Building a simple Beowulf cluster with Ubuntu

Serrano Pereira – Version 1.1, September 11, 2013 | Updated for Ubuntu 12.10.

This document describes the basic steps to setting up a basic Beowulf cluster using the Ubuntu operating system.

> *A Beowulf cluster is a group of what are normally identical, commercially available computers, which are running a Free and Open Source Software (FOSS), Unix-like operating system, such as BSD, GNU/Linux, or Solaris. They are networked into a small TCP/IP LAN, and have libraries and programs installed which allow processing to be shared among them.* [1]
>
> — *Wikipedia*
> *Beowulf cluster*

This means a Beowulf cluster can be easily built with "off the shelf" computers running GNU/Linux in a simple home network. So building a Beowulf like cluster is within reach if you already have a small TCP/IP LAN at home with desktop computers running Ubuntu Linux (http://www.ubuntu.com/), or any other GNU/Linux distribution.

There are many ways to install and configure a cluster. There is OSCAR [2], which allows any user, regardless of experience, to easily install a Beowulf type cluster on supported Linux distributions. It installs and configures all required software according to user input.

There is also the NPACI Rocks toolkit [3] which incorporates the latest Red Hat distribution and cluster-specific software. Rocks addresses the difficulties of deploying manageable clusters. Rocks makes clusters easy to deploy, manage, upgrade and scale.

Both of the afore mentioned toolkits for deploying clusters were made to be easy to use and require minimal expertise from the user. But the purpose of this tutorial is to explain how to manually build a Beowulf like cluster. Basically, the toolkits mentioned above do most of the installing and configuring for you, rendering the learning experience mute. So it would not make much sense to use any of these toolkits if you want to learn the basics of how a cluster works. This tutorial therefore explains how to manually build a cluster, by manually installing and configuring the required tools. In this tutorial I assume that you have some basic knowledge of

the Linux-based operating system and know your way around the command line. I tried however to make this as easy as possible to follow. Keep in mind that this is new territory for me as well and there's a good chance that this tutorial shows methods that may not be the best.

The clustering tutorial from SCFBio [4] gives a good introduction to Beowulf clusters. It describes the prerequisites for building a Beowulf cluster and why these are needed.

Table of Contents

1. What's a Beowulf Cluster, exactly?

The book *Engineering a Beowulf-style Compute Cluster* by [brown2004] gives a more detailed answer to this question. According to this book, there is an accepted definition of a beowulf cluster. This book describes the true beowulf as a cluster of computers interconnected with a network with the following characteristics:

1. The nodes are dedicated to the beowulf cluster.
2. The network on which the nodes reside are dedicated to the beowulf cluster.
3. The nodes are Mass Market Commercial-Off-The-Shelf (M2COTS) computers.
4. The network is also a COTS entity.
5. The nodes all run open source software.

6. The resulting cluster is used for High Performance Computing (HPC).

(http://en.wikipedia.org/wiki/File:Beowulf.png)

Figure 1. The typical setup of a beowulf cluster.

2. Building a virtual Beowulf Cluster

It is not a bad idea to start by building a virtual cluster using virtualization software like VirtualBox (http://en.wikipedia.org/wiki/VirtualBox). I simply used my laptop running Ubuntu as the master node, and two virtual computing nodes running Ubuntu Server Edition were created in VirtualBox. The virtual cluster allows you to build and test the cluster without the need for the extra hardware. However, this method is only meant for testing and not suited if you want increased performance.

When it comes to configuring the nodes for the cluster, building a virtual cluster is practically the same as building a cluster with actual machines. The difference is that you don't have to worry about the hardware as much. You do have to properly configure the virtual network interfaces of the virtual nodes. They need to be configured in a way that the master node (e.g. the computer on which the virtual nodes are running) has network access to the virtual nodes, and vice versa.

3. Building the actual cluster

It is good practice to first build and test a virtual cluster as described above. If you have some spare computers and network parts lying around, you can use those to build the actual cluster. The nodes (the computers that are part of the cluster) and the network hardware are the usual kind available to the general public (beowulf requirement 3 and 4). In this tutorial we'll use the Ubuntu operating system to power the machines and open source software to allow for distributed parallel computing (beowulf requirement 5). We'll test the cluster with cluster specific versions of bioinformatics (http://en.wikipedia.org/wiki/Bioinformatics) tools that perform some sort of heavy calculations (beowulf requirement 6).

