Parallel Astronomical Data Processing or How to Build a Beowulf Class Cluster for High Performance Computing?

Navtej Singh

n.saini1@nuigalway.ie

Version 1.0

Centre for Astronomy, School of Physics
National University of Ireland, Galway

DISCLAIMER

Revisions:

1. Version 1.0: January 2012
   Original version

Contents

1 Introduction

Beowulf cluster is a collection of dedicated computing nodes made with commodity class hardware, connected using commercial off the self (COTS) network interfaces, and running open source infrastructure [1]. It can be used for High Availability (HA) or High Performance (HP) applications. Technically, cluster of workstations isn't a beowulf, as the workstations are not dedicated to beowulf but performing other tasks also. For the present work, we will be using cluster of workstations and beowulf cluster interchangeably, as processor and network load is minimal for most of the other tasks.

The beowulf cluster mentioned in this document is created for high performance computing and can be easily scaled up to include more computing nodes.

We will start with listing the hardware and software requirements for creating such a cluster (Section-2). A step-by-step procedure on how to construct the cluster is discussed in Section-3. Basic sanity test to check every part of the cluster are discussed in Section-4. Two parallel astronomical data processing programs are used to highlight the power of cluster for such tasks (Section-5). Some of the issues that may arise during cluster construction and their resolution are outlines in Section-6.

2 Requirements

2.1 Hardware

Commodity hardware is used to create the beowulf cluster. Such a cluster can be heterogenous i.e. computing nodes made of personal computers, laptops, headless (and diskless) machines etc. Similarly, network interfaces between the machines can be commercial off-the-self. In the present configuration, two personal computers (quad core machines) and one macbook pro were connected through a gigabit switch (1000Base-TX). Router or hub can also be used instead of switch although most of the routers available for home and office use only support 10Base-T and 100Base-TX networking. Machines in the cluster were able to talk to outside world through a router (optional). Refer to Section-3.2 for cluster networking layout and configuration.

Hardware specifications of the machines and network devices in the cluster are listed in Table 1.

Indentifier	Device	Specification	Reference
Node1	Personal Computer 1	AMD Phenom II X4 B60 Quad Core @ 3.51GHz. 4GB @ 1066MHz DDR3 RAM.	[2]
Node2	Personal Computer 2	Intel Core i7 920 Quad Core @ 2.67GHz (4 x 2 = 8 threads). 6GB @ 1066MHz DDR3 RAM	[3]
Node3	Macbook Pro	Intel Core 2 Duo P8700 @ 2.53GHz. 4GB @ 1066MHz DDR3 RAM	[4]
Switch	Gigabit Switch	Netgear Prosafe 5 port gigabit switch	[5]
Router	10Base-T/100Base-TX Router	Netgear 54Mbps Wireless Router	[6]

Table 1: Beowulf cluster hardware configuration

2.2 Software

Theoretically, cluster with different operating systems (OS) on the nodes can be constructed but to keep things simple, 32-bit Linux operating system was taken as the base OS on all the nodes. Ubuntu linux was installed natively on Node1 whereas it was installed as virtual machine on on Node2 and Node3. Open source virtualization software *VirtualBox* was used. Following software were also installed to have a functional linux beowulf cluster:

Software	**Version**	**Website**
Ubuntu 32-bit OS	11.10	Ubuntu Linux OS
VirtualBox	4.1.8	VirtualBox virtualization
Openssh server	5.8	Openssh SSH server
NFS server	4	Network File System server
GlusterFS	3.2.1	Gluster distributed file system
MPICH2	1.4	MPI protocol library
MPI4Py	1.2.1	MPI python binding
Torque	2.5.5	PBS resource manager
Maui scheduler	3.3.1	Maui scheduler
ESO Scisoft	7.5	Scisoft software package

VirtualBox allows running multiple virtual machines (operating systems) on the same machine and can utilize up to 32 virtual processor cores. Although, setting the number of virtual core equal to actual processor cores is recommended for better performance. Details about installing and configuring software is discussed in Section-3.

3 Linux Cluster Setup

A step-by-step procedural flowchart to construct a functional beowulf cluster is show in Figure 1. It is assumed that Ubuntu OS (or any other linux distribution) is already installed on the cluster nodes, either natively or in virtual environment.

The discussion is targeted for Ubuntu linux but should be valid for any Debian based Linux distribution. Commands and configuration setting may vary for RPM based distributions.

3.1 Virtualbox Configuration

If using VirtualBox to run Ubuntu as guest operating system, a small tweak is required in the networking setting. By default, guest OS uses the same internet protocol (IP) address as its host machine. To assign unique IP address to the guest machine, change the wired network interface adapter to 'Bridged Adapter' . This can be done in virtual machine's network setting preference as shown in Figure 2.

