

Practical – 1

Aim: Setup High Performance MPI Cluster using MPICH library and running MPI Application Program.

Theory:

MPI (Message Passing Interface):

- The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum, which has over 40 participating organizations, including vendors, researchers, software library developers, and users. The goal of the Message Passing Interface is to establish a portable, efficient, and flexible standard for message passing that will be widely used for writing message passing programs. As such, MPI is the first standardized, vendor independent, message passing library. The advantages of developing message passing software using MPI closely match the design goals of portability, efficiency, and flexibility. MPI is not an IEEE or ISO standard, but has in fact, become the "industry standard" for writing message passing programs on HPC platforms.
- MPI is a specification for the developers and users of message passing libraries. By itself, it is NOT a library - but rather the specification of what such a library should be. MPI primarily addresses the message-passing parallel programming model: data is moved from the address space of one process to that of another process through cooperative operations on each process.

Reasons for using MPI:

- **Standardization** - MPI is the only message passing library that can be considered a standard. It is supported on virtually all HPC platforms. Practically, it has replaced all previous message passing libraries.
- **Portability** - There is little or no need to modify your source code when you port your application to a different platform that supports (and is compliant with) the MPI standard.
- **Performance Opportunities** - Vendor implementations should be able to exploit native hardware features to optimize performance. Any implementation is free to develop optimized algorithms.
- **Functionality** - There are **over 430** routines defined in MPI-3, which includes the majority of those in MPI-2 and MPI-1.

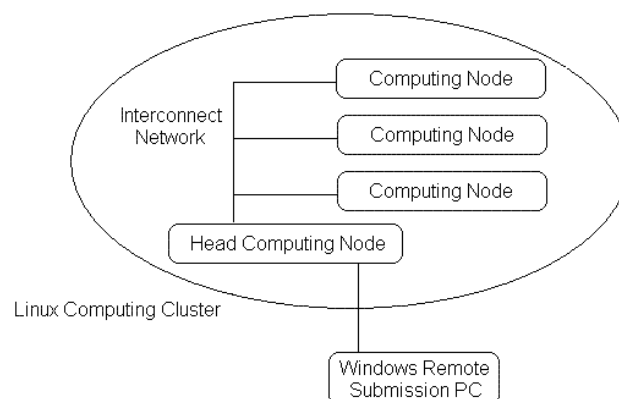


Figure 1 MPI cluster on linux

Output:

1. Edit host file

```
GNU nano 2.2.6 File: /etc/hosts
127.0.0.1 localhost
127.0.1.1 master
#
54.198.35.232 master
54.89.123.203 slave
#
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Figure 2 Host file of server

```
GNU nano 2.2.6 File: /etc/hosts
127.0.0.1 localhost
127.0.1.1 slave
#
54.89.123.203 slave
54.198.35.232 master
#
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Figure 3 Host file of slave

2. Install open-ssh server:

```
root@master:/home/ubuntu# sudo apt-get install openssh-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
openssh-server is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
root@master:/home/ubuntu#
```

3. Apply permission in sudoers file

```

GNU nano 2.2.6      File: /etc/sudoers

Defaults            secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:$
# Host alias specification
# User alias specification
# Cmnd alias specification
# User privilege specification
root    ALL=(ALL:ALL) ALL
ubuntu  ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin   ALL=(ALL)  ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

```

4. Install nfs server in master and nfs-client in slave

```

root@master:/home/ubuntu# apt-get install nfs-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'nfs-kernel-server' instead of 'nfs-server'
nfs-kernel-server is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
root@master:/home/ubuntu#

```

Figure 4 installation on master

```

root@slave:/# apt-get install nfs-client
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'nfs-common' instead of 'nfs-client'
nfs-common is already the newest version.
nfs-common set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
root@slave:/#

```

Figure 5 Installation on slave

5. Make test directory

```

root@master:/# mkdir /test

```

6. Sync the test directory with slaves

```

root@master:/# echo "/test *(rw, sync)" | sudo tee -a /etc/exports
/test *(rw, sync)
root@master:/#

```

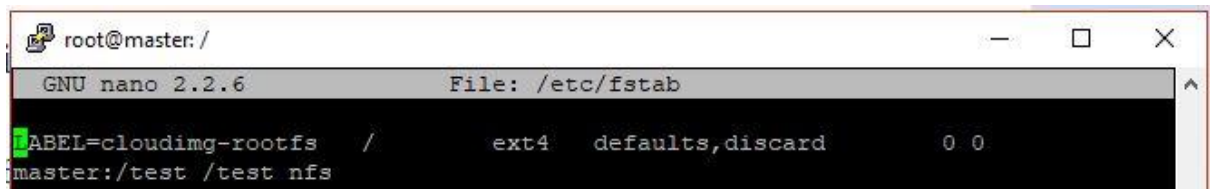
7. Restart nfs-server

```

root@master:/# sudo service nfs-kernel-server restart
* Stopping NFS kernel daemon                                [ OK ]
* Unexporting directories for NFS kernel daemon...          [ OK ]
* Exporting directories for NFS kernel daemon...
exportfs: /etc/exports [1]: Neither 'subtree_check' or 'no_subtree_check' specif
ied for export "*/test".
    Assuming default behaviour ('no_subtree_check').
    NOTE: this default has changed since nfs-utils version 1.0.x
                                                                [ OK ]
* Starting NFS kernel daemon                                [ OK ]
root@master:/#