The cluster consists of the following hardware parts:

- Network
- Server / Head / Master Node (common names for the same machine)

- Compute Nodes
- Gateway

All nodes (including the master node) run the following software:

- GNU/Linux OS (http://en.wikipedia.org/wiki/Linux_distribution)
- Ubuntu Server Edition (http://www.ubuntu.com/server).
- Network File System (NFS) (http://en.wikipedia.org/wiki/Network_File_System_%28protocol%29)
- Secure Shell (SSH) (http://en.wikipedia.org/wiki/Secure_Shell)
- Message Passing Interface (MPI) (http://en.wikipedia.org/wiki/Message_Passing_Interface)
- MPICH (http://www.mpich.org/)

I will not focus on setting up the network (parts) in this tutorial. I assume that all nodes are part of the same private network and that they are properly connected.

4. Configuring the Nodes

Some configurations need to be made to the nodes. I'll walk you through them one by one.

4.1. Add the nodes to the hosts file

It is easier if the nodes can be accessed with their host name rather than their IP address. It will also make things a lot easier later on. To do this, add the nodes to the hosts file of all nodes ([ubuntuwiki], [linuxcom]). All nodes should have a static local IP address set. I won't go into details here as this is outside the scope of this tutorial. For this tutorial I assume that all nodes are already properly configured to have a static local IP address.

Edit the hosts file (`sudo vim /etc/hosts`) like below and remember that you need to do this for all nodes,

```
127.0.0.1       localhost
192.168.1.6     master
192.168.1.7     node1
192.168.1.8     node2
192.168.1.9     node3
```

Make sure it doesn't look like this:

```
127.0.0.1       localhost
127.0.1.1       master
192.168.1.7     node1
192.168.1.8     node2
192.168.1.9     node3
```

neither like this:

```
127.0.0.1       localhost
127.0.1.1       master
192.168.1.6     master
192.168.1.7     node1
192.168.1.8     node2
192.168.1.9     node3
```

Otherwise other nodes will try to connect to localhost when trying to reach the master node.

Once saved, you can use the host names to connect to the other nodes,

```
$ ping -c 3 master
PING master (192.168.1.6) 56(84) bytes of data.
64 bytes from master (192.168.1.6): icmp_req=1 ttl=64 time=0.606 ms
64 bytes from master (192.168.1.6): icmp_req=2 ttl=64 time=0.552 ms
64 bytes from master (192.168.1.6): icmp_req=3 ttl=64 time=0.549 ms

--- master ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.549/0.569/0.606/0.026 ms
```

Try this with different nodes on different nodes. You should get a response similar to the above.

In this tutorial, `master` is used as the master node. Once the cluster has been set up, the master node will be used to start jobs on the cluster. The master node will be used to spawn jobs on the cluster. The compute nodes are `node1` to `node3` and will thus execute the jobs.

4.2. Defining a user for running MPI jobs

Several tutorials explain that all nodes need a separate user for running MPI jobs ([ubuntuwiki], [linuxcom], [wong2008]). I haven't found a clear explanation to why this is necessary, but there could be several reasons:

1. There's no need to remember different user names and passwords if all nodes use the same username and password.

2. MPICH2 can use SSH for communication between nodes. Passwordless login with the use of authorized keys only works if the username matches the one set for passwordless login. You don't have to worry about this if all nodes use the same username.
3. The NFS directory can be made accessible for the MPI users only. The MPI users all need to have the same user ID for this to work.
4. The separate user might require special permissions.

The command below creates a new user with username "mpiuser" and user ID 999. Giving a user ID below 1000 prevents the user from showing up in the login screen for desktop versions of Ubuntu. It is important that all MPI users have the same username and user ID. The user IDs for the MPI users need to be the same because we give access to the MPI user on the NFS directory later. Permissions on NFS directories are checked with user IDs. Create the user like this,

```
$ sudo adduser mpiuser --uid 999
```

You may use a different user ID (as long as it is the same for all MPI users). Enter a password for the user when prompted. It's recommended to give the same password on all nodes so you have to remember just one password. The above command should also create a new directory `/home/mpiuser`. This is the home directory for user `mpiuser` and we will use it to execute jobs on the cluster.