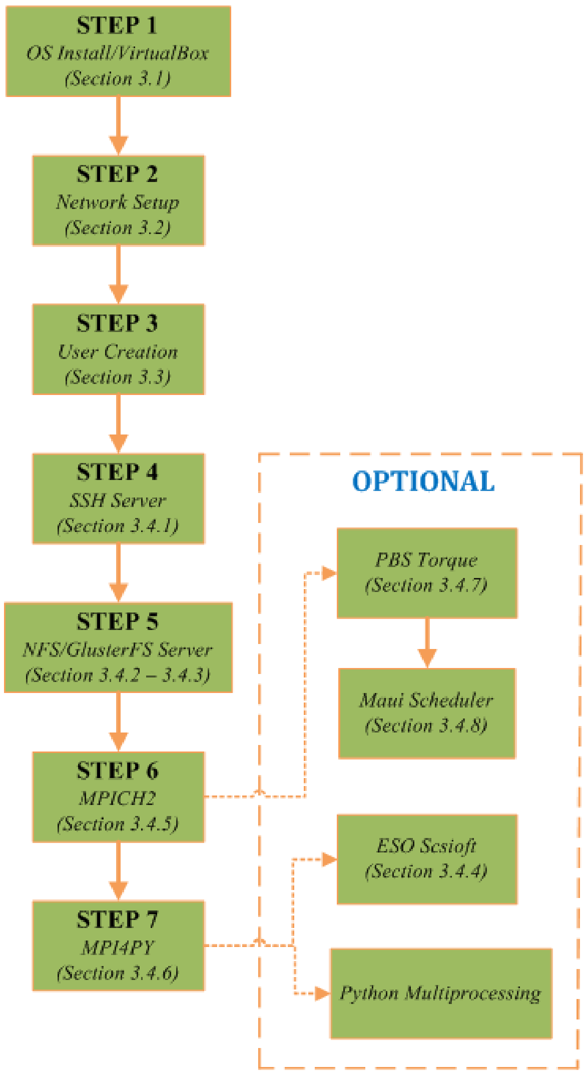
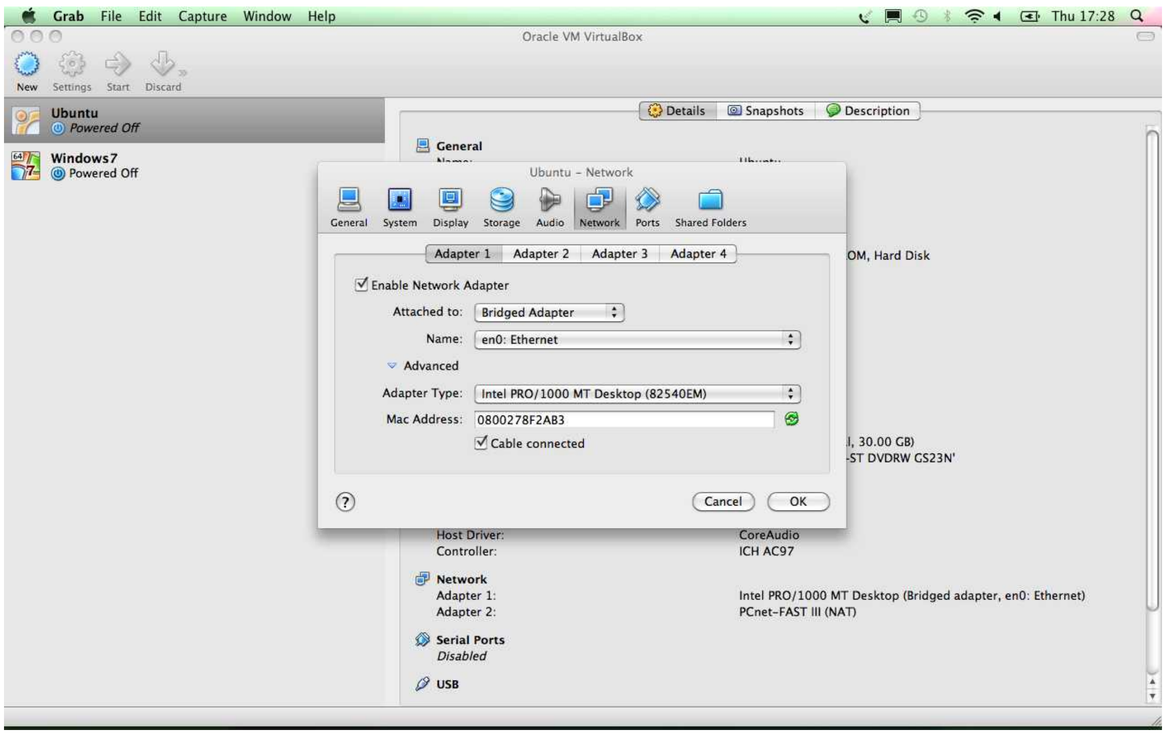


Figure 1: Beowulf cluster setup flowchart


Grab File Edit Capture Window Help
Thu 17:28
Oracle VM VirtualBox
New Settings Start Discard
Details Snapshots Description
Ubuntu – Network
General System Display Storage Audio Network Ports Shared Folders
Adapter 1 Adapter 2 Adapter 3 Adapter 4
Cancel OK

Ubuntu
Powered Off
Windows7
Powered Off

☑ Enable Network Adapter
Attached to: Bridged Adapter
Name: en0: Ethernet
Advanced
Adapter Type: Intel PRO/1000 MT Desktop (82540EM)
Mac Address: 0800278F2AB3
☑ Cable connected

