnodes -a

pbsnodes -a will list extended information on each node. Such as, node state, currently running jobs, and resources.

Ganglia

Ganglia web monitoring - <http://ice1-admin.nmt.edu/ganglia> - still firewalled.

9. Examples

9.1. MPI with MVAPICH-2 - Interactive Batch Session - Walkthrough

Procedure 5. MVAPICH2 MPI Hello World

1. Create `hello.c`

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

int main(int argc, char **argv){

    int myRank, numProcs;
    char processorName[MPI_MAX_PROCESSOR_NAME];
    int nameLen;

    MPI_Init(&argc,&argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
    MPI_Comm_size(MPI_COMM_WORLD, &numProcs);
    MPI_Get_processor_name(processorName,&nameLen);
    printf("Hello World! From Process: %d/%d on %s\n",
           myRank+1,numProcs,processorName);

    MPI_Finalize();

    return 0;
}
```

2. Load `mvapich2_gcc` module for compilation.

```
module load mvapich2_gcc
```

3. Compile `hello.c` with `mpicc`

```
mpicc hello.c -o hello
```

4. Request resources from the batch system to run the application.

Here we will be requesting an interactive batch session on 2 nodes each using all 8 processors (16 processors total).

```
qsub -I -l walltime=1:00:00,select=2:ncpus=8:mpiprocs=8
```

```
khan@service0:~> qsub -I -l walltime=1:00:00,select=2:ncpus=8:mpiprocs=8
qsub: waiting for job 100.service0-ib0 to start
```

```
qsub: job 100.service0-ib0 ready
khan@r1i0n0:~>
```

5. Reload mvapich2_gcc module for execution.

The request to the batch manager established a new shell session. We need to reload the environment with *module*.

```
module load mvapich2_gcc
```

6. Boot the MPD multiprocessing daemons.

mpdboot starts MPDs on the nodes you have access to. These MPDs make the nodes into a "virtual machine" that can run MPI programs. When you run an MPI program under mvapich2, requests are sent to MPD daemons to start up copies of the program.

Here we boot up on 10 nodes, and specify the nodes the batch system has allocated us.

```
mpdboot -n 2 -f $PBS_NODEFILE
```

7. Launch the hello program.

Now that the environment is set up, the mpi program can run. *mpiexec* will find the mpd environment and pass processes across it. Here we are launching 10 processes, one for each node.

```
mpiexec -np 16 ./hello
```

```
khan@r1i0n0:~> mpiexec -np 16 ./hello
Hello World! From Process: 4/16 on r1i0n1
Hello World! From Process: 2/16 on r1i0n1
Hello World! From Process: 3/16 on r1i0n0
Hello World! From Process: 6/16 on r1i0n1
Hello World! From Process: 13/16 on r1i0n0
Hello World! From Process: 7/16 on r1i0n0
Hello World! From Process: 1/16 on r1i0n0
Hello World! From Process: 9/16 on r1i0n0
Hello World! From Process: 8/16 on r1i0n1
Hello World! From Process: 16/16 on r1i0n1
Hello World! From Process: 11/16 on r1i0n0
Hello World! From Process: 10/16 on r1i0n1
Hello World! From Process: 12/16 on r1i0n1
Hello World! From Process: 15/16 on r1i0n0
Hello World! From Process: 14/16 on r1i0n1
Hello World! From Process: 5/16 on r1i0n0
```

8. Shutdown the MPI environment.

The mpd processes need to be properly shutdown, if you leave the session and there are mpd processes still running, they can prevent another *mpdboot* from working properly.

```
mpdallexit
```

9. Exit the batch session.

Simply logout of the shell given by the batch manager.

```
exit
```

```
khan@r1i0n0:~> exit
logout
```

```
qsub: job 100.service0-ib0 completed  
khan@service0:~>
```