Scyld Beowulf Series 30

Administrator’s Guide

Scyld Software

Scyld Beowulf Series 30: Administrator's Guide

by Scyld Software
Series 30 Edition
Published Feb 2006

Table of Contents

Preface

Welcome to the Scyld Beowulf Cluster Operating System Administrator's Guide. It is written for use by the Scyld Beowulf cluster administrators and advanced users. This document covers not only cluster configuration, but also how the system operates internally. It will go into properly maintaining and optimizing the cluster. As is true with any Linux-type operating system, the administrator must have root privileges to perform the administrative tasks described in this document.

The beginning of this guide describes the Scyld Beowulf system and its design. The system architecture and design are critical to understand in order to fully configure and administer the cluster. The guide then goes further into specific information about the tools and ways to setup and maintain the cluster, the cluster boot process, cluster file systems and configuration, what file systems can be used, how to partition them, how to control cluster usage, how to batch jobs and control the job queue, a description of how load balancing is handled in the cluster, and coverage of extra optional tools which can be useful in administrating your cluster. Finally, there is an appendix which covers the important files and directories that pertain to operation of the Scyld Beowulf cluster.

This guide is written with the assumption that the administrator has a background in a Unix or Linux operating environment and therefore does *not* cover basic Linux system administration. If you do not have knowledge on how to properly use or administer a Linux system, it is highly recommended that you first consult *Linux in a Nutshell* and other very useful books published by *O'Reilly and Associates*[1].

This guide does not cover the initial installation of the Scyld Beowulf software. The *Installation Guide* goes into explicit detail on how to setup the master node and boot the compute nodes. It is suggested that the administrator who is new to the Beowulf concept should read the *User's Guide* first, as it introduces Beowulf computing concepts.

Feedback

We welcome any reports on problems that you may find. We also would like your suggestions on improving this document. Please direct all comments and suggestions to: <`support@scyld.com`>.

When writing your e-mail, please be as specific as possible. Include the chapter and section information. Also, mention in which version of the Administrator's Guide you found the error. This version is Series 30, published Feb 2006.

Notes

1. http://www.oreilly.com

Chapter 1. System Design Description

This chapter presents a description of the design behind the Scyld Beowulf Cluster Operating System. The discussion begins with a high-level description of the system architecture as it relates to the cluster as a whole. From there it moves down a level and discusses the software components resident on the master node and compute nodes. After these baseline issues have been explained, the chapter builds on this foundation by describing such topics as the cluster's process migration technology and the compute node boot procedure. Finally, the chapter describes the remaining cluster specific software components available in the Scyld distribution.

As mentioned in the preface, this document assumes a certain level of knowledge from the reader and therefore, does not cover any system design decisions related to a basic Linux system. In addition, it is assumed the reader has a general understanding of Beowulf clustering and how the second generation Scyld Beowulf system differs from the traditional Beowulf. For more information of these topics, see the *Scyld Beowulf Overview* chapter in the *User's Guide*. This chapter tries *not* to discuss topics dealt with elsewhere in the Scyld documentation set. When appropriate, this chapter refers the reader to other sources for more detailed explanations on the topic at hand.

Scyld Beowulf System Architecture

The Scyld Beowulf Cluster Operating System provides a software solution designed specifically for Beowulf clusters. The Scyld distribution streamlines the process of configuring, administering, running and maintaining a Beowulf-class cluster computer. It was developed with the goal of providing the operating system software infrastructure for commercial production cluster systems. The following figure is a system level block diagram of a Scyld Beowulf.

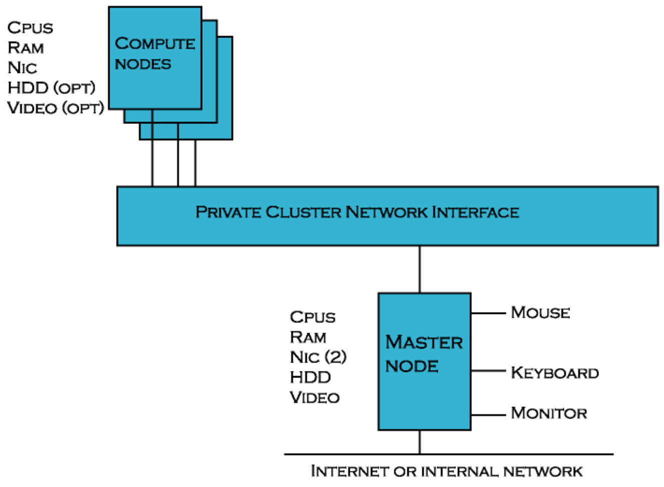


Figure 1-1. Scyld Beowulf System Block Diagram

A quick examination of the system block diagram shown above identifies the three primary components of Scyld Beowulf: 1) the master node; 2) compute nodes; and, 3) the cluster's private network interface. The remaining element in the diagram is the public/building network interface connected to the master node. This network connection is *not* required for the cluster to operate properly and may not even be connected (for example, for security reasons).

The master node and compute nodes have different roles in Scyld Beowulf and therefore have different hardware requirements. The master node is the central administration console for the cluster and is the machine all users of the cluster login to for starting their jobs. The master node is then responsible for sending these jobs out to the appropriate compute node(s) for execution. The master node also performs all the standard tasks of a Linux machine such as queuing print jobs or running shells for individual users. Given the role of the master node, it's easy to see why its hardware closely resembles that of a standard Linux machine.