General
OM, Hard Disk
l, 30.00 GB)
-ST DVDRW GS23N'
Host Driver: CoreAudio
Controller: ICH AC97
Network
Adapter 1: Intel PRO/1000 MT Desktop (Bridged adapter, en0: Ethernet)
Adapter 2: PCnet-FAST III (NAT)
Serial Ports
Disabled
USB

Figure 2: VirtualBox Bridged Adapter

Number of virtual processors can be set under 'System' configuration panel. It is recommended that the virtual processors be equal to the actual number of processors (or cores) in the machine. For Node2, memory for the virtual machine was set to 3.6 GB (out of 6 GB available) and 1.8 GB on Node3 virtual machine (out of 4 GB available). Please refer to VirtualBox's user manual for details about VirtualBox configuration parameters and options.

3.2 Network setup

Network diagram of the linux cluster with three nodes is shown in Figure 3. Three machines are connected through a gigabit switch, which in turn is connected to the desktop router. Router is optional and is only required to have access to outside world.

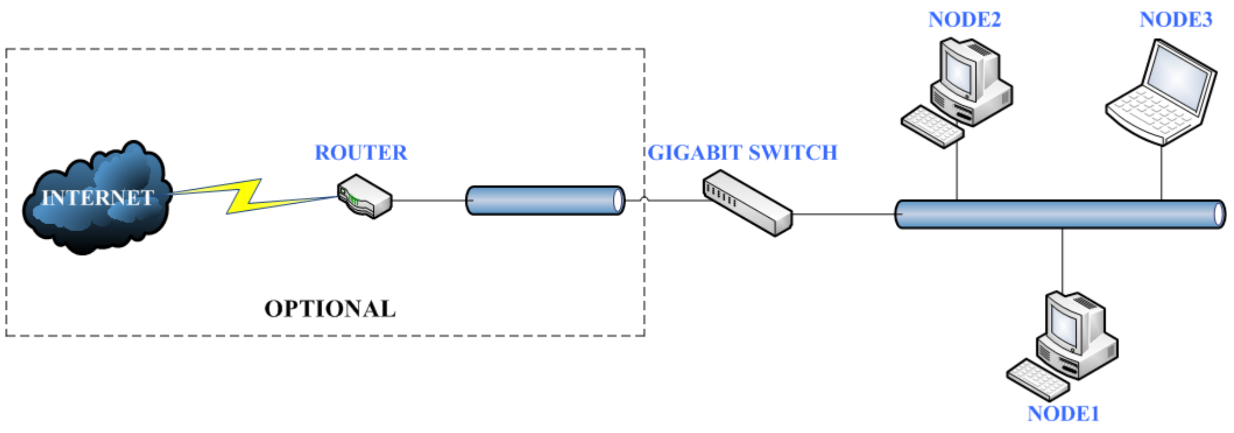


Figure 3: Linux beowulf cluster diagram

As the cluster include only three nodes, Class-C network addressing was used. For network with hundreds or thousands of nodes, Class-B or Class-A network addressing can be used [7]. Class-C addressing uses 192.168.*.* internet protocol (IP) addresses. Nodes on the cluster were assigned static ip addresses (i.e. 192.168.1.x; where x varies from 2 to 254). To access the machines using their host name (instead of IP address), following lines were added to */etc/hosts* file (on all the three machines):

```
192.168.1.2   Node1
192.168.1.3   Node2
192.168.1.4   Node3
```

Network interface[1] on the nodes were configured with static ip addresses using the following commands (on Node1) -

1. Open network configuration file for editing (need root access) using vi (or any other text editor)[2] -

   ```
   $ sudo vi /etc/network/interfaces
   ```

2. If *eth0* is configured with DHCP, comment it out[3] -

   ```
   % iface eth0 inet dhcp
   ```

[1]wired ethernet interface eth0
[2]$ corresponds to shell command prompt
[3]% corresponds to comment

3. Add a new static ip entry for *eth0* -

   ```
   address      192.168.1.2
   netmask    255.255.255.0
   network      192.168.1.0
   broadcast    192.168.1.1
   gateway      192.168.1.1
   ```

4. Save and close the file.

5. Restart the networking service (or stop and start *eth0* interface) -

   ```
   $ sudo /etc/init.d/networking restart
   ```

Similarly, static IP address was configured on Node2 and Node3 (the only difference is IP address).

3.3 User creation

A new user was created on all the nodes to run the parallel jobs under it. It was named *mpiu* (mpi user for short). New user can either be created using user/group graphical user interface (GUI) manager in Ubuntu or through command line. If using Ubuntu server edition (instead of desktop) then command line may be the only method to create a new user. Use the following command to create a new user *mpiu* (need root access):

```
$ sudo useradd --base-dir /home/mpiu --create-home --shell /bin/bash -U mpiu
```

User *mpiu* is created with */home/mpiu* as its home directory, bash as its default shell and it belongs to *mpiu* primary group. Set the password using following command -

```
$ sudo passwd mpiu
```

Some important commands for user management that may be useful -

1. To find user id (UID) and group id (GID) of user *mpiu*, use the following command -

   ```
   $ id mpiu
   ```

2. To change the user id for user *mpiu*, use the following command -

   ```
   $ sudo usermod -u new_id mpiu
   ```

   where *new_id* is new UID for user *mpiu*.

