The Beowulf HOWTO

Kurt Swendson

<lam32767@lycos.com>

2004−05−17

Revision History

Revision 1.0	2005−01−08	
first official release		
Revision 0.9	2004−05−17	Revised by: 01
initial revision		

This document describes step by step instructions on building a Beowulf cluster. This is a Red Hat and LAM specific version of this document.

Table of Contents

1. Introduction.....1
1.1. Copyright and License.....1
1.2. Disclaimer.....1
1.3. Credits / Contributors.....1
1.4. Feedback.....2

2. Definitions.....3

3. Requirements.....4

4. Set Up The Head Node.....5
4.1. Hosts.....5
4.2. Groups.....6
4.3. NFS.....6
4.4. IP Addresses.....6
4.5. Services.....6
4.6. SSH.....7
4.7. MPI.....8

5. Set Up Slave Nodes.....9
5.1. Base Linux Install.....9
5.2. Hardware.....9
5.3. Post Install Commands.....9
5.4. SSH On Slave Nodes.....10
5.5. NFS Settings On Slave Nodes.....10
5.6. Lilo Modifications On Slave Nodes.....10

6. Verification.....12

7. Run A Program.....13

1. Introduction

This document describes step by step instructions on building a Beowulf cluster. After seeing all of the documentation that was available, I felt there were enough gaps and omissions that my own document, which I believe accurately describes how to build a Beowulf cluster, would be beneficial.

I first saw Thomas Sterling's article in Scientific American, and immediately got the book, because its title was "How to Build a Beowulf". No doubt, it was a valuable reference, but it does not walk you through instructions on exactly what to do.

What follows is a description of what I got to work. It is only one example − my example. You may choose a different message passing interface; you may choose a different Linux distribution. You may also spend as much time as I did researching and experimenting, and learn on your own.

1.1. Copyright and License

This document, *The Beowulf HOWTO*, is copyrighted (c) 2004 by *Kurt Swendson*. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front−Cover Texts, and with no Back−Cover Texts. A copy of the license is available at http://www.gnu.org/copyleft/fdl.html.

Linux is a registered trademark of Linus Torvalds.

1.2. Disclaimer

No liability for the contents of this document can be accepted. Use the concepts, examples and information at your own risk. There may be errors and inaccuracies which could damage to your system. Though this is highly unlikely, proceed with caution. The author(s) do not accept responsibility for your actions.

All copyrights are held by their by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark. Naming of particular products or brands should not be seen as endorsements.

1.3. Credits / Contributors

Thanks to Thomas Johnson for all of his support and encouragement and, of course, for the hardware without which I would not have been able to even start.

Thanks to my lovely wife Sharron for her understanding and patience during my many hours spent with "the wolves".

The original Beowulf HOWTO by Jacek Radajewski and Douglas Eadline.

1.4. Feedback

Send your additions, comments and criticisms to `<lam32767@lycos.com>`.

2. Definitions

What is a Beowulf cluster? The authors of the original Beowulf HOWTO, Jacek Radajewski and Douglas Eadline, provide a good definition in their document: "Beowulf is a multi−computer architecture which can be used for parallel computations. It is a system which usually consists of one server node, and one or more client nodes connected together via Ethernet or some other network". The site beowulf.org lists many web pages about Beowulf systems built by individuals and organizations. From these two links, one can be exposed to a large number of perspectives on the Beowulf architecture, and draw his / her own conclusions.

What's the difference between a true Beowulf cluster and a COW [cluster of workstations]? Brahma gives a good definition: http://www.phy.duke.edu/brahma/beowulf_book/node62.html.

If you are a "user" at your organization, and you have the use of some nodes, you may still do the instructions shown here to create a cow. But if you "own" the nodes, that is, if you have complete control of them, and are able to completely erase and rebuild them, you may create a true Beowulf cluster.

In Brahma's web page, he suggests you manually configure each box, and then later on (after you get the feel of doing this whole "wolfing up" procedure), you can set up new nodes automatically (which I will describe in a later document).

---

3. Requirements

Let's briefly outline the requirements:

- More than one box, each equipped with a network card.
- A switch or hub to connect them
- Linux
- A message−passing interface [I used lam]

It is not a requirement to have a kvm switch, [you know, the switch to share one keyboard, video, and mouse between many boxes], but it is convenient while setting up and / or debugging.