The master node will typically have the standard human user interface devices such as a monitor, keyboard and mouse. It may have a fast 3D video card depending on the cluster's application. The master is usually equipped with two network interface cards, or NICs. One card connects the master to the cluster's compute nodes over the private cluster network, and the other NIC connects the master to the *outside world*. The master should be equipped with enough hard disk space to satisfy the demands of its users and the applications it must execute. The OS itself uses about 3GB of disk space. Any network attached storage should be connected to both the private cluster network and the public network through *separate* interfaces. The master node should contain at least 256MB of RAM. You should have enough RAM to avoid swap during normal operations. Having to swap programs to disk will degrade performance significantly, and RAM is relatively cheap.

In contrast, the compute nodes are single purpose machines. Their role is to run the jobs sent to them by the master node. If the cluster is viewed as a single, large-scale, parallel computer, then the compute nodes are its CPU and memory resources. They don't have any login capabilities and aren't running any of the daemons typically found on a standard Linux box. Since it's impossible to login to a compute node, they don't need a monitor, keyboard or mouse. The hard drive is *not* a required component for a compute node, but must be used for swap space if all data will not fit into physical memory. The amount of RAM in a compute node is typically sized to avoid swap usage. Many users have installed 2 GB RAM and create a large RAMdisk to hold their database or other large data files. This avoids using the much slower disk drives for file accesses. Video cards aren't required for compute nodes (but may be required by the BIOS) either, but having an inexpensive one installed may prove cost effective when debugging hardware problems. Another hardware debug solution is to use a serial port connection back to the master node from the compute nodes. The kernel command line options for a compute node can be configured to display all boot information to the serial port.

Network Topologies

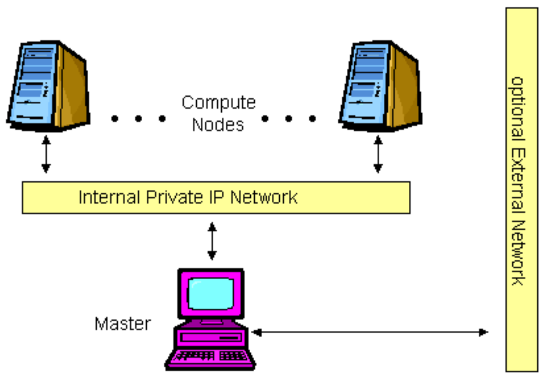


Figure 1-2. Minimal Network Configuration

Scyld Beowulf requires at least one IP network be installed to enable master and compute node communications. This network can range in speed from 10MB/s (Fast Ethernet) to over 1GB/s depending on cost and performance requirements. For many applications an inexpensive Ethernet network is all that is needed. Other applications might require multiple networks to obtain the best performance. These applications generally fall into two categories, message and server intensive.

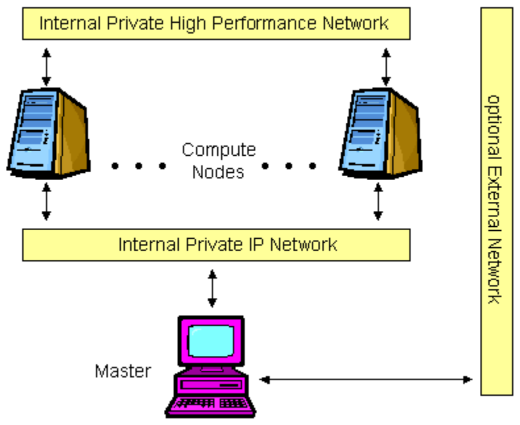

Internal Private High Performance Network
Compute Nodes
Internal Private IP Network
Master
optional External Network

Figure 1-3. Performance Network Configuration

The performance network configuration is intended for applications that can benefit from the low message latency of proprietary networks like Myrinet, Dolphin SCI or Infiniband. These networks can optionally run without the overhead of an IP stack with direct memory to memory messaging. Here the lower bandwidth requirements of the Scyld software run over a standard IP network, freeing the other network from any OS related overhead completely. It should be noted that these high performance interfaces may also run an IP stack, in which case they may also be used in the other configurations as well.

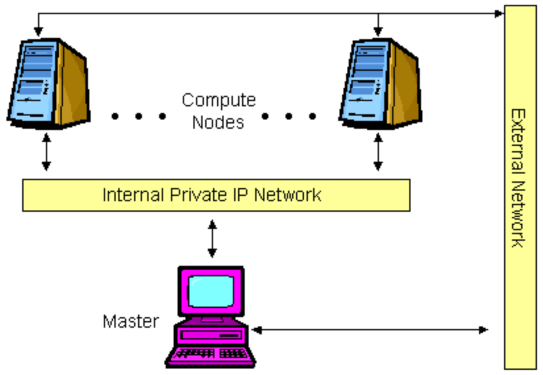


Figure 1-4. Server Network Configuration

The server configuration is intended for web, database, or application servers. In this configuration, each compute node has multiple network interfaces, one for the private control network and one or more for the external public networks. The Scyld Beowulf security model is well suited for this configuration. Even though the compute nodes have a public network interface, there is no way to log into them. There is no `/etc/passwd` file or other configuration files to hack. There are no shells on the compute nodes to execute user programs. The only open ports on the public network interface are the ones your specific application opened. To maintain this level of security, you may wish to have the master node on the internal private network only. The setup for this configuration is not described in this guide, because it is very dependent on your target deployment. Contact Scyld's technical support for help on this kind of configuration.