3. GID for a group can be changed using the following command -

   ```
   $ sudo groupmod -g new_id mpiu
   ```

   where *new_id* is new GID for group *mpiu*.

4. In case you made a mistake and want to delete the user (along with its home directory), use the following command -

   ```
   $ sudo userdel -r mpiu
   ```

Important - Set password, user id (UID) and group id (GID) of *mpiu* same on all the nodes of the cluster.

3.4 Software Installation and Configuration

3.4.1 Openssh server

Install openssh server on all the three nodes from Ubuntu repository -

```
$ sudo apt-get install openssh-server
```

It can also be installed from the source code if the nodes are not connected to internet.

To have password-less access to ssh server, public key authentication can be used. The way to do this is to generate private-public key pair on the server using either RSA or DSA encryption algorithm. The public key generated can then be appended to authorized_keys on the client side. For our purpose, we will be using RSA authentication algorithm. Follow the steps below to create private-public keys on Node1 and append it to authorized_keys on Node2 and Node3:

1. Log in as *mpiu* user on Node1 and generate private-public key pair using *ssh-keygen* command -

   ```
   $ ssh-keygen -t rsa
   ```

   Leave passphrase empty when prompted. By default, public and private keys are generated in */home/mpiu/.ssh* directory -

   ```
   /home/mpiu/.ssh/id_rsa: Private key
   /home/mpiu/.ssh/id_rsa.pub: Public key
   ```

2. Append id_ras.pub file from Node1 to Node2 and Node3's /home/mpiu/.ssh/authorized_keys file. Issue the following commands -

   ```
   $ ssh-copy-id -i /home/mpiu/.ssh/id_rsa.pub mpiu@Node2
   $ ssh-copy-id -i /home/mpiu/.ssh/id_rsa.pub mpiu@Node3
   ```

3.4.2 Network File System server

The simplest way to run parallel programs on a cluster is to create a network file system (NFS) on the master node (ideally the fastest machine with large hard disk) and mount it on all the other compute nodes in the cluster. In our case, Node1 is the master node running NFS server and is mounted on Node2 and Node3.

Install NFS server on Node1 using the following command -

```
$ sudo apt-get install nfs-kernel-server
```

Node1, Node2 and Node3 have to be configured so that network file system on Node1 can be mounted on Node2 and Node3 at boot time. Follow the steps below to configure the cluster nodes:

NODE1

1. Create directory structure that will act as network file system -

   ```
   $ sudo mkdir /mirror/mpiu
   ```

2. Change the owner and group of */mirror/mpiu* directory to *mpiu* -

   ```
   $ sudo chown -R mpiu:mpiu /mirror/mpiu
   ```

3. Open */etc/exports* file for editing in vi or your favorite text editor -

   ```
   $ sudo vi /etc/exports
   ```

4. Make a new entry to allow write access to the network file system on Node1 to Node2 and Node3[4] -

   ```
   /mirror/mpiu Node2(rw,async,subtree_check,tcp,nohide)
   /mirror/mpiu Node3(rw,async,subtree_check,tcp,nohide)
   ```

5. Save, close the file and restart the NFS server -

   ```
   $ sudo service nfs-kernel-server restart
   ```

NODE2 AND NODE3

1. Install *nfs-client* to mount nfs drives -

   ```
   $ sudo apt-get install nfs-client
   ```

2. Create directory structure to mount Node1's NFS share (directory structure name can be different) -

   ```
   $ sudo mkdir /mirror/mpiu
   ```

3. Change the owner and group of /mirror/mpiu directory to *mpiu* -

   ```
   $ sudo chown -R mpiu:mpiu /mirror/mpiu
   ```

4. Open */etc/fstab* file for editing in vi or any other text editor -

   ```
   $ sudo vi /etc/fstab
   ```

5. Create an entry in */etc/fstab* to auto mount Node1 NFS share at boot up -

   ```
   Node1:/mirror/mpiu /mirror/mpiu nfs rw,async,tcp 0 0
   ```

Note: NFS on Node1 can also be manually mounted using the following command -

```
$ sudo mount -t nfs -o async,tcp Node1:/mirror/mpiu /mirror/mpiu
```

3.4.3 GlusterFS [Optional]

There are couple of distributed file system alternatives to NFS which are specifically developed for high performance computing. One of these open source distributed file system is GlusterFS. GlusterFS is already included in Ubuntu's software repository. Follow the steps below to install and configure GlusterFS server and client -

NODE1

1. Install GlusterFS on Node1 (master node) from Ubuntu software repository -

   ```
   $ sudo apt-get install glusterfs-server
   ```

2. GlusterFS volumes are collection of bricks - each brick is an export directory on the server. Volume can be distributed, replicated, striped, distributed striped or distributed replicated. In our case, we will create a distributed volume with single brick pointing to */mirror/mpiu* directory on Node1. Issue the following command to create volume *mpi* on Node1 -