4. Set Up The Head Node

So let's get "wolfing." Choose the most powerful box to be the head node. Install Linux there and choose every package you want. The only requirement is that you choose "Network Servers" [in Red Hat terminology] because you need to have NFS and ssh. That's all you need. In my case, I was going to do development of the Beowulf application, so I added X and C development.

It is my experience that you do not actually need NFS, but I found it invaluable for copying files between nodes, and for automating the install process. Later in this document I will describe how you can run a simple Beowulf application without the use of NFS, but a more complex application may use NFS or actually depend upon it.

Those of you researching Beowulf systems will also know how you can have a second network card on the head node so you can access it from the outside world. This is not required for the operation of a cluster.

I learned the hard way: use a password that obeys the strong password constraints for your Linux distribution. I used an easily typed password like "a" for my user, and the whole thing did not work. When I changed my password to a legal password, with mixed numbers, characters, upper and lower case, it worked.

If you use lam as your message passing interface, you will read in the manual to turn OFF the firewalls, because they use random port numbers to communicate between nodes. Here is a rule: If the manual tells you to do something, DO IT! The lam manual also tells you to run as a non−root user. Make the same user for every box. Build every box on the cluster with that same user and password. I named that non root user "wolf".

4.1. Hosts

First we modify /etc/hosts. In it, you will see the comments telling you to leave the "localhost" line alone. Ignore that advice and change it to not include the name of your box in the loopback address.

Modify the line that says:

```
127.0.0.1 wolf00 localhost.localdomain localhost
```

...to now say:

```
127.0.0.1 localhost.localdomain localhost
```

Then add all the boxes you want on your cluster. Note: This is not required for the operation of a Beowulf cluster; only convenient, so that you may type a simple "wolf01" when you refer to a box on your cluster instead of the more tedious 192.168.0.101:

```
192.168.0.100 wolf00
192.168.0.101 wolf01
192.168.0.102 wolf02
192.168.0.103 wolf03
192.168.0.104 wolf04
```

4.2. Groups

In order to responsibly set up your cluster, especially if you are a "user" of your boxes [see Definitions], you should have some measure of security.

After you create your user, create a group, and add the user to the group. Then, you may modify your files and directories to only be accessible by the users within that group:

```
groupadd beowulf
usermod -g beowulf wolf
```

...and add the following to /home/wolf/.bash_profile:

```
umask 007
```

Now any files created by the user "wolf" [or any user within the group] will be automatically only writeable by the group "beowulf".

4.3. NFS

Refer to the following web site: http://www.ibiblio.org/mdw/HOWTO/NFS−HOWTO/server.html

Print that up, and have it at your side. I will be directing you how to modify your system in order to create an NFS server, but I have found this site invaluable, as you may also.

Make a directory for everybody to share:

```
mkdir /mnt/wolf
chmod 770 /mnt/wolf
chown wolf:beowulf /mnt/wolf -R
```

Go to the /etc directory, and add your "shared" directory to the exports file:

```
cd /etc
cat >> exports
/mnt/wolf 192.168.0.100/192.168.0.255 (rw)
<control d>
```

4.4. IP Addresses

My network is 192.168.0.nnn because it is one of the "private" IP ranges. Thomas Sterling talks about it on page 106 of his book. It is inside my firewall, and works just fine.

My head node, which I call "wolf00" is 192.168.0.100, and every other node is named "wolfnn", with an ip of 192.168.0.100 + nn. I am following the sage advice of many of the web pages out there, and setting myself up for an easier task of scaling up my cluster.

4.5. Services

Make sure that services we want are up:

```
chkconfig -add sshd
```

```
chkconfig -add nfs
chkconfig -add rexec
chkconfig -add rlogin
chkconfig -level 3 rsh on
chkconfig -level 3 nfs on
chkconfig -level 3 rexec on
chkconfig -level 3 rlogin on
```

...And, during startup, I saw some services that I know I don't want, and in my opinion, could be removed. You may add or remove others that suit your needs; just include the ones shown above.

```
chkconfig -del atd
chkconfig -del rsh
chkconfig -del sendmail
```

4.6. SSH

To be responsible, we make ssh work. While logged in as root, you must modify the /etc/ssh/sshd_config file. The lines:

```
#RSAAuthentication yes
#AuthorizedKeysFile .ssh/authorized_keys
```

...are commented out, so uncomment them [remove the #].