System Data Flow

The following data flow diagram shows the primary messages sent over the private cluster network between the master and compute nodes in a Scyld Beowulf cluster.

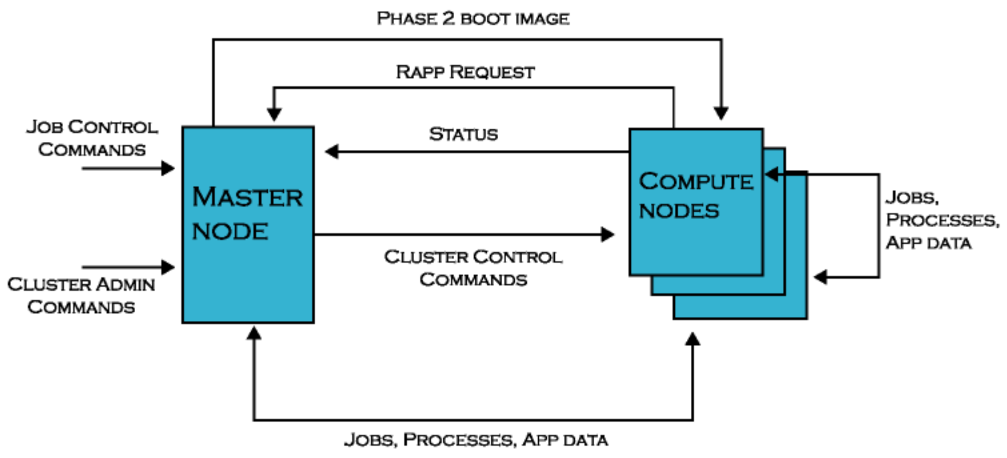


Figure 1-5. Scyld Beowulf Data Flow Diagram

Master Node to Compute Node

The following is a list of the data items sent from the master node to a compute node as depicted in the previous diagram.

- *cluster control commands* - these are the "commands" sent from the master to the compute node telling it to perform such tasks as rebooting, halting, powering off...
- *jobs, processes, app data* - these include the process snapshots captured by BeoMaster for migrating processes between nodes as well as the application data sent between jobs
- *phase 2 boot images* - the phase 2 boot image is sent from the master to a compute node in response to its RARP requests during its boot procedure
- *user job commands, cluster admin commands* - this information is somewhat out of place in this section, as these commands are not sent from the master to the compute nodes; instead these items represent "inputs" to the master from the user / administrator

Compute Node to Master Node

The following is a list of the data items sent from a compute node to the master node as depicted in the previous diagram.

- *RARP requests* - these requests are sent to the master from a compute node while it's booting; in response, the master replies back with the node's IP address and the Phase 2 boot image.
- *jobs, processes, app data* - these include the process snapshots captured by BeoMaster for migrating processes between nodes as well as the application data sent between jobs
- *node status* - all the compute nodes in a Scyld Beowulf send periodic status information back to the master

Compute Node to Compute Node

The following is a list of the data items sent between compute nodes as depicted in the previous diagram.

- *jobs, processes, app data* - these are both the process snapshots captured by BeoMaster for migrating processes between nodes as well as the application data sent between jobs

System Software Context

The following diagram shows which software components are available on the nodes in a Scyld Beowulf.

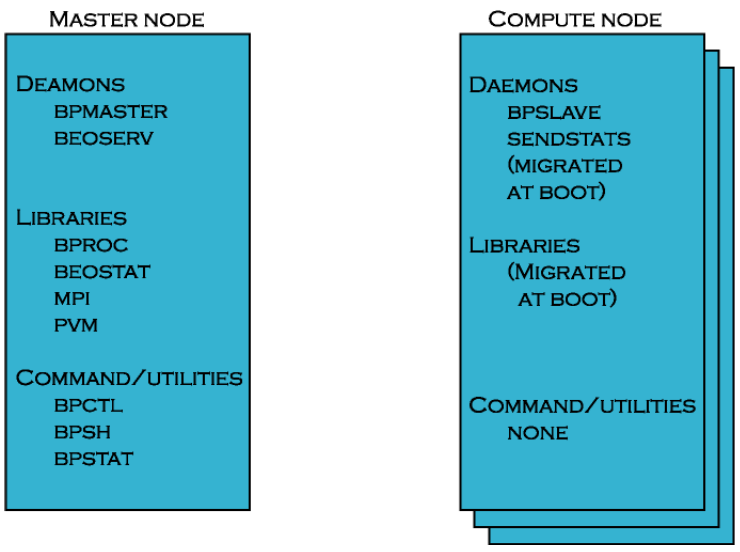


Figure 1-6. System Software Context Diagram

Master Node Software Components

As shown in the diagram above, the *daemons* running on the master node are **bpmaster** and **beoserv**. The Scyld-specific libraries on the master node are `libbproc`, `libbeostat`, `libmpi`, and `libpvm`. Finally, the commands and utilities shown in the above diagram are a small subset of all the software tools available on the master node.