   ```
   $ sudo gluster volume create mpi transport tcp Node1:/mirror/mpiu
   ```

[4] Refer to NFS user guide for details about mount options

3. Volume has to be started before accessing -

   ```
   $ sudo gluster volume start mpi
   ```

GlusterFS volume information can be displayed using the following command -

```
$ sudo gluster volume info mpi
```

NODE2 and Node3

1. Install GlusterFS client on Node2 and Node3 -

   ```
   $ sudo apt-get install glusterfs-client
   ```

2. Test the installation by mounting the file system on Node2 and Node3 using following command[5] -

   ```
   $ sudo mount -t glusterfs Node1:/mpi /mirror/mpiu
   ```

3.4.4 ESO Scisoft

Image Reduction and Analysis Facility (IRAF) from National Optical Astronomy Observatories (NOAO) is one of the leading software package used by professional astronomers for astronomical image and data processing. Space Telescope Science Institute's PyRAF provides python wrapper for IRAF, which allows scripting in user friendly python programming language. ESO's Scisoft combines IRAF, PyRAF and many other astronomical software in a single easy to install package. Follow the steps below to install it on all the nodes -

1. Download the latest tar version of Scisoft from ESO's FTP site

2. Scisoft is developed for Fedora Linux and few of the package dependencies are missing from Ubuntu. Install the following packages from Ubuntu repository -

   ```
   $ sudo apt-get install tcsh libgfortran3 libreadline5
   $ sudo apt-get install libsdl-image1.2 libsdl-ttf2.0-0 unixodbc
   ```

3. Also, download the following two packages from Ubuntu's archive website -

   ```
   libg2c0_3.4.6-8ubuntu2_i386.deb   gcc-3.4-base_3.4.6-8ubuntu2_i386.deb
   ```

   Install using *dpkg* command -

   ```
   $ sudo dpkg -i gcc-3.4-base_3.4.6-8ubuntu2_i386.deb
   $ sudo dpkg -i libg2c0_3.4.6-8ubuntu2_i386.deb
   ```

4. Unarchive Scisoft to /scisoft using root permissions -

   ```
   $ cd /
   $ sudo tar xvfz /path_to_scisoft/scisoft*.tar.gz
   ```

5. To run IRAF/PyRAF, environment variables PATH and LD_LIBRARY_PATH have to be set -

   ```
   $ . /scisoft/bin/Setup.bash
   ```

   Instead of running this command every time, make the entry "*. /scisoft/bin/Setup.bash*" in */etc/bash.bashrc* file so that the environment variables are set at boot up.

[5]follow steps under NFS client installation to create */mirror/mpiu* directory first

6. Set terminal type for IRAF as *xgterm* for user *mpiu* -
   ```
   $ mkdir /home/mpiu/iraf
   $ cd /home/mpiu/iraf
   $ mkiraf
   ```

3.4.5 MPICH2 library

Message passing Interface (MPI) protocol is one of most common message passing system used in parallel computing. MPI version 2 library MPICH2 from Argonne National Laboratory (ANL) is used for its ease of use and extensive documentation. It can be installed from Ubuntu software repository or from the latest source code from ANL's website. It has to be installed on all the cluster nodes.

```
$ sudo apt-get install mpich2
```

Any other MPI library can also be used.

3.4.6 MPI for Python - MPI4PY

MPI for Python provides binding of MPI for python programming language. MPI4PY module was chosen for its maturity and ease of use although the documentation is scarce. Download the latest version of the software from MPI4PY's google code website. Install it on all the nodes of the cluster. Issue the following commands to install[6] -

```
$ tar xvfz /path_to_mpi4py/mpi4py-ver.tar.gz
$ cd mpi4py-ver
$ sudo python setup.py install
```

It can also be installed using python setup tools[7].

3.4.7 PBS Torque Resource Manager [Optional]

Torque resource manager is open source counterpart of the commercially available resource manager PBS. It is one of the most commonly used resource manager for high performance computing. Torque has two components - server and client. Server is installed on the master node and client on all the other nodes on the cluster. Start the installation with torque server on the Node1[8] -

1. Download torque source package from Cluster Resources website.
2. Unarchive source code into */usr/local/src* directory as it will make it easy to un-install or update the package in future.
3. Run the following commands to compile-link and install the default libraries -
   ```
   $ cd /usr/local/src/torque-ver
   $ sudo ./configure
   ```

[6] *ver* is version number of mpi4py
[7] more details about setup tools on python website
[8] refer to PBS/torque admin manual

```
$ sudo make
$ sudo make install
```

By default, torque files are installed in */var/spool/torque* ($TORQUEHOME henceforth) directory. Refer to torque admin manual on how to install torque in a non-default directory.

