# Using the Linux Cluster Lab

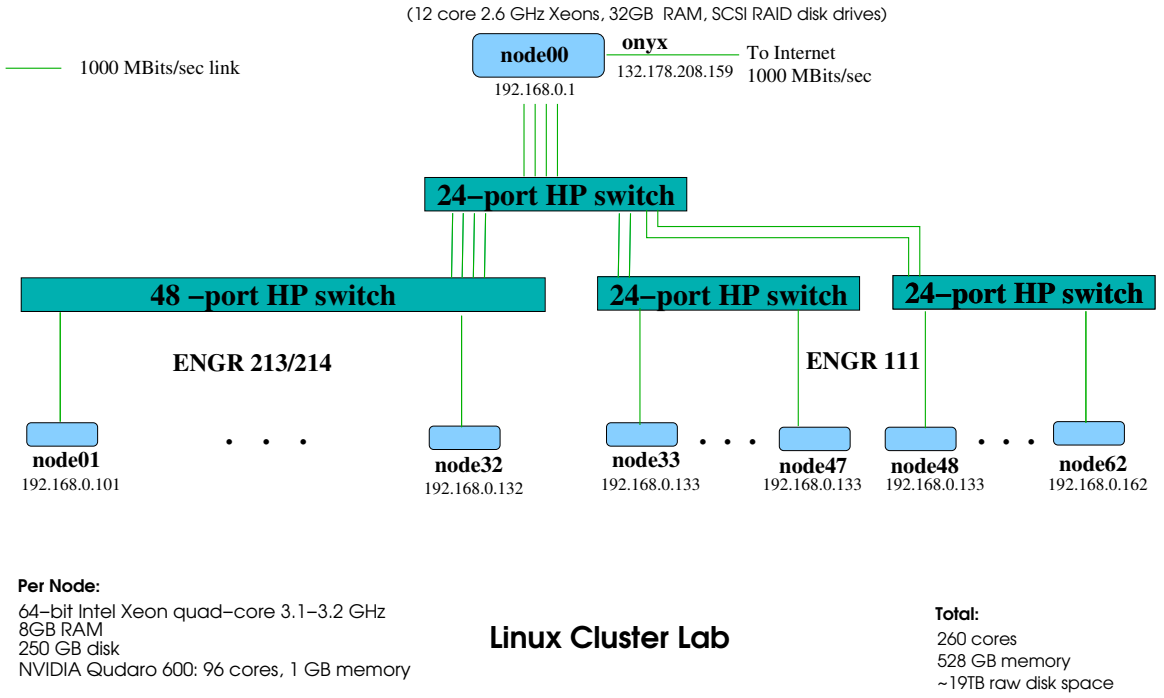
*Amit Jain*
*Department of Computer Science*
*Boise State University*
*September 8, 2016*

## Contents

# 1 Cluster Configuration

The Linux Cluster Labin ET213-214 and ENGR 111 has 62 nodes, named `node01`, ..., `node62`, which are connected with a private Gigabit Ethernet switch. The machines, named `node01`, ..., `node62` are often called as **compute nodes** or simply **nodes**.

The machine `onyx.boisestate.edu` is the **master node** and the only one that is on the public Internet. The master node `onyx` is connected to the private switch with the name `node00`. The figure below shows the layout of the Linux Cluster Lab.




*To use the cluster you must login to onyx.* Your home directory resides on onyx, which is the file server for the lab. Thus you have the same home directory on all machines in the lab.

# 2 Getting Started

## 2.1 Setup

Your home directory is shared among all nodes of the cluster. The master node onyx.boisestate.eduacts as a network file server for the home directories. To get setup, follow the steps shown below:

- Login to the master node, onyx, using the command `ssh -Y onyx`.
- **SSH setup**. Type in the following command to setup password less login in the lab from any machine to another machine.

`ssh-setup`

You only need to run the above command once and from now on you do not need a password to log in to onyx or any other node from a node in the lab

## 2.2 Accessing the cluster from a remote system

The following are some common commands for accessing the master node of the cluster from a remote site.

- `ssh <machinename>` starts a secure remote login session on the machine `<machinename>`.
- `ssh <machinename> <command>` runs the command remotely on the machine called machinename and redirects the output to your terminal.
- `scp` lets you copy files to remote machines.

Note that the letter `s` stands for secure. All these commands encrypt your data on the network to protect against eavesdroppers. These commands replace the unsecure equivalents (`rlogin, rsh, rcp`) that have been used commonly in the past for accessing remote machines.