4.3. Install and setup the Network File System

Files and programs used for MPI (http://en.wikipedia.org/wiki/Message_Passing_Interface) jobs (jobs that are run in parallel on the cluster) need to be available to all nodes, so we give all nodes access to a part of the file system on the master node. Network File System (NFS) enables you to mount part of a remote file system so you can access it as if it is a local directory. To install NFS, run the following command on the master node:

```
master:~$ sudo apt-get install nfs-kernel-server
```

And in order to make it possible to mount a Network File System on the compute nodes, the `nfs-common` package needs to be installed on all compute nodes:

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

We will use NFS to share the MPI user's home directory (i.e. `/home/mpiuser`) with the compute nodes. It is important that this directory is owned by the MPI user so that all MPI users can access this directory. But since we created this home directory with the `adduser` command earlier, it is already owned by the MPI user,

```
master:~$ ls -l /home/ | grep mpiuser
drwxr-xr-x   7 mpiuser mpiuser  4096 May 11 15:47 mpiuser
```

If you use a different directory that is not currently owned by the MPI user, you must change it's ownership as follows,

```
master:~$ sudo chown mpiuser:mpiuser /path/to/shared/dir
```

Now we share the `/home/mpiuser` directory **of the master node** with all other nodes. For this the file `/etc/exports` on the master node needs to be edited. Add the following line to this file,

```
/home/mpiuser *(rw,sync,no_subtree_check)
```

You can read the man page to learn more about the exports file (`man exports`). After the first install you may need to restart the NFS daemon:

```
master:~$ sudo service nfs-kernel-server restart
```

This also exports the directores listed in `/etc/exports`. In the future when the `/etc/exports` file is modified, you need to run the following command to export the directories listed in `/etc/exports`:

```
master:~$ sudo exportfs -a
```

The `/home/mpiuser` directory should now be shared through NFS. In order to test this, you can run the following command from a compute node:

```
$ showmount -e master
```

In this case this should print the path `/home/mpiuser`. All data files and programs that will be used for running an MPI job must be placed in this directory on the master node. The other nodes will then be able to access these files through NFS.

The firewall is by default enabled on Ubuntu. The firewall will block access when a client tries to access an NFS shared directory. So you need to add a rule with UFW (https://help.ubuntu.com/community/UFW) (a tool for managing the firewall) to allow access from a specific subnet. If the IP addresses in your network have the format `192.168.1.*`, then `192.168.1.0` is the subnet. Run the following command to allow incoming access from a specific subnet,

```
master:~$ sudo ufw allow from 192.168.1.0/24
```

You need to run this on the master node and replace "192.168.1.0" by the subnet for your network.

You should then be able to mount `master:/home/mpiuser` on the compute nodes. Run the following commands to test this,

```
node1:~$ sudo mount master:/home/mpiuser /home/mpiuser
node2:~$ sudo mount master:/home/mpiuser /home/mpiuser
node3:~$ sudo mount master:/home/mpiuser /home/mpiuser
```

If this fails or hangs, restart the compute node and try again. If the above command runs without a problem, you should test whether `/home/mpiuser` on any compute node actually has the content from `/home/mpiuser` of the master node. You can test this by creating a file in `master:/home/mpiuser` and check if that same file appears in `node*:/home/mpiuser` (where `node*` is any compute node).

If mounting the NFS shared directory works, we can make it so that the `master:/home/mpiuser` directory is automatically mounted when the compute nodes are booted. For this the file `/etc/fstab` needs to be edited. Add the following line to the fstab file of all compute nodes,

```
master:/home/mpiuser /home/mpiuser nfs
```

Again, read the man page of fstab if you want to know the details (`man fstab`). Reboot the compute nodes and list the contents of the `/home/mpiuser` directory on each compute node to check if you have access to the data on the master node,

```
$ ls /home/mpiuser
```

This should lists the files from the `/home/mpiuser` directory of the master node. If it doesn't immediately, wait a few seconds and try again. It might take some time for the system to initialize the connection with the master node.