Reboot, and log back in as wolf, because the operation of your cluster will always be done from the user "wolf". Also, the hosts file modifications done earlier must take effect. Logging out and back in will not do this. To be sure, reboot the box, and make sure your prompt shows hostname "wolf00".

To generate your public and private SSH keys, do this:

```
ssh-keygen -b 1024 -f ~/.ssh/id_rsa -t rsa -N ""
```

...and it will display a few messages, and tell you that it created the public / private key pair. You will see these files, id_rsa and id_rsa.pub, in the /home/wolf/.ssh directory.

Copy the id_rsa.pub file into a file called "authorized_keys" right there in the .ssh directory. We will be using this file later. Verify that the contents of this file show the hostname [the reason we rebooted the box]. Modify the security on the files, and the directory:

```
chmod 644 ~/.ssh/auth*
chmod 755 ~/.ssh
```

According to the LAM user group, only the head node needs to log on to the slave nodes; not the other way around. Therefore when we copy the public key files, we only copy the head node's key file to each slave node, and set up the agent on the head node. This is MUCH easier than copying all authorized_keys files to all nodes. I will describe this in more detail later.

Note: I only am documenting what the LAM distribution of the message passing interface requires; if you chose another message passing interface to build your cluster, your requirements may differ.

At the end of /home/wolf/.bash_profile, add the following statements [again this is lam−specific; your requirements may vary]:

```
export LAMRSH='ssh -x'
ssh-agent sh -c 'ssh-add && bash'
```

4.7. MPI

Lastly, put your message passing interface on the box. As stated in 1.2 Requirements, I used lam. You can get lam from here:

http://www.lam−mpi.org/

...but you can use any other message passing interface or parallel virtual machine software you want. Again, I am showing you what worked for me.

You can either build LAM from the supplied source, or use their precompiled RPM package. It is not in the scope of this document to describe that; I just got the source and followed the directions, and in another experiment I installed their rpm. Both of them worked fine. Remember the whole reason we are doing this is to learn; go forth and learn.

You may also read more documentation regarding LAM and other message passing interface software here.

5. Set Up Slave Nodes

Get your network cables out. Install Linux on the first non−head node. Follow these steps for each non−head node.

5.1. Base Linux Install

Going with my example node names and IP addresses, this is what I chose during setup:

```
Workstation
auto partition
remove all partitions on system
use LILO as the boot loader
put boot loader on the MBR
host name wolf01
ip address 192.168.0.101
add the user "wolf"
same password as on all other nodes
NO firewall
```

The ONLY package installed: network servers. Un−select all other packages.

It doesn't matter what else you choose; this is the minimum that you need. Why fill the box up with non−essential software you will never use? My research has been concentrated on finding that minimal configuration to get up and running.

Here's another very important point: when you move on to an automated install and config, you really will NEVER log in to the box. Only during setup and install do I type anything directly on the box.

5.2. Hardware

When the computer starts up, it will complain if it does not have a keyboard connected. I was not able to modify the BIOS, because I had older discarded boxes with no documentation, so I just connected a "fake" keyboard.

I am in the computer industry, and see hundreds of keyboards come and go, and some occasionally end up in the garbage. I get the old dead keyboard out of the garbage, remove JUST the cord with the tiny circuit board up there in the corner, where the num lock and caps lock lights are. Then I plug the cord in, and the computer thinks it has a complete keyboard without incident.

Again, you would be better off modifying your bios, if you are able to. This is just a trick to use in case you don't have the bios program.

5.3. Post Install Commands

After your newly installed box reboots, log on as root again, and...

- do the same chkconfig commands stated above to set up the right services.
- modify hosts; remove "wolfnn" from localhost, and just add wolfnn and wolf00.

- install lam
- create the /mnt/wolf directory and set up security for it.
- do the ssh configuration

Up to this point, we are pretty much the same as the head node. I do NOT do the modification of the exports file.

Also, do NOT add this line to the .bash_profile:

```
sh -c 'ssh-add && bash'
```

5.4. SSH On Slave Nodes

Recall that on the head node, we created a file "authorized_keys". Copy that file, created on your head node, to the ~/.ssh directory on the slave nodes. The HEAD node will log on the all the SLAVE nodes.

The requirement, as stated in the LAM user manual, is that there should be no interaction required when logging in from the head to any of the slaves. So, copying the public key from the head node into each slave node, in the file "authorized_keys", tells each slave that "wolf user on wolf00 is allowed to log on here without any password; we know it is safe."