The command `scp` also has an option `-r` that lets you copy directories. In addition, you can specify a directory to copy to on the remote machine. Remember that the pathnames are relative to your home directory on the remote machine. For example to copy recursively a directory named `prog1` on your local system to the master node `onyx.boisestate.edu`, use the following command:

```
scp -r prog1/ onyx.boisestate.edu:cs430/prog1
```

## 2.3 Useful cluster commands

### 2.3.1 Cluster monitoring

The following list is a set of useful cluster commands that are available on the cluster lab. They can be run from the master node in the cluster.

- `cchk` Check the machines on the cluster by pinging the machines. Quick way to check if any machines are down.
- `cdate` Check the current time and date on each node of the cluster. The master node is a NTP time server (and also a client to another trusted source of accurate time). All the other nodes in the cluster are NTP time service clients of the master node.
- `cmips` Check the total computing power of the cluster. The computing power is reported in MIPS (using the bogoMIPS value reported by the kernel).
- `cfree` Reports memory usage across the cluster.
- `cdisks` Reports temporary disk space usage across the cluster.
- `ctemp` Reports CPU temperatures across the cluster. May not be available on all clusters.

### 2.3.2 Parallel Shell

The cluster comes with a simple parallel shell named `pdsh`. The `pdsh` shell is handy for running commands across the cluster. There is man page that describes the capabilities of `pdsh` is detail. One of the useful features is the capability of specifying all or a subset of the cluster. For example:

`pdsh -a <command>` targets the `<command>` to all nodes of the cluster, including the master.

`pdsh -a -x node00<command>` targets the `<command>` to all nodes of the cluster except the master.

`pdsh node[01-08] <command>` targets the `<command>` to the 8 nodes of the cluster named `node01, node02, ..., node08`.

Another utility that is useful for formatting the output of `pdsh` is `dshbak`. Here we will show some handy uses of `pdsh`.

- Show the current date and time on all nodes of the cluster.
  `pdsh -a date`
- Show the current load and system uptime for all nodes of the cluster.
  `pdsh -a uptime`
- Show the version of the Operating System on all nodes.
  `pdsh -a cat /etc/redhat-release`
- Check who is logged in the MetaGeek lab!
  `pdsh -w node[01-32] who`
- Show all process that have the substring `pbs` on the cluster. These will be the PBS servers running on each node.
  `pdsh -a ps augx | grep pbs | grep -v grep`
- The utility `dshbak` formats the output from `pdsh` by consolidating the output from each node. The option `-c` shows identical output from different nodes just once. Try the following commands.

  ```
  pdsh -w node[01-32] who | dshbak
  pdsh -w node[01-32] who | dshbak -c
  pdsh -a date | dshbak -c
  ```

# 3 Using Swarm to run multiple sequential programs

Using the swatm utility, you can run mulitple sequential programs on multiple nodes of the cluster. This helps you improve the throughput but doesn't speedup any particular instance of your program.

List the sequential programs that you want to run in a text file (one per line). For example, suppose we have the following commands in the file named `myjobs`

```
program1; program1
```

```
program2
program3
```

Then submit your jobs with the following command

```
swarm -f myjobs
```

which creates PBS jobs and submits them. By default, swarm will run two processes per node and create one PBS job per node. So the output from the two commands will show up in the output file corresponding to the PBS job generated by swarm.

If you like to get email when your jobs are done, then use:

```
swarm -f myjobs -m e
```

You may want to set up a .forward file on master node onyxso that the email notification from PBS gets routed to your favorite mail address.

If your program is I/O or memory intensive, then you may want to tell swarm to only run one process per node. This can be done as follows:

```
swarm -f myjobs -n 1 -m e
```

Check the status of your jobs with the following command.

```
qstat -a
```

To kill a job, use

```
qdel <job#>.beowulf
```

To kill all your jobs, use

```
qdelall
```

You should use the above command with caution! For more information about swarm, see the man page: `man swarm`

# 4 Running parallel programs interactively

To run a parallel program interactively on the cluster requires two steps.

- Acquire desired number of compute nodes via the Portable Batch System that controls compute node allocation to users.
- Run the parallel program using `mpiexec`.

## 4.1 Acquiring nodes from the PBS system

The cluster uses the TORQUE system to manage the resources effectively. TORQUE stands for Terascale Open-Source Resource and QUEue Manager. It is an Open Source distributed resource manager originally based on OpenPBS, the Portable Batch System (PBS). We will simply call the resource manager the PBS system.