Compute Node Software Components

As shown in the diagram above, the *daemons* running on the compute nodes are **bpslave** and **sendstats**. These *daemons* are migrated out to the compute node from the master at boot time. All of the libraries resident on the compute node, are also migrated from the master at boot time. Finally, there are no binaries on the compute node for running commands. Any command that's to be run on the compute node, must be migrated at runtime using a command such as **bpsh**.

System Level Files

The following two sections briefly describe the system level files found on the master and compute nodes in a Scyld Beowulf.

Master Node Files

The file layout on a master node is quite similar to the file layout on most Linux/UNIX boxes. For those who are not familiar with the file layout that is commonly used by Linux distributions, here are some things to keep in mind.

`/bin`, `/usr/bin`

directories with user level command binaries

`/sbin`, `/usr/sbin`

directories with administrator level command binaries

`/lib`, `/usr/lib`

directories with static and shared libraries

`/usr/include`

directory with include files

`/etc`

directory with configuration files

`/var/log`

directory with system log files

`/usr/share/doc`

directory with various documentation files

Scyld Beowulf also has some special directories on the master node that are useful to know about. The status logs for compute nodes are stored in `/var/log/beowulf`. The individual files are named `node.#`, where `#` is replaced by the node's identification number as reported by the *BeoSetup* tool, discussed later in this document.

Configuration files for Scyld Beowulf are found in `/etc/beowulf`. The directory `/usr/lib/beoboot/bin` contains the **node_up** script and various helper scripts that are used to configure compute nodes during boot.

For more information on the special directories, files and scripts used by Scyld Beowulf, see *Appendix A* in this document as well as the *Reference Guide*.

Compute Node Files

Compute nodes only have a very small set of files that exist on them. For the most part, these files are all dynamic libraries and almost no actual binaries. For a detailed list of exactly what files do exist on the compute nodes, please see *Appendix A* in this document.

Scyld Beowulf Technical Description

The following sections discuss some of the technical details behind a Scyld Beowulf such as the compute node boot procedure and the BeoMaster process migration software. The individual sections provide references to other Scyld documents for additional information.

Compute Node Boot Procedure

The compute nodes in a Scyld Beowulf cluster boot using a two-stage procedure. Compute nodes begin their boot process using a local, minimal *stage 1* boot image, after which they contact the master node to obtain their final *stage 2* boot image. Stage 1 boot images are approximately 1Mbyte in size and contain a minimal Linux kernel with just enough functionality to configure a reliable TCP/IP connection between the compute node and the master node. Scyld Beowulf supports the following types of boot media for the stage 1 boot image: floppy disk, CDROM, DVD, LinuxBIOS, flash disk, hard-disk and PXE boot. *PXE boot is the preferred booting method in Scyld Series 29.*

Once the stage 1 image is booted, the compute node attempts to communicate with the master to obtain its required runtime files and complete its initialization procedure. The compute node uses an extended version of the Reverse Address Request Protocol (RARP) to contact the master node. This RARP request, which is broadcast on all network interfaces until a reply is received, includes the compute node's unique Ethernet MAC address along with other information about the system. After the master node validates the compute node's Ethernet address and verifies the node is *officially* part of the cluster, it replies back to the slave with its IP address and a fully functional stage 2 kernel. The compute node then switches to this downloaded stage 2 kernel using the *Two Kernel Monte* boot technology developed by Scyld. Finally, shared runtime libraries are cached on the compute node from the master and an initial daemon (**bpslave**) is started on the compute node.

A more detailed explanation of the cluster boot procedure can be found in the chapter on *Booting the Cluster* in this document. The chapter describes how to build and specify the boot images used during each phase of the boot process.

BProc, Beowulf Distributed Process Space

Scyld BeoMaster is able to provide a single system image through its use of BProc, the Beowulf process space management kernel enhancement. *BProc* enables the processes running on cluster compute nodes to be visible and manageable on the master node. Processes start on the master node and are migrated to the appropriate compute node by *BProc Process Migration* code. Process parent-child relationships and UNIX job control information are both maintained with migrated jobs:

- all processes appear in the master node's process table
- all standard UNIX signals (kill, suspend, resume...) can be sent to any process on a slave node from the master node
- *stdin, stdout* and *stderr* output from jobs is redirected back to the master through a socket

BProc is one of the primary features that makes Scyld Beowulf different from traditional Beowulfs. It’s the key software component to making slave nodes appear as attached computational resources to the master node. The figure shown below depicts the role BProc plays in a Scyld Beowulf.