4. Use cluster resources tpackage to create tarballs for compute nodes by running the following command in the source package on Node1 -

   ```
   $ sudo make packages
   ```

   Copy the mom (*torque-package-mom-linux-i686.sh*) and client (*torque-package-clients-linux-i686.sh*) packages to Node2 and Node3 and run the following command to install torque client on compute nodes -

   ```
   $ sudo sh torque-package-mom-linux-i686.sh --install
   $ sudo sh torque-package-clients-linux-i686.sh --install
   ```

5. Enable torque as service on server (on Node1) and client (on Node1, Node2 and Node3) by copying the startup scripts from source package -

   Node1

   ```
   $ sudo cp contrib/init.d/debian.pbs_server /etc/init.d/pbs_server
   $ sudo update-rc.d pbs_server defaults
   ```

   Node1, Node2 and Node3

   ```
   $ sudo cp contrib/init.d/debian.pbs_mom /etc/init.d/pbs_mom
   $ sudo update-rc.d pbs_mom defaults
   ```

6. Now we need to initialize and configure torque server on Node1. Torque server's *serverdb* file contains configuration information of **pbs_server** and its queues. Run the following command to initialize *serverdb* and restart the server -

   ```
   $ sudo pbs_server -t create
   $ sudo /etc/init.d/pbs_server restart
   ```

   This will initialize basic server parameters and create a single *batch* queue.

7. Compute nodes can be added to the server either dynamically using *qmgr* or manually updating the nodes file. Compute nodes Node1, Node2 and Node3 are added to torque server -

   Dynamically

   ```
   $ sudo qmgr -c 'create node Node1'
   $ sudo qmgr -c 'create node Node2'
   $ sudo qmgr -c 'create node Node3'
   ```

   Manually

   Update $TORQUEHOME/server_priv/nodes file and insert the following three lines (for the three compute nodes) -

   ```
   Node1 np=4 cluster01 RAM4GB'
   Node2 np=4 cluster01 RAM3GB'
   Node3 np=2 cluster01 RAM2GB'
   ```

We have assumed 4 virtual processors each for Node1 and Node2 and 2 for Node3. Number of virtual processors (np) can be greater than actual processors (cores) on the node.

8. Restart the torque server on Node1 and start torque client on all the compute nodes -

   Node1

   ```
   $ sudo /etc/init.d/pbs_server restart
   ```

   Node1, Node2 and Node3

   ```
   $ sudo /etc/init.d/pbs_mom start
   ```

3.4.8 Maui Scheduler [Optional]

Torque's scheduler *pbs_sched* is very basic and therefore open source job scheduler Maui for cluster and supercomputers is used. It is only required to be installed on the master node. Follow the steps below to install it on Node1 -

1. Download Maui scheduler from Cluster Resources website. Registration is required before downloading the software.
2. Unarchive the source code in */usr/local/src* directory (as root) and run the following commands to install it[9] -

   ```
   $ cd /usr/local/src
   $ sudo tar xvfz /path_to_maui/maui-ver.tar.gz
   $ cd maui-ver
   $ sudo ./configure
   $ sudo make
   $ sudo make install
   ```

   By default, files are installed in */usr/local/maui* directory.
3. Start the scheduler -

   ```
   $ sudo /usr/local/bin/maui &
   ```

Note - Maui can be started at boot time by creating a service script for it and placing it in */etc/init.d/* directory.

Torque and Maui are optional software and does not necessarily required to run parallel jobs. But they make it easy to administrator large number of batch jobs on bigger installations.

4 Testing

If all of the previous steps were successful, it's time to test various components of the cluster. We will start with testing MPI installation, followed by python binding of MPI and ending with Torque/Maui resource manager functionality. Log into Node1 as *mpiu* user and follow the steps below to start testing -

MPI Testing

[9] *ver* is version number

1. MPI test programs can be downloaded from ANL's website. It has test code in C, C++ and FORTRAN programming languages. Place the test code in */mirror/mpiu* directory.
2. We will test MPI installation using the the standard *helloworld* C code. Save the following code as *hello.c* on */mirror/mpiu.*

Listing 1: hello.c

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

main(int argc,char* argv[]){
  int my_rank;
  int p;
  int source;
  int dest;
  int tag = 0;
  char message[100];
  MPI_Status status;

  MPI_Init(&argc,argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

  MPI_Comm_size(MPI_COMM_WORLD, &p);

  if (my_rank != 0) {
      sprintf(message, "Greetings from process %d!", my_rank);
      dest = 0;
      MPI_Send(message, strlen(message)+1, MPI_CHAR, dest, tag,
      MPI_COMM_WORLD);
  }
  else {
      for(source = 1; source < p; source++) {
      MPI_Recv(message, 100, MPI_CHAR, source, tag, MPI_COMM_WORLD,
      &status);
      printf("%s\n",message);
            }
   }

   MPI_Finalize();
```

3. Compile and link the code using MPICH2 C compiler *mpicc*[10].
4. To run code in parallel mode, host and number of processors has to provided. We will be running the code on all the three compute nodes using 10 processors -

```
$ mpiexec -np 10 -host Node1,Node2,Node3 ./hello
```

[10] *mpic++* for C++ programs and *mpif77* or *mpif90* for FORTRAN programs

This will automatically divide the total number of processes on the three nodes. To control the number of processes to start on each node, create a *hosts* file with following lines -