To run a parallel program, the user needs to request nodes from the PBS system. The master node is a shared resource and is always allocated to the user. The compute nodes are allocated in an exclusive mode. Currently there is a time limit of one hour for the use of compute nodes at a time in the interactive mode.

To check the status of the nodes in the PBS system:

- `cnodes` shows the number of allocatable nodes that are currently available.
- `qstat -a` gives a list of all jobs running on the cluster with the number of nodes allocated per job.
- `qstat -n` gives a list of all jobs running on the cluster along with the nodes allocated to the jobs.
- `pbsnode -a` gives a list of all the nodes with their attributes, `pbsnode -l down` gives a list of nodes that are down and `pbsnode -l up` gives a list of nodes that are up.

To request `n` nodes, use the command `pbsget` on the master node. Here is a sample session.

```
amit@onyx ~]$ pbsget -8
#########################################################################
Allocate cluster nodes via PBS for running interactive parallel jobs.
#########################################################################
Trying for 8 nodes
**************************************************
 Scheduling an interactive cluster session with PBS.
 Please end session by typing in exit.
 Use qstat -n to see nodes allocated by PBS.

 You may now run your mpi programs. They will automatically use
 only the nodes allocated by PBS.
 For running MPI programs use the following commands:
    mpiexec [options]  <program> [<prog args>]
**************************************************

qsub: waiting for job 2608.onyx.boisestate.edu to start
qsub: job 2608.onyx.boisestate.edu ready

[amit@node14 PBS ~]:qstat -n
onyx.boisestate.edu:
                                                                  Req'd       Elap
Job ID       Username    Queue    Jobname    SessID  NDS   TSK Memory   Time    S   Time
---------------------- ----------- -------- ---------------- ------ --------- - ---------
2608.onyx.boisestate.e  amit    interact STDIN  0   8 8:node   --     00:30:00 R  00:00:04
   node14/0+node21/0+node20/0+node19/0+node18/0+node17/0+node16/0+node15/0
[amit@node14 PBS ~]:cat $PBS_NODEFILE
```

```
node14
node21
node20
node19
node18
node17
node16
node15
[amit@node14 PBS ~]:exit
logout

qsub: job 2608.onyx.boisestate.edu completed
[amit@onyx ~]$
```

The command pbsget attempts to allocate the requested number of nodes from the PBS system. If it succeeds, it starts a new shell with the prompt modified to have PBS in the prompt. Note that the environment variable `PBS_NODEFILE` contains the name of a file that contains the list of noes allocated by PBS to the user. Now the user can run MPI parallel programs. When the user is done they would type `exit` to end the interactive PBS session.

If the required number of nodes are not available, then `pbsget` will wait. A user can cancel the request by typing in `Ctrl-c` and try again later. Remember to use `qstat -n` (or `cnodes`) to check the status of the nodes.

# 5 Running parallel programs in batch mode

## 5.1 Getting ready to run PBS batch jobs

To be able to run PBS jobs, you must check your `.bash_profile` and `.bashrc` files carefully. Any command that manipulates the terminal must be put in a conditional statement of the following form.

```
if test "$PBS_ENVIRONMENT" = "PBS_INTERACTIVE" -o -z "$PBS_ENVIRONMENT"
then
# set up the prompt to the hostname
#
PS1="[\u@\h \W]":
fi
```

The reason is that there is no terminal when you are running in batch mode. If you try to manipulate the terminal in batch mode, your login will fail and your batch job will not run. The environment variable `PBS_ENVIRONMENT` is set by PBS to be either `PBS_INTERACTIVE` or `PBS_BATCH`. When you are not running under PBS, then the variable is unset.

### 5.2 Preparing a PBS batch job script

Any parallel program that takes more than a few minutes should normally be run as a PBS batch job. In order to run it as a PBS batch job, you will need to prepare a PBS batch script (which is just a shell script with some additional features). Here is a sample PBS batch job (`~/amit/cs430/lab/MPI/parallel-sum/sum.pbs`):

```
#!/bin/sh
#PBS -l nodes=16:node
# This is a PBS job submission script. It asks for 16 nodes in the cluster
# to run the MPI application on.
#
# IMPORTANT NOTE:  Be sure to modify the "cd" command below to switch
# to the directory in which you are currently working!
#
#------------------------------------------------------------------------

cd /home/faculty/amit/cs430/lab/MPI/parallel-sum
mpiexec -n 16 spmd_sum_3 20000000
```