4.4. Setup passwordless SSH for communication between nodes

For the cluster to work, the master node needs to be able to communicate with the compute nodes, and vice versa ([ubuntuwiki]). Secure Shell (SSH) is usually used for secure remote access between computers. By setting up passwordless SSH between the nodes, the master node is able to run commands on the compute nodes. This is needed to run the MPI daemons on the available compute nodes.

First install the SSH server on all nodes:

```
$ sudo apt-get install ssh
```

Now we need to generate an SSH key for all MPI users on all nodes. The SSH key is by default created in the user's home directory. Remember that in our case the MPI user's home directory (i.e. `/home/mpiuser`) is actually the same directory for all nodes: `/home/mpiuser` on the master node. So if we generate an SSH key for the MPI user on one of the nodes, all nodes will automatically have an SSH key. Let's generate an SSH key for the MPI user on the master node (but any node should be fine),

```
$ su mpiuser
$ ssh-keygen
```

When asked for a passphrase, leave it empty (hence passwordless SSH).

When done, all nodes should have an SSH key (the same key actually). The master node needs to be able to automatically login to the compute nodes. To enable this, the public SSH key of the master node needs to be added to the list of known hosts (this is usually a file `~/.ssh/authorized_keys`) of all compute nodes. But this is easy, since all SSH key data is stored in one location: `/home/mpiuser/.ssh/` on the master node. So instead of having to copy master's public SSH key to all compute nodes separately, we just have to copy it to master's own `authorized_keys` file. There is a command to push the public SSH key of the currently logged in user to another computer. Run the following commands on the master node as user "mpiuser",

```
mpiuser@master:~$ ssh-copy-id localhost
```

Master's own public SSH key should now be copied to `/home/mpiuser/.ssh/authorized_keys`. But since `/home/mpiuser/` (and everything under it) is shared with all nodes via NFS, all nodes should now have master's public SSH key in the list of known hosts. This means that we should now be able to login on the compute nodes from the master node without having to enter a password,

```
mpiuser@master:~$ ssh node1
mpiuser@node1:~$ echo $HOSTNAME
node1
```

You should now be logged in on node1 via SSH. Make sure you're able to login to the other nodes as well.

4.5. Setting up the process manager

In this section I'll walk you through the installation of MPICH and configuring the process manager. The process manager is needed to spawn and manage parallel jobs on the cluster. The MPICH wiki explains this nicely:

> *Process managers are basically external (typically distributed) agents that spawn and manage parallel jobs. These process managers communicate with MPICH processes using a predefined interface called as PMI (process management interface). Since the interface is (informally) standardized within MPICH and its derivatives, you can use any process manager from MPICH or its derivatives with any MPI application built with MPICH or any of its derivatives, as long as they follow the same wire protocol.* [5]
>
> — *Frequently Asked Questions - MPICH*

The process manager is included with the MPICH package, so start by installing MPICH on all nodes with,

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

MPD has been the traditional default process manager for MPICH till the 1.2.x release series. Starting the 1.3.x series, Hydra (http://wiki.mpich.org/mpich/index.php/Hydra) is the default process manager ([mpichfaq]). So depending on the version of MPICH you are using, you should either use MPD or Hydra for process management. You can check the MPICH version by running `mpich2version` in the terminal. Then follow either the steps for MPD **or** Hydra in the following sub sections.

4.5.1. Setting up Hydra

This section explains how to configure the Hydra process manager and is for users of MPICH 1.3.x series and up. In order to setup Hydra, we need to create one file on the master node. This file contains all the host names of the compute nodes ([hydra]). You can create this file anywhere you want, but for simplicity we create it in the the MPI user's home directory,

```
mpiuser@master:~$ cd ~
mpiuser@master:~$ touch hosts
```

In order to be able to send out jobs to the other nodes in the network, add the host names of all compute nodes to the `hosts` file,

```
node1
node2
node3
```

You may choose to include `master` in this file, which would mean that the master node would also act as a compute node. The `hosts` file only needs to be present on the node that will be used to start jobs on the cluster, usually the master node. But because the home directory is shared among all nodes, all nodes will have the `hosts` file.