```
Node1:4
Node2:8
Node3:2
```

Now, run mpi program using following command -

```
$ mpiexec -np 10 -f hosts ./hello
```

If the job fails, run the code on each node separately to pinpoint the problem. For example, to run the program only on Node2 with 8 processes, execute the following command -

```
$ mpiexec -np 8 -host Node2 ./hello
```

Some basic errors encountered during MPI execution are listed in Section 6.

MPI4PY Testing

1. Save the following code listing to *helloworld.py* file on */mirror/mpiu* -

   Listing 2: helloworld.py

   ```
   #!/usr/bin/env python

   from mpi4py import MPI

   size = MPI.COMM_WORLD.Get_size()
   rank = MPI.COMM_WORLD.Get_rank()
   name = MPI.Get_processor_name()

   print 'Hello, World!I am process ' + str(rank) + ' of ' +
          str(size) + ' on ' + str(name)
   ```

2. Run the code -

   `$ mpiexec -n 10 -host Node1,Node2,Node3 python helloworld.py` OR
   `$ mpiexec -n 10 -f hosts python helloworld.py`[11]

Torque/Maui Testing

1. Check if all the nodes are up and running -

   ```
   $ pbsnodes -a
   ```

   state should be free for all the nodes. If any of the nodes is down or offline, it will show *state = down* and *state = offline* respectively. If it is down, check if network file system is mounted on that node and *pbs_mom* service is running.

2. Check if *maui* server is running and properly configured -

   ```
   $ showq
   ```

   It display all the active, idle and blocked jobs. Check out the number of nodes and processors - should match with the number of active nodes and processors in the cluster.

[11]*hosts* is file created in MPI testing

3. Run the following commands to test torque/maui -

   ```
   $ echo 'sleep 5' | qsub
   ```

   ```
   $ showq
   ```

   ```
   $ qstat -a
   ```

   Status of the job can be checked using either *showq* or *qstat* command.

4. To run *hello* through torque/maui, save the following code to *pbsjob* file -

   Listing 3: pbsjob

   ```
   #!/bin/bash
   #PBS -N pbsjob
   #PBS -q batch
   #PBS -l nodes=Node1:ppn=4+Node2:ppn=4+Node3:ppn=2
   #PBS -l walltime=1:00:00
   #PBS -e stderr.log
   #PBS -o stdout.log
   #PBS -V

   cd $PBS_O_WORKDIR

   mpiexec -n 10 ./hello
   ```

5. Python code *helloworld.py* can be run similarly using the following batch job code -

   Listing 4: pbsjob

   ```
   #!/bin/bash
   #PBS -N pbsjob
   #PBS -q batch
   #PBS -l nodes=Node1:ppn=4+Node2:ppn=4+Node3:ppn=2
   #PBS -l walltime=1:00:00
   #PBS -e stderr.log
   #PBS -o stdout.log
   #PBS -V

   cd $PBS_O_WORKDIR

   mpiexec -n 10 python helloworld.py
   ```

6. Submit the batch job *pbsjob* -

   ```
   $ qsub pbsjob
   ```

   Standard output and errors are written in *stdout.log* and *stderr.log* files. Refer to torque admin manual for details about job parameters.

5 Astronomical Data Processing

The main objective of this document is to create a beowulf class cluster for astronomical data and image processing. We are going to take two example codes - one written in C and another in python to demonstrate parallel data processing on a cluster. As mentioned in Section 3, ESO Scisoft is used for astronomical image processing. For interactive image display, DS9 python wrapper *pyds9* is used. It can be downloaded from Harvard-Smithsonian Center for Astrophysics website.

5.1 CRBLASTER

A parallel version of van Dokkum's L.A.COSMIC algorithm to remove cosmic rays from astronomical images was developed by Kenneth J. Mighell [8]. It uses message passing interface protocol and is written in C. We will be using this program to remove cosmic rays from a 800x800 pixel HST WFPC2[12] image.

Follow the steps below to install and execute CRBLASTER [13] on our cluster -

1. Download CRBLASTER source code from Mighell's webpage
2. Unarchive the tar file in */mirror/mpiu* directory -
   ```
   $ tar xvfz crblaster.tar.gz
   ```
3. Change to *crblaster* directory and make CTFITSIO library (used to handle FITS[14] image files) -
   ```
   $ make cfitsio
   ```
4. Make *crblaster* using following command -
   ```
   $ make
   ```
5. Run CRBLASTER on a 800x800 pixel image -
   ```
   $ cp images/in_800x800.fits in.fits
   $ mpiexec -np 10 -f hosts ./crblaster 1 2 5
   ```
   OR
   ```
   $ qsub pbs_job
   ```
   This will generate a clean output image - *out.fits*. Refer to CRBLASTER's website for details about input parameters.

5.2 PIX2SKY

IRAF package STSDAS has a task for transforming image pixel coordinates in HST images to sky's RA/DEC coordinates. But it processes only one pixel coordinate at a time. Running it on an image which requires thousands to millions of coordinates position transformations will take a very long time (e.g. running it on an image of a dense globular star cluster). We have developed a parallel pythonic version of this module - *pix2sky*.

Pix2sky uses STScI's *pyfits*[15] module. ESO Scisoft package already includes *pyfits*. If not using ESO Scisoft, *pyfits* can be downloaded from STScI's website and installed locally. To execute *pix2sky* on the cluster, follow the steps below -

[12]Hubble Space Telescope Wide Field Planetary Camera 2
[13]more details on Mighell's website
[14]Flexible Image Transport System
[15]Space Telescope Science Institute

1. Download the code from our website.
2. Unarchive the package in */mirror/mpiu* directory.
3. Apart from the program, the package includes a 800x800 pixel HST image and file with 1 million X,Y pixel coordinates to be transformed to sky RA,DEC coordinates. Change to *pix2sky* directory and execute the following command -

   ```
   $ mpiexec -n 10 -f hosts python pix2sky.py data/in.fits data/in_xy.cat
   ```
   OR
   ```
   $ qsub pbs_job
   ```

   A pbs/torque batch job can also be created to execute the software on the cluster. Output is a file with X,Y pixel coordinates and corresponding RA,DEC values.

It does not only runs on cluster of machines (using MPI protocol) but can also be executed on a single multicore machine (using python's *multiprocessing* module). *Multiprocessing* module was only introduced in python version 2.6. Latest version[16] of ESO Scisoft is still using python 2.5 and therefore multiprocessing module is not natively available. But backport for *multiprocessing* module is available for python 2.4 and 2.5. To run *pix2sky* on a multicore machine (if using python $<$ 2.6), download *multiprocessing* backport from python website and install it locally.

The program automatically detects the number of processors (cores) on the system and utilizes all of them. Execute the following command to run the program on all the cores -