The line starting with `#PBS` is a PBS directive. There are many PBS directives but the one we will use is mainly the one that lists the nodes that we need to run our program. The following list shows some common options that can be used in the PBS directives:

| PBS option | Description |
|---|---|
| -N jobname | name the job jobname |
| -l cput=N | request N seconds of CPU time; N can also be in hh:mm:ss form |
| -l mem=N[KMG][BW] | request N kilo—mega—gigabytes—words of memory |
| -l nodes=N:ppn=M | request N nodes with M processors per node |
| -m b | mail the user when the job begins execution |
| -m e | mail the user when the job completes |
| -m a | mail the user if the job aborts |
| -a 1800 | start job on or after 6pm |
| -o outfile | redirect standard output to outfile |
| -e errfile | redirect standard error to errfile |
| -j oe | combine standard output and standard error |

For a full list, see the man page for `pbs_resources` on the cluster.

Here is another sample PBS batch job.

```
#!/bin/sh
#PBS -l nodes=16:node
#PBS -m  be
#PBS -a  2200
#PBS -o psum.log
# This is a PBS job submission script. It asks for 16 nodes in the cluster
# and asks the job to be scheduled at 10pm or later, the user to be mailed
# when the job begins and when it ends and capture the output in the file psum.log
#
```

```
# IMPORTANT NOTE:  Be sure to modify the "cd" command below to switch
# to the directory in which you are currently working!
#
#-----------------------------------------------------------------------
cd /home/faculty/amit/cs430/lab/MPI/parallel-sum
mpiexec -n 16 spmd_sum_3 1000000000
```

### 5.3 Submitting a PBS batch job script

The command `qsub` can be used to submit a PBS job. Continuing with the example script `psort.pbs` from the previous subsection, we can submit it for execution as follows.

```
cd lab/MPI/parallel-sum
qsub sum.pbs
```

The status of a job can be checked with the `qstat -a` command. Using `qstat -n` also shows the nodes that were allocated to your job.

```
[amit@onyx parallel-sum]$ qstat -a
onyx.boisestate.edu:
                                                                  Req'd   Elap
Job ID                  Username    Queue    Jobname SessID  NDS  TSK Memory Time S  Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ----------
2594.onyx.boisestate.e  broyster    interact STDIN   21811  2 2:node  -- 00:30:00 R 00:00:54
2595.onyx.boisestate.e  amit        batch    sum.pbs   --   8 8:node  -- 02:00:00 Q    --
[amit@onyx parallel-sum]$ qstat -n

onyx.boisestate.edu:
                                                                    Req'd   Elap
Job ID                  Username    Queue    Jobname SessID  NDS  TSK Memory  Time  S Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ----------
2594.onyx.boisestate.e  broyster    interact STDIN    21811  2 2:node -- 00:30:00   R 00:00:58
   node18/0+node19/0
2595.onyx.boisestate.e  amit        batch    sum.pbs    --   8 8:node -- 02:00:00   Q  -- --
[amit@onyx parallel-sum]$ qstat -n

onyx.boisestate.edu:
                                                                  Req'd       Elap
Job ID                  Username    Queue    Jobname SessID  NDS TSK Memory Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------   ----------
2594.onyx.boisestate.e  broyster    interact STDIN   21811   2 2:node  --   00:30:00 R  00:01:06
   node18/0+node19/0
2595.onyx.boisestate.e  amit        batch    sum.pbs   0     8 8:node  --   02:00:00 R  00:00:06
   node01/0+node08/0+node07/0+node06/0+node05/0+node04/0+node03/0+node02/0
[amit@onyx parallel-sum]$
```

You can delete jobs with the `qdel` command.

The standard output and standard error streams are redirected into the files `psort.pbs.oxxx` and `psort.pbs.exxx`, where `xxx` is the job number assigned by PBS.

In case of an error in running the job after it has been accepted in the queue, PBS sends an email to the user.