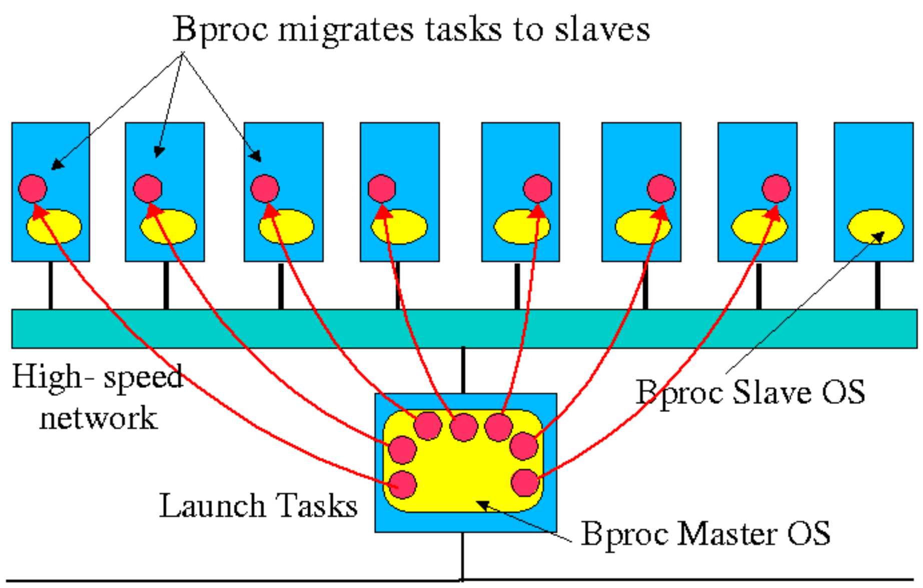


Figure 1-7. BProc Data Flows in a Scyld Beowulf Cluster

BProc itself is divided into two components: **bpmaster**, a daemon program that runs on the master node at all times; and, **bpslave**, a daemon program which runs on each of the compute nodes. The user of a Scyld Beowulf cluster will never directly need to run or interact with these daemons, however, their presence greatly simplifies the task of running parallel jobs on a Scyld Beowulf.

VMADump is the module used by the **bpmaster** daemon to freeze a running process so it can be transferred to a remote node. The same **VMADump** module is also used by the **bpslave** daemon to thaw the process after it’s been received. In a nutshell, the **VMADump** module saves or restores a process’s memory space to or from a stream. In the case of BProc, the stream is a TCP socket connected to the remote machine.

VMADump implements an optimization which greatly reduces the size of the memory space required for storing a frozen process. Most programs on the system are dynamically linked. At run time, they will use **mmap** to map copies of various libraries into their memory spaces. Since these libraries are *demand* paged, the entire library is always mapped even if most of it will never be used. These regions must be included when copying a process’s memory space and again, when the process is restored. This is expensive since the C library dwarfs most programs in size.

Here is an example of the memory space for the program **sleep**. This is taken directly from `/proc/pid/maps`.

```
08048000-08049000 r-xp 00000000 03:01 288816     /bin/sleep
08049000-0804a000 rw-p 00000000 03:01 288816     /bin/sleep
40000000-40012000 r-xp 00000000 03:01 911381     /lib/ld-2.1.2.so
40012000-40013000 rw-p 00012000 03:01 911381     /lib/ld-2.1.2.so
```

```
40017000-40102000 r-xp 00000000 03:01 911434     /lib/libc-2.1.2.so
40102000-40106000 rw-p 000ea000 03:01 911434     /lib/libc-2.1.2.so
40106000-4010a000 rw-p 00000000 00:00 0
bfffe000-c0000000 rwxp fffff000 00:00 0
```

The total size of the memory space for this trivial program is 1089536 bytes. All but 32K of that comes from shared libraries. **VMADump** takes advantage of this. Instead of storing the data contained in each of these regions, it stores a reference to the regions. When the image is restored, those files will be **mmap**'ed to the same memory locations.

In order for this optimization to work, **VMADump** must know which files it can expect to find in the location where they are restored. **VMADump** has a list of files it presumes are present on remote systems. The **vmadlib** utility exists to manage this list.

Compute Node Categories

Each compute node in the cluster is classified into one of three categories by the master node: *unknown*, *ignored* or *configured*. As described below, the classification of a node is dictated by where, or if, it's listed in one of the following two files:

1. the unknown addresses file, `/var/beowulf/unknown_addresses`
2. the cluster configuration file, `/etc/beowulf/config`

An *unknown* node is one not formally recognized by the cluster as being either a *configured* node or an *ignored* node. When a compute node completes its phase 1 boot process, it begins to send out RARP requests on all the network interface devices that it finds. When the master node receives a RARP request from a node not already listed as *configured* or *ignored* in the cluster configuration file, it considers the request to be from an *unknown* node and subsequently adds the node to the *unknown addresses file*. When bringing a new compute node online, or after replacing an existing node's network interface card, the node will be classified as *unknown*.

An *ignored* node is one that is listed in the cluster configuration file using the `ignore` tag. These are typically nodes that for one reason or another you'd like the master node to simply ignore. They are not considered part of the cluster and will not receive the appropriate responses from the master during their boot process. As an example, you may choose to temporarily reclassify a node as *ignored* while performing hardware maintenance activities when the node may be rebooting frequently.

A *configured* node is one that is listed in the cluster configuration file using the `node` tag. These are nodes that are formally part of the cluster and recognized as such by the master node. When running jobs on your cluster, these are the nodes actually used as computational resources by the master.

When first building and configuring your cluster, all nodes will be initially classified as *unknown*. It requires user interaction to change the classification of a node and that interaction can only be done by the cluster administrator. For more information, see the *BeoSetup* chapter and the appendix on *Special Directories, Configuration Files and Scripts* in this document.