```
$ python pix2sky_multi.py data/in.fits data/in_xy.cat
```

Number of processors can be controlled by using *-n* flag. Refer to *pix2sky* help - *$ python pix2sky_multi.py –help*, for all the program options.

6 Troubleshooting

Some common issues faced during cluster construction and their resolution -

1. **Communication error between cluster nodes**. There can be many different reasons for communication errors between the nodes. Few things to check -
   (a) Network file system (or GlusterFS) not mounted on all the nodes.
   (b) SSH server not running or properly configured on the nodes
   (c) Error in */etc/hosts* file. Hostname should point to one and only one IP address. On many machines, hostname may be pointing to 127.0.0.1. Comment it out.
2. **Proxy server**. Installing python packages on nodes behind proxy servers may fail. Set environment variable *http_proxy* to proxy server for root user. On bash shell, execute -

   ```
   $ export http_proxy=proxy_serve_hostnamer:port
   ```
3. **NFS version 4**. User and group assigned to xxxxxxx rather than *mpiu*. This may give file permission errors while running jobs on the cluster. Set the following parameters in */etc/default/nfs-common* file on master node (Node1) -

   ```
   NEED_STATD="no"
   NEED_IDMAPD="yes"
   ```

[16]version 7.5

Restart *nsf-server* -

```
$ sudo /etc/init.d/nfs-kernel-server restart
```

NFS server can be set permanently to run as version 3 by making the following changes to */etc/default/nfs-kernel-server* -

```
RPCNFSDCOUNT="16 –no-nfs-version 4"
```

and restart the server.

4. **SSH password-less log in**. Even if after appending master node's public key to authorized keys on nodes it is asking for password, verify that the UID and GID of the directory on nodes is same as on the master node.

5. **Torque node is down**. Verify that *pbs_mom* is running on compute nodes -

   ```
   $ ps aux | grep pbs_mom
   ```

6. **Python.h missing**. Python header files are required for compiling *mpi4py* and *pyfits* python modules. Install them -

   ```
   $ sudo apt-get install python-dev
   ```

7. **PBS_MOM error: security violation on checkpoint file**. Checkpoint folder on master node should have 751 permission.

References

[1] R. G. Brown, “Engineering a beowulf-style compute cluster.”

[2] “AMD phenom^{TM} II processors.” http://www.amd.com/US/PRODUCTS/DESKTOP/PROCESSORS/PHENOM-II/Pages/phenom-ii.aspx, Jan. 2012.

[3] “Intel® core^{TM} i7-920 processor.” http://ark.intel.com/products/37147/Intel-Core-i7-920-Processor-(8M-Cache-2_66-GHz-4_80-GTs-Intel-QPI), Jan. 2012.

[4] “Intel® core^{TM}2 duo processor p8700.” http://ark.intel.com/products/37006/Intel-Core2-Duo-Processor-P8700-(3M-Cache-2_53-GHz-1066-MHz-FSB), Jan. 2012.

[5] “Netgear gigbit switch GS105.” http://www.netgear.com/business/products/switches/unmanaged-desktop-switches/GS105.aspx, Jan. 2012.

[6] “Netgear router WGR614v7.” http://support.netgear.com/app/products/model/a_id/2589, Jan. 2012.

[7] “Classful network.” http://en.wikipedia.org/wiki/Classful_network, Dec. 2011.

[8] K. J. Mighell, “CRBLASTER: a Parallel-Processing computational framework for embarrassingly parallel Image-Analysis algorithms,” *Publications of the Astronomical Society of the Pacific*, vol. 122, pp. 1236–1245, Oct. 2010.