# 6 Debugging MPI programs

## 6.1 Using print statements

Using print statements will work with a MPI program, as the output from all processes is sent back to the console. However, it can be hard to sort out the output from different processes. Using the `-l` option with `mpiexec` will tag each output line with the process number that it came from. See the example below.

```
[amit@node05 PBS ~/cs430/lab/MPI/parallel-sum]:mpiexec -n 4 spmd_sum_3 1000000
 I got 250000 from 0
Sending partial sum back: 250000
Sending partial sum back: 250000
Sending partial sum back: 250000
I got 250000 from 3
I got 250000 from 2
I got 250000 from 1
The total is 1000000. Time taken = 0.001284 seconds

[amit@node05 PBS ~/cs430/lab/MPI/parallel-sum]:mpiexec -l -n 4 spmd_sum_3 1000000
[2] Sending partial sum back: 250000
[3] Sending partial sum back: 250000
[1] Sending partial sum back: 250000
[0]  I got 250000 from 0
[0] I got 250000 from 2
[0] I got 250000 from 3
[0] I got 250000 from 1
[0] The total is 1000000. Time taken = 0.000842 seconds
[amit@node05 PBS ~/cs430/lab/MPI/parallel-sum]:
```

## 6.2 Using logging and visualization

Logging and visualization support for MPI programs is provided with the MPE (MPI Programming Environment) software. The logging library provided with MPE tracks all MPI calls and saves the information to a log file that can be visualized.

- Link with the libraries `-llmpe -lmpe` to enable logging and the MPE environment. Then run the program as usual and a log file will be produced.

- The log file can be visualized using the `jumpshot` program that comes bundled with MPE.

When a program is linked with the MPE libraries, you will see a bit of extra output at the end that shows the logfile being written out.

```
[amit@localhost token-ring]: mpiexec -n 4 token_ring_mpi
Pass a token through the   4 id ring:
     0 ->      1 ->      2 ->      3 ->      0
received token ring on 1 from 0
received token ring on 2 from 1
received token ring on 3 from 2
token ring done
...
Writing logfile....
Enabling the Default clock synchronization...
Finished writing logfile token_ring_mpi.clog2.
[amit@localhost token-ring]
```

After the program finishes, then there will be a log file created with the exntensiom `clog2`. Start the jumpshot program on this file as follows:
`jumpshot token_ring_clog2`
and follow directions from the program. Figure 1 shows a sample visualization.

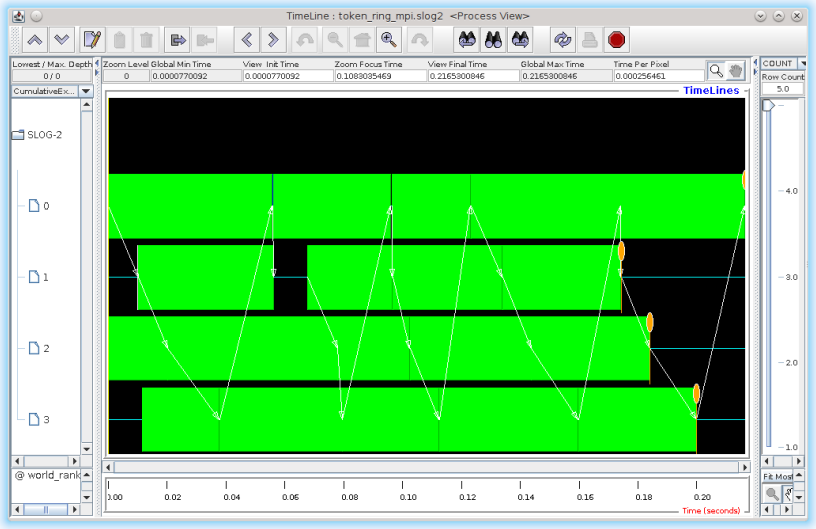



Figure 1: MPI Visualization using Jumpshot

### 6.3 Using Valgrind

Don't underesitmate the power of Valgrind! You can simply invoke it as shown below.

```
mpiexec -l -n 2 valgrind spmd_sum_3 1000
```

Valgrind also has direct hooks for MPI. See the page at:

`http://valgrind.org/docs/manual/mc-manual.html#mc-manual.mpiwrap`

for more details.

### 6.4 Using a debugger

- We can start up multiple MPI processes under the control of a debugger like GDB. An example is shown below. The option `-enable-x` is required in the lab but not needed if you are running mpiexec locally on your system.

  ```
  mpiexec -enable-x -n 2 konsole -e gdb spmd_sum_3 1000
  ```

  This can be convenient in pinpointing a specific problem. However, this is a bit like one person riding multiple bikes!

- Use a commercial parallel debugger, such as TotalView or DDT, that understands MPI jobs and can deal with multiple processes at once. These are much more sophisticated. But they are also expensive.

## 7 Documentation

Here are some sources of documentation:

- There are man pages for all MPI utilities and library calls. For example, to read the man page for `MPI_send()`, type:

  ```
  man MPI_send
  ```