Compute Node States

Cluster compute nodes may be in any of several functional states, such as (*down*, *up*, or *unavailable*). Some of these states are transitional (*boot* or *shutdown*), or informational variants of the *up* state (*unavailable*, and *error*). BProc really needs to handle only 3 node operational variations:

1. The node is not communicating: *down*. Variations of the down state may record the reason, such as *halted* (known to be halted by the master) or *reboot* (the master shut down the node with a reboot command).
2. The node is communicating: *up*, *unavailable*, *error*. Here the strings indicate different levels of usability.

3. The node is transitioning: *boot*. This state has varying levels of communication, operating on scripted sequence.

During a normal power on sequence, the user will see the node status change from *down*, to *boot*, to *up*. Depending on the machine speed, the *boot* phase may be very short and not visible due to the update rate of **beostatus**. All state information is reset to *down* whenever the *bpmaster* daemon is started/restarted.

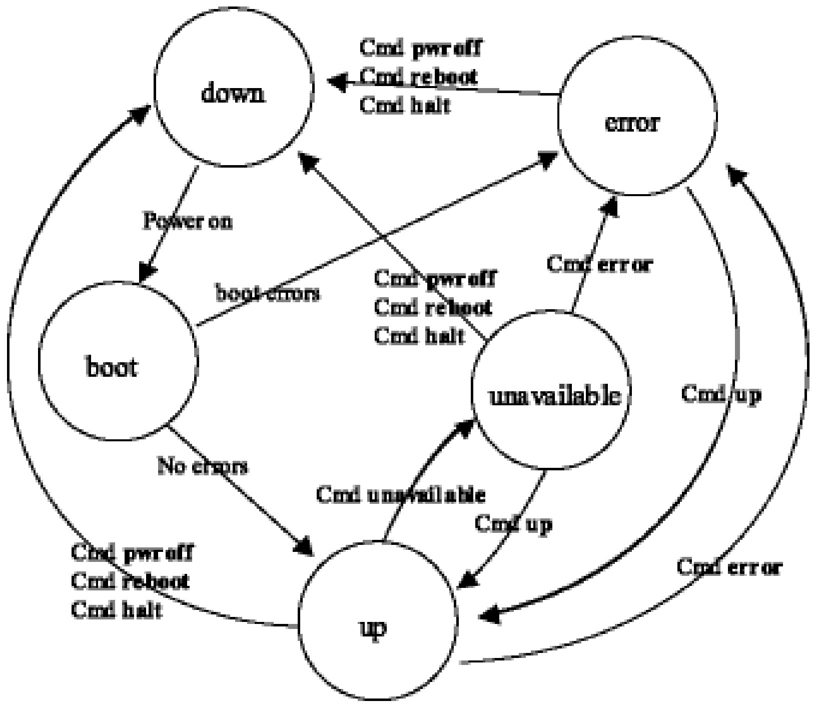


Figure 1-8. Node State Transition Diagram

down

From the master node's view, *down* means only that there is no communication with the compute node. A node is *down* when it is powered off, has been halted, been rebooted, has a network link problem, or has some other hardware problem that prevents communication.

boot

This is a transitional state during which the node will not accept user commands. The *boot* state is set when the **node_up** script has started and will transition to *up* or *error* when the script has completed. While in this state, the node will respond to administrator commands, but indicates that the node is still being configured for normal operation. The duration of this state varies with the complexity of the *node_up* script.

up

This is a functional state. The *up* state is set when the **node_up** script has completed without encountering any errors. BProc checks the return status of that script and sets the node state to *up* if the script was successful. This is the only state where the node is available to normal users, as BProc checks this before moving any program to a node. Administrator programs bypass this check. This state may also be commanded when the previous state was *unavailable* or *error*.

error

This is an informational state. The *error* state is set when the **node_up** script has exited with errors. The administrator may access the node, or look in the `/var/log/beowulf/node.x` to determine the problem. When a problem is seen to be non-critical, the administrator may then set the node to *up*.

unavailable

This is a functional state. For the administrator, the node is completely available for use. The node is just *unavailable* to users. Currently running jobs will not be affected by a transition to this state. With respect to job control, only when attempting to run new jobs, does this state come into play, as new jobs will fail to migrate to a node marked with this state. This state is intended to allow node maintenance without having to bring the node offline.

Miscellaneous Components

This section briefly touches on a few technical areas that didn't really fall into a specific category but were still worthy of mention.

BeoNSS

BeoNSS provides name service lookup functionality for the Scyld Beowulf cluster. The information it provides includes:

Hostnames

This provides dynamically generated hostnames for all the nodes in the cluster. The hostnames are of the form `.<nodenumber>`, so the hostname for node 0 would be `.0`, the hostname for node 50 would be `.50`, and the hostname for the master node would be `.-1`. It also provides the hostname `master`, to always point to the IP of the master node on the beowulf cluster's internal network. The hostnames `.-1` and master will always point to the same IP. These hostnames will always point to the right IP address based on how your IP range is configured. You don't need to do anything special for these hostnames to work. Also, these hostnames will work on the master node, or any of the compute nodes.

There is also the hostname `self`. This hostname always points to the IP of the node it's run on. So, `self` on node 0 is the same as `.0`, and on the master node, it is the same as `.-1`.