For more details about setting up Hydra see this page: Using the Hydra Process Manager (http://wiki.mpich.org/mpich/index.php/Using_the_Hydra_Process_Manager).

4.5.2. Setting up MPD

This section explains how to configure the MPD process manager and is for users of MPICH 1.2.x series and down. Before we can start any parallel jobs with MPD, we need to create two files in the home directory of the MPI user. Make sure you're logged in as the MPI user and create the following two files in the home directory,

```
mpiuser@master:~$ cd ~
mpiuser@master:~$ touch mpd.hosts
mpiuser@master:~$ touch .mpd.conf
```

In order to be able to send out jobs to the other nodes in the network, add the host names of all compute nodes to the `mpd.hosts` file,

```
node1
node2
node3
```

You may choose to include `master` in this file, which would mean that the master node would also act as a compute node. The `mpd.hosts` file only needs to be present on the node that will be used to start jobs on the cluster, usually the master node. But because the home directory is shared among all nodes, all nodes will have the `mpd.hosts` file.

The configuration file `.mpd.conf` (mind the dot at the beginning of the file name) must be accessible to the MPI user only (in fact, MPD refuses to work if you don't do this),

```
mpiuser@master:~$ chmod 600 .mpd.conf
```

Then add a line with a secret passphrase to the configuration file,

```
secretword=random_text_here
```

The `secretword` can be set to any random passphrase. You may want to use a random password generator the generate a passphrase.

All nodes need to have the `.mpd.conf` file in the home directory of `mpiuser` with the same passphrase. But this is automatically the case since `/home/mpiuser` is shared through NFS.

The nodes should now be configured correctly. Run the following command on the master node to start the mpd deamon on all nodes,

```
mpiuser@master:~$ mpdboot -n 3
```

Replace "3" by the number of compute nodes in your cluster. If this was successful, all nodes should now be running the mpd daemon. Run the following command to check if all nodes entered the ring (and are thus running the mpd daemon),

```
mpiuser@master:~$ mpdtrace -l
```

This command should display a list of all nodes that entered the ring. Nodes listed here are running the mpd daemon and are ready to accept MPI jobs. This means that your cluster is now set up and ready to rock!

5. Running jobs on the cluster

5.1. Running MPICH2 example applications on the cluster

The MPICH2 package comes with a few example applications that you can run on your cluster. To obtain these examples, download the MPICH2 source package from the MPICH website (http://www.mpich.org/) and extract the archive to a directory. The directory to where you extracted the MPICH2 package should contain an "examples" directory. This directory contains the source codes of the example applications. You need to compile these yourself.

```
$ sudo apt-get build-dep mpich2
$ wget http://www.mpich.org/static/downloads/1.4.1/mpich2-1.4.1.tar.gz
$ tar -xvzf mpich2-1.4.1.tar.gz
$ cd mpich2-1.4.1/
$ ./configure
$ make
$ cd examples/
```

The example application `cpi` is compiled by default, so you can find the executable in the "examples" directory. Optionally you can build the other examples as well,

```
$ make hellow
$ make pmandel
...
```

Once compiled, place the executables of the examples somewhere inside the `/home/mpiuser` directory on the master node. It's common practice to place executables in a "bin" directory, so create the directory `/home/mpiuser/bin` and place the executables in this directory. The executables should now be available on all nodes.

We're going to run an MPI job using the example application `cpi`. Make sure you're logged in as the MPI user on the master node,

```
$ su mpiuser
```

And run the job like this,

When using MPD:

```
mpiuser@master:~$ mpiexec -n 3 /home/mpiuser/bin/cpi
```

When using Hydra:

```
mpiuser@master:~$ mpiexec -f hosts -n 3 /home/mpiuser/bin/cpi
```

Replace "3" by the number of nodes on which you want to run the job. When using Hydra, the `-f` switch should point to the file containing the host names. When using MPD, it's important that you use the **absolute** path to the executable in the above command, because only then MPD knows where to look for the executable on the compute nodes. The absolute path used should thus be correct for all nodes. But since `/home/mpiuser` is the NFS shared directory, all nodes have access to this path and the files within it.