```

8. Make add test directory in fstab

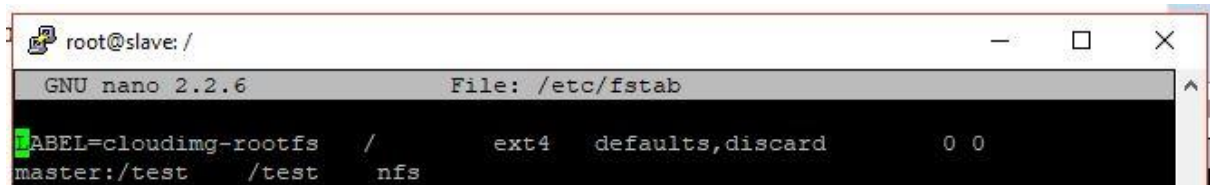


```

root@master: /
GNU nano 2.2.6 File: /etc/fstab
LABEL=cloudimg-rootfs / ext4 defaults,discard 0 0
master:/test /test nfs

```

Figure 6 Added to Master



```

root@slave: /
GNU nano 2.2.6 File: /etc/fstab
LABEL=cloudimg-rootfs / ext4 defaults,discard 0 0
master:/test /test nfs

```

Figure 7 Added to Slave

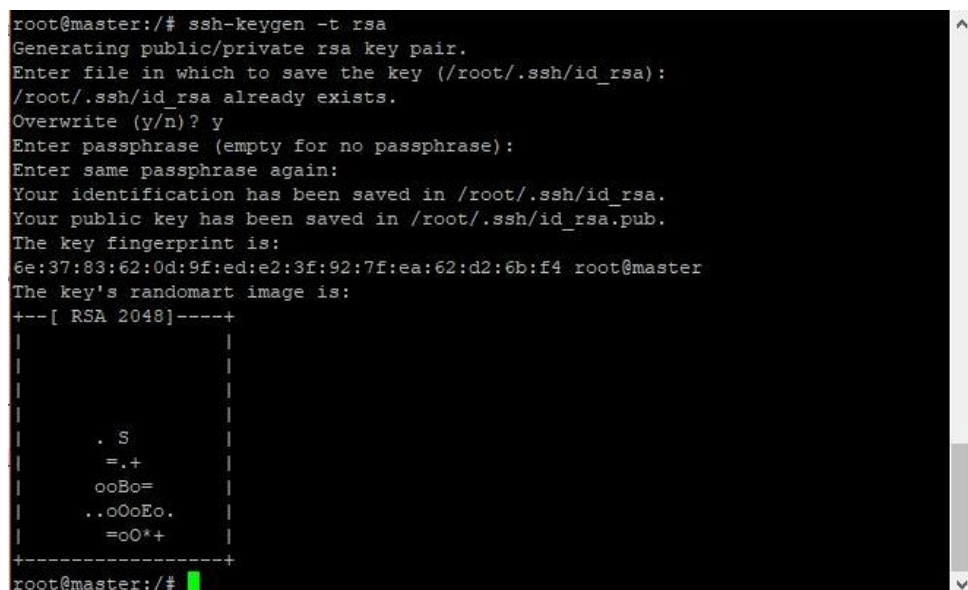
9. Mount test directory

```

root@slave:/# sudo mount master:/test /test
mount.nfs: /test is busy or already mounted
root@slave:/# mount -a
root@slave:/#

```

10. Generate rsa key



```

root@master:/# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
/root/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
6e:37:83:62:0d:9f:ed:e2:3f:92:7f:ea:62:d2:6b:f4 root@master
The key's randomart image is:
+--[ RSA 2048]-----+
|
|
|
|
|
|
|
|
|
|
+-----+
root@master:/#

```


11. Copy generated key into authorized_keys file

```
root@master:/# cd ~
root@master:~# cd .ssh
root@master:~/.ssh# cat id_rsa.pub >> authorized_keys
```

12. Install build_essentials

```
root@master:/# sudo apt-get install build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
root@master:/#
```

13. Install mpich2

```
root@master:/# sudo apt-get install mpich2
Reading package lists... Done
Building dependency tree
Reading state information... Done
mpich2 is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
root@master:/#
```

14. Create c file

```
root@master:/# cd /test
root@master:/test# ls
test_mpi  test_mpi.c
root@master:/test#
```

15. Compile and run c file

```
root@master:/test# mpicc test_mpi.c -o test_mpi
root@master:/test# ls
test_mpi  test_mpi.c
root@master:/test# ./test_mpi
Hey ,how are you ? from processor 0 of 1
root@master:/test#
```

16. Execute c code on cluster

```
root@master:/test# mpiexec -np 8 --hosts master,slave ./test_mpi
Hey ,how are you ? from processor 7 of 8
Hey ,how are you ? from processor 4 of 8
Hey ,how are you ? from processor 6 of 8
Hey ,how are you ? from processor 5 of 8
Hey ,how are you ? from processor 2 of 8
Hey ,how are you ? from processor 1 of 8
Hey ,how are you ? from processor 0 of 8
Hey ,how are you ? from processor 3 of 8
root@master:/test#
```

Conclusion:

We have learned the concept of cluster and implemented with mpich2 on ubuntu 14.04 on aws instance.