Also note that BeoNSS does not know the hostname and IP address that the master node uses for the outside network. If your master node has the name `mycluster` and uses the IP address 1.2.3.4 for the outside network, when you are on a slave node you will be unable to open a connection to `mycluster` or 1.2.3.4, however you will be able to connect to `master` or `.-1`

Netgroups

Netgroups are a concept from NIS. They make it easy to specify an arbitrary list of machines, then treat all those machines the same when doing things such as specifying what machines you want to export NFS filesystems to. BeoNSS creates one netgroup called 'cluster' that includes all of the nodes in the cluster. This is used in our default `/etc/exports` file in order to easily export `/home` to all of the compute nodes.

User information

When running jobs on the compute nodes, BeoNSS allows the standard `getpwnam()` and `getpwuid()` functions to successfully retrieve information, such as username, home directory, shell, uid, as long as they are retrieving information on the user that is running the program. All other information that `getpwnam()` and `getpwuid()` normally retrieve will be set to `NULL`.

BeoNSS currently retrieves the user information by looking at the environment that BProc sent with the job to the compute node. This is not an optimal solution, but still allows programs that rely on this information to run. The backend for how BeoNSS retrieves this information may change in future versions of Scyld Beowulf.

IP Communications Ports

Scyld Beowulf uses a few TCP/IP and UDP/IP communication ports when sending information between nodes. Normally, this should be completely transparent to the user. However if the cluster is using a switch that blocks various ports, it may be important to know which ports are being used for what.

The **beoserv** daemon is responsible for replying to the RARP request from a compute node when it's booting. The reply includes a new kernel, the kernel command line options and a small stage 2 boot ramdisk. Beoserv supports both multicast and unicast file serving. By default, **beoserv** uses TCP port 1556. This can be overridden by changing the value of the **server** directive in the `/etc/beowulf/config` to the desired transport mode and port number.

BProc provides the service that migrates jobs out to the compute nodes. By default, BProc uses TCP port 2223.

The **beostat** utility is the service that monitors cluster status. It uses multicast address 224.27.0.1 and 224.27.0.2 on UDP port 3040.

Caching Libraries

One of the features Scyld Beowulf uses to improve the performance of transferring jobs to and from compute nodes is to cache libraries. When BProc needs to migrate a job between nodes, it uses **VMADump** to take a snapshot of all the memory the process is using (including the binary and shared libraries). This memory snapshot is then sent across the private cluster network during process migration. **VMADump** takes advantage of the fact that libraries are being cached on the compute nodes by not including the shared library data in the snapshot thus reducing the amount of information that needs to be sent during process migration. In this way, Scyld Beowulf is able to speed up cluster operations by not having to waste network traffic by continually sending over the libraries with each process.

Accessing External Data

There are two common ways for processes running on a compute node to access data stored externally to the cluster. The first way is to transfer the data to the master node using something such as **scp** or **ftp**, then treating it as any other file that resides on the master node.

The second method involves accessing the data through a network filesystem such as NFS or AFS. Any remote filesystem mounted on the master node, can't be re-exported to the compute node. Therefore, you will have to use another method to access the data on the compute nodes. One way around this is to use **bpsh** to start your job, and use shell redirection on the master node to send the data as `stdin` for the job. Another is to use MPI and have the rank 0 job read the data and use MPI's message passing capabilities to send the data. If you have a job that is natively using Scyld BeoMaster BProc functions, you can also have your job read the data on the master node before it moves itself to the compute nodes.

Scyld Beowulf Software Components

The following sections highlight the various software packages in a Scyld Beowulf along with their individual components. For additional information, refer to the *Reference Guide* or the appropriate locations mentioned below.

BeoBoot Tools

The following are the tools associated with the **beoboot** package. For more details, refer to the *Reference Guide*.

beoboot

This utility is used to generate boot images for the compute nodes in the cluster. There are two sets of images: phase 1 and phase 2. Phase 1 images are placed on the hard disk or a floppy disk and are used to boot the nodes. The phase 2 image is downloaded from the master node by the phase 1 image. By default, the phase 2 image is stored on the master node in `/var/beowulf/boot.img` (this is where the **beoserv** *daemon* expects to find it.)

beoboot-install

This utility installs a phase 1 boot image created by **beoboot** onto the hard disk in a compute node, allowing the compute node to boot without using a floppy or CD-ROM drive.

beoserv

This is the BeoBoot *daemon*. It responds to RARP requests from the compute nodes in the cluster and serves them their phase 2 boot images over the private cluster network.

BProc Daemons

The following are the *daemons* associated with *BProc*. For more details, refer to the *Reference Guide*.

bpmaster

The BProc master daemon; it runs on the master node listening on a TCP port and accepting connections from **bpslave** daemons; configuration information comes from the Beowulf configuration file, `/etc/beowulf/config`.

bpslave

The BProc compute daemon; it runs on a compute node to accept jobs from the master; it connects to the master through a TCP port.

BProc Clients

The following is a list of the command line utilities closely related to BProc. For more details, refer to the *Reference Guide*.

bpsh

This is a **rsh** (remote shell) replacement; it runs a specified command on an individually referenced node; the *nodenum* parameter may be a single node number, a comma delimited list of nodes, -a for all nodes which are `up` or -A for all nodes which are not `down`.

bpsh will forward standard input, standard output and standard error for the remote processes it spawns; standard output and error are forwarded subject to specified options; standard input will be forwarded to the remote processes; if there is more than one remote process, standard input will be duplicated for every remote node; for a single remote process, the exit status of **bpsh** will be the exit status of the remote process.

bpctl

This is the *BProc control* utility; it can be used to apply various commands to individually referenced nodes; this utility can be used to change the `user` and `group` ownership settings for a node; it can also be used to set a node's *state*; finally, this utility can be used to query such information as the node's IP address.

bpcp

This utility can be used to copy files between machines in the cluster; each file `(f1...fn)` or directory argument `(dir)` is either a remote file name of the form *node:path*, or a local file name (containing no ':' characters.)

bpstat

This command displays various pieces of status information about the compute nodes; the display is formatted in columns specifying node number, node status, node permission, user access and group access; this program also includes a number of options intended to be useful for scripts.