The example application `cpi` is useful for testing because it shows on which nodes each sub process is running and how long it took to run the job. This application is however not useful to test performance because this is a very small application which takes only a few milliseconds to run. As a matter of fact, I don't think it actually computes pi (http://en.wikipedia.org/wiki/Pi). If you look at the source, you'll find that the value of pi is hard coded into the program.

5.2. Running bioinformatics tools on the cluster

By running actual bioinformatics (http://en.wikipedia.org/wiki/Bioinformatics) tools you can give your cluster a more realistic test run. There are several parallel implementations of bioinformatics tools that are based on MPI. There are two that I currently know of:

- mpiBLAST (http://www.mpiblast.org/), a parallel implementation of NCBI BLAST (http://en.wikipedia.org/wiki/BLAST).
- ClustalW-MPI (http://www.bii.a-star.edu.sg/achievements/applications/clustalw/index.php), a parallel implementation of Clustal-W (http://en.wikipedia.org/wiki/Clustal).

It would have been nice to test mpiBLAST, but a compilation issue made this difficult. [6] So I ended up testing with ClustalW-MPI instead.

The MPI implementation of ClustalW is fairly out-dated, but it's good enough to perform a test run on your cluster. Download the source from the website, extract the package, and compile the source. Copy the resulting executable to the `/home/mpiuser/bin` directory on the master node. Use for example Entrez (http://www.ncbi.nlm.nih.gov/Entrez/) to search for some DNA/protein sequences and put these in a single FASTA file (http://en.wikipedia.org/wiki/FASTA_format) (the NCBI website can do that for you). Create several FASTA files with multiple sequences to test with. Copy the multi-sequence FASTA files to a data directory inside mirror (e.g. `/home/mpiuser/data`). Then run a job like this,

When using MPD:

```
mpiuser@master:~$ mpiexec -n 3 /home/mpiuser/bin/clustalw-mpi
/home/mpiuser/data/seq_tyrosine.fasta
```

When using Hydra:

```
mpiuser@master:~$ mpiexec -f hosts -n 3 /home/mpiuser/bin/clustalw-mpi
/home/mpiuser/data/seq_tyrosine.fasta
```

and let the cluster do the work. Again, notice that we must use absolute paths. You can check if the nodes are actually doing anything by logging into the nodes (`ssh node*`) and running the `top` command. This should display a list of running processes with the processes using the most CPU on the top. In this case, you should see the process `clustalw-mpi` somewhere along the top.

6. Credits

Thanks to Reza Azimi for mentioning the `nfs-common` package.

7. References

- [brown2004] Robert G. Brown. *Engineering a Beowulf-style Compute Cluster*. 2004. Duke University Physics Department. http://www.phy.duke.edu/~rgb/Beowulf/beowulf_book/beowulf_book/index.html.
- [wong2008] Kerry D. Wong. *A Simple Beowulf Cluster*. http://www.kerrywong.com/2008/11/04/a-simple-beowulf-cluster/.

- [ubuntuwiki] *Setting Up an MPICH2 Cluster in Ubuntu*. https://help.ubuntu.com/community/MpichCluster.
- [linuxcom] *Building a Beowulf Cluster in just 13 steps*. https://www.linux.com/community/blogs/133-general-linux/9401.
- [mpichfaq] *Frequently Asked Questions - MPICH*. http://wiki.mpich.org/mpich/index.php/Frequently_Asked_Questions.
- [hydra] *Using the Hydra Process Manager - MPICH*. http://wiki.mpich.org/mpich/index.php/Using_the_Hydra_Process_Manager.

---

1. Wikipedia. *Beowulf cluster*. 28 February 2011. http://en.wikipedia.org/wiki/Beowulf_cluster.
2. OpenClusterGroup. *OSCAR*. http://svn.oscar.openclustergroup.org/trac/oscar.
3. Rocks. http://www.rocksclusters.org/
4. Clustering Tutorial. http://www.scfbio-iitd.res.in/doc/clustering.pdf.
5. Frequently Asked Questions - MPICH. http://wiki.mpich.org/mpich/index.php/Frequently_Asked_Questions
6. mpiBLAST-Users. http://lists.mpiblast.org/pipermail/users_lists.mpiblast.org/2011-February/001248.html

Version 1.1
Last updated 2015-01-19 21:38:37 CET