However you may recall that the documentation states that the first time you log on, it will ask for confirmation. So only once, after doing the above configuration, go back to the head node, and type ssh wolfnn where "wolfnn" is the name of your newly configured slave node. It will ask you for confirmation, and you simply answer "yes" to it, and that will be the last time you will have to interact.

Prove it by logging off, and then ssh back to that node, and it should just immediately log you in, with no dialog whatsoever.

5.5. NFS Settings On Slave Nodes

As root, enter these commands:

```
cat >> /etc/fstab
wolf00:/mnt/wolf /mnt/wolf nfs rw,hard,intr 0 0
<control d>
```

What we did here was automatically mount the exported directory we put in the /etc/exports file on the head node. More discussion regarding nfs later in this document.

5.6. Lilo Modifications On Slave Nodes

Then modify /etc/lilo.conf.

The 2nd line of this file says

```
timeout=nn
```

Modify that line to say:

```
timeout=1200
```

After it is modified, we invoke the changes. You type "/sbin/lilo", and it will display back "added linux *" to confirm that it took the changes you made to the lilo.conf file:

```
/sbin/lilo
Added linux *
```

Why do I do this lilo modification? If you were researching Beowulf on the web, and understand everything I have done so far, you may wonder, "I don't remember reading anything about lilo.conf."

All my Beowulf nodes share a single power strip. I turn on the power strip, and every box on the cluster starts up immediately. As the startup procedure progresses, it mounts file systems. Seeing that the non−head nodes mount the shared directory from the head node, they all will have to wait a little bit until the head node is up, with NFS ready to go. So I make each slave node wait 2 minutes in the lilo step. Meanwhile, the head node comes up, and making the shared directory available. By then, the slave nodes finally start booting up because lilo has waited 2 minutes.

---

6. Verification

All done! You are almost ready to start wolfing.

Reboot your boxes. Did they all come up? Can you ping the head node from each box? Can you ping each node from the head node? Can you ssh? Don't worry about doing ssh as root; only as wolf. Also only worry about ssh from the head to the slave, not the other way around.

If you are logged in as wolf, and ssh to a box, does it go automatically, without prompting for password?

After the node boots up, log in as wolf, and say "mount". Does it show wolf00:/mnt/wolf mounted? On the head node, copy a file into /mnt/wolf. Can you read and write that file from the slave node?

This is really not required; it is merely convenient to have a common directory reside on the head node. With a common shared directory, you can easily use scp to copy files between boxes. Sterling states in his book, on page 119, a single NFS server causes a serious obstacle to scaling up to large numbers of nodes. I learned this when I went from a small number of boxes up to a large number.

---

7. Run A Program

Once you can do all the tests shown above, you should be able to run a program. From here on in, the instructions are lam specific.

Go back to the head node, log in as wolf, and enter the following commands:

```
cat > /nnt/wolf/lamhosts
wolf01
wolf02
wolf03
wolf04
<control d>
```

Go to the lam examples directory, and compile "hello.c":

```
mpicc -o hello hello.c
cp hello /mnt/wolf
```

Then, as shown in the lam documentation, start up lam:

```
[wolf@wolf00 wolf]$ lamboot -v lamhosts
LAM 7.0/MPI 2 C++/ROMIO - Indiana University
n0<2572> ssi:boot:base:linear: booting n0 (wolf00)
n0<2572> ssi:boot:base:linear: booting n1 (wolf01)
n0<2572> ssi:boot:base:linear: booting n2 (wolf02)
n0<2572> ssi:boot:base:linear: booting n3 (wolf04)
n0<2572> ssi:boot:base:linear: finished
```

So we are now finally ready to run an app. [Remember, I am using lam; your message passing interface may have different syntax].

```
[wolf@wolf00 wolf]$ mpirun n0-3 /mnt/wolf/hello
Hello, world! I am 0 of 4
Hello, world! I am 3 of 4
Hello, world! I am 2 of 4
Hello, world! I am 1 of 4
[wolf@wolf00 wolf]$
```

Recall I mentioned the use of NFS above. I am telling the nodes to all use the nfs shared directory, which will bottleneck when using a larger number of boxes. You could easily copy the executable to each box, and in the mpirun command, specify node local directories: mpirun n0–3 /home/wolf/hello. The prerequisite for this is to have all the files available locally. In fact I have done this, and it worked better than using the nfs shared executable. Of course this theory breaks down if my cluster application needs to modify a file shared across the cluster.