Beowulf Utilities

The following is a list of various command line and graphical user interface (GUI) utilities that are part of a Scyld Beowulf. For additional information, refer to the *Reference Guide*.

beosetup (GUI)

This GUI tool is primarily used for configuring and administering the cluster but can also be used by general users in a read-only mode for monitoring cluster status; an entire chapter of this guide is devoted to a discussion of this tool.

beostatus (GUI)

This GUI tool is used for monitoring cluster status and performance; a discussion of this tool can be found in the *Monitoring the Status of Your Cluster* chapter in this guide.

beostat

This command line tool is a text-based utility for monitoring the status and performance of the cluster; the tool provides a text listing of the information from the `/proc` structure on each node; a discussion of this tool can be found in the *Monitoring the Status of your Cluster* chapter in this guide.

bbq

This command line tool provides a job batch spooler based on the **at** package and enhanced for use with a Scyld Beowulf; see the chapter on *Running Programs* in the *User's Guide* for more info.

beoqstat (GUI)

This GUI tool constantly monitors the contents and status of jobs in the **bbq** batch queue; see the chapter on *Running Programs* in the *User's Guide* for more info.

Chapter 2. Configuring the Cluster with the BeoSetup GUI

What's New in this Version?

The BeoSetup program has been rewritten and internally restructured, while retaining the appearance and most external details of the original release. The following is a list of the most apparent changes and new features.

1. The settings controlled by BeoSetup have been separated into **Preferences**, which control the appearance and actions of the GUI, and **Settings**, which affect the cluster configuration. Preferences are saved per-user (typically in `~/.gnome/beosetup`), while Settings are written to the cluster configuration file `/etc/beowulf/config`
2. Nodes indicate their state, e.g. Up or Down, by color highlights. The colors, along with other preferences such as which columns are displayed, may be set per-user.
3. Nodes in the Configured Nodes panel may be sorted, and reverse sorted, by the contents of any column by clicking on the column header.
4. Per-node menu items have been reordered and grouped by function.
5. A **Node CD** button has been added to simplify the creation of BeoBoot ISO-9660 "iso" images. This is similar to the node boot floppy creation, although an additional hardware- and media-specific step is needed to "burn" the resulting image to a CD-R or DVD-R.

BeoSetup Introduction

This section serves as a brief introduction to BeoSetup, the Scyld Beowulf cluster configuration program. It will give an overview and FAQ of BeoSetup, and it will explain how to run BeoSetup with different privileges.

BeoSetup Overview

The BeoSetup program is a graphical front-end for configuring, managing and monitoring a Scyld Beowulf cluster. It is intended to be used by the cluster system administrator, to whom all functions are available. It is also usable by non-privileged users which may use BeoSetup as a *read-only* window of the cluster's current status and use it to run commands.

Although BeoSetup makes many administrative operations easier, it *utilizes* the underlying functionality rather than *implementing* it. Using BeoSetup is optional: All operations performed by BeoSetup may also be done by editing the appropriate configuration files or running command line utilities.

The following is a list of the main features found in BeoSetup as well as of some common cluster administrative tasks. Each item contains a reference to the relevant sections of this document where more information can be found.

- To run BeoSetup with the expected privilege level: Full vs. Limited Privileges
- To check a node's state: The Configured Nodes List
- To add a new node to the cluster: Auto Activate New Nodes, The Unknown Addresses List and The Configured Nodes List
- To remove a node from the cluster: The Configured Nodes List
- To create a node floppy for booting a compute node: **Node Floppy**
- To change a node's state between on-line and off-line: The Configured Nodes List
- To change the kernel command line options used when booting a compute node: **BeoBoot File**

- To determine the node ordering of your cluster: Numbering Your Cluster's Nodes
- To ignore the boot request from a compute node: The Ignored Addresses List

Refer to the appropriate section(s) as noted above for further details.

BeoSetup's Frequently Asked Questions

The most common problems encountered when using BeoSetup are caused by configuration settings intended to allow explicit control over cluster operations.

- Nodes appear on the Unknown address list, and are not activated into the cluster.
- After reordering nodes, or moving them between Unknown, Ignored and Configured tables, the changes do not take effect.
- When manual configuration is selected, node activation and movement must be explicitly indicated and confirmed. Confirmation is done by clicking on the **Apply** button. Automatic activation and Auto-apply are set in the Preferences entries.

Running BeoSetup

You can start BeoSetup using any one of these two methods: 1) click on the BeoSetup icon on your Gnome Panel; or, 2) type **beosetup** on the command line in any terminal window. For a complete list of BeoSetup's options, including all the GNOME/GTK options, use the **-?/--help** command line option. The following command line options are specific to BeoSetup:

