MpichCluster - Community Help Wiki

Setting Up an MPICH2 Cluster in Ubuntu

This guide describes how to build a simple MPICH cluster in ubuntu.

To understand the guide, a basic knowledge of command line usage and the principle mpich & clustering is assumed.

Here we have 4 nodes running Ubuntu server with these host names: ub0,ub1,ub2,ub3;

1. Defining hostnames in etc/hosts/

Edit /etc/hosts like these:

```
127.0.0.1     localhost
192.168.133.100 ub0
192.168.133.101 ub1
192.168.133.102 ub2
192.168.133.103 ub3
```

Note that the file shouldn't be like this:

```
127.0.0.1     localhost
127.0.1.1     ub0
192.168.133.100 ub0
192.168.133.101 ub1
192.168.133.102 ub2
192.168.133.103 ub3
```

or like this:

```
127.0.0.1     localhost
127.0.1.1     ub0
192.168.133.101 ub1
192.168.133.102 ub2
192.168.133.103 ub3
```

otherwise other hosts will try to connect to localhost when they try to reach ub0.

2. Installing NFS

NFS allows us to create a folder on the master node and have it synced on all the other nodes. This folder can be used to store programs. To Install NFS just run this in the master node's terminal:

```
omid@ub0:~$ sudo apt-get install nfs-server
```

To install the client program on other nodes run this command on each of them:

```
omid@ub1:~$ sudo apt-get install nfs-client
```

Note: if you want to be more efficient in controlling several nodes using same commands, ClusterSSH is a nice tool and you can find a basic two-line tutorial here.

3. Sharing Master Folder

Make a folder in all nodes, we'll store our data and programs in this folder.

```
omid@ub0:~$ sudo mkdir /mirror
```

And then we share the contents of this folder located on the master node to all the other nodes. In order to do this we first edit the /etc/exports file on the master node to contain the additional line

*/mirror *(rw,sync)*

This can be done using a text editor such as vim or by issuing this command:

```
omid@ub0:~$ echo "/mirror *(rw,sync)" | sudo tee -a /etc/exports
```

Now restart the nfs service on the master node to parse this configuration once again.

```
omid@ub0:~$ sudo service nfs-kernel-server restart
```

Note than we store out data and programs only in master node and other nodes will access them with NFS.

4. Mounting /master in nodes

Now all we need to do is to mount the folder on the other nodes. This can be done manually each time like this:

```
omid@ub1:~$ sudo mount ub0:/mirror /mirror
omid@ub2:~$ sudo mount ub0:/mirror /mirror
```

```
omid@ub3:~$ sudo mount ub0:/mirror /mirror
```

But it's better to change fstab in order to mount it on every boot. We do this by editing /etc/fstab and adding this line:

```
ub0:/mirror    /mirror    nfs
```

and remounting all partitions by issuing this on all the slave nodes:

```
omid@ub1:~$ sudo mount -a
omid@ub2:~$ sudo mount -a
omid@ub3:~$ sudo mount -a
```

5. Defining a user for running MPI programs

We define a user with same name and same userid in all nodes with a home directory in /mirror.

Here we name it "mpiu"! Also we change the owner of /mirror to mpiu:

```
omid@ub0:~$ sudo chown mpiu /mirror
```

6. Installing SSH Server

Run this command in all nodes in order to install OpenSSH Server

```
omid@ub0:~$ sudo apt-get install openssh-server
```

7. Setting up passwordless SSH for communication between nodes

First we login with our new user to the master node:

```
omid@ub0:~$ su - mpiu
```

Then we generate an RSA key pair for mpiu:

```
mpiu@ub0:~$ ssh-keygen -t rsa
```

You can keep the default ~/.ssh/id_rsa location. It is suggested to enter a strong passphrase for security reasons.

Next, we add this key to authorized keys:

```
mpiu@ub0:~$ cd .ssh
mpiu@ub0:~/.ssh$ cat id_rsa.pub >> authorized_keys
```

As the home directory of mpiu in all nodes is the same (/mirror/mpiu) , there is no need to run these commands on all nodes. If you didn't mirror the home directory, though, you can use `ssh-copy-id <hostname>` to copy a public key to another machine's authorized_keys file safely.

To test SSH run:

```
mpiu@ub0:~$ ssh ub1 hostname
```

If you are asked to enter a passphrase every time, you need to set up a keychain. This is done easily by installing... Keychain.

```
mpiu@ub0:~$ sudo apt-get install keychain
```

And to tell it where your keys are and to start an ssh-agent automatically edit your ~/.bashrc file to contain the following lines (where id_rsa is the name of your private key file):

```
if type keychain >/dev/null 2>/dev/null; then
  keychain --nogui -q id_rsa
  [ -f ~/.keychain/${HOSTNAME}-sh ] && . ~/.keychain/${HOSTNAME}-sh
  [ -f ~/.keychain/${HOSTNAME}-sh-gpg ] && . ~/.keychain/${HOSTNAME}-sh-gpg
fi
```

Exit and login once again or do a `source ~/.bashrc` for the changes to take effect.

Now your hostname via ssh command should return the other node's hostname without asking for a password or a passphrase. Check that this works for all the slave nodes.

8. Installing GCC

To be able to compile all the code on our master node (it's sufficient to do it only there if we do it inside the /mirror folder and all the libraries are in place on other machines) we need a compiler.

You can get gcc and other necessary stuff by installing the build-essential package:

```
mpiu@ub0:~$ sudo apt-get install build-essential
```

9.Installing Other Compilers

Other prefered compilers should be installed before installing MPICH.

In this step you may install other compilers such as Inter Fortran, SGI compiler , ...

10. Installing MPICH2

Now the last ingredient we need installed on all the machines is the MPI implementation. You can install MPICH2 using Synaptic by typing:

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

Alternatively, MPICH2 can be installed from source as explained in the MPICH installer guide or you can try using some other implementation such as OpenMPI.

To test that the program did indeed install successfully enter this on all the machines:

```
mpiu@ub0:~$  which mpiexec
mpiu@ub0:~$  which mpirun
```

11. setting up a machinefile

Create a file called "machinefile" in mpiu's home directory with node names followed by a colon and a number of processes to spawn:

```
ub3:4  # this will spawn 4 processes on ub3
ub2:2  # this will spawn 2 processes on ub2
ub1    # this will spawn 1 process on ub1
ub0    # this will spawn 1 process on ub0
```

11. Testing

Change directory to your mirror folder and write this MPI helloworld program in a file mpi_hello.c (courtesy of this blog):

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

int main(int argc, char** argv) {
    int myrank, nprocs;

    MPI_Init(&argc, &argv);
```

```
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    printf("Hello from processor %d of %d\n", myrank, nprocs);

    MPI_Finalize();
    return 0;
}
```

Compile it:

```
mpiu@ub0:~$ mpicc mpi_hello.c -o mpi_hello
```

and run it (the parameter next to -n specifies the number of processes to spawn and distribute among nodes):

```
mpiu@ub0:~$ mpiexec -n 8 -f machinefile ./mpi_hello
```

You should now see output similar to this:

```
Hello from processor 0 of 8
Hello from processor 1 of 8
Hello from processor 2 of 8
Hello from processor 3 of 8
Hello from processor 4 of 8
Hello from processor 5 of 8
Hello from processor 6 of 8
Hello from processor 7 of 8
```

Congratulations! You have a working MPI platform

References

For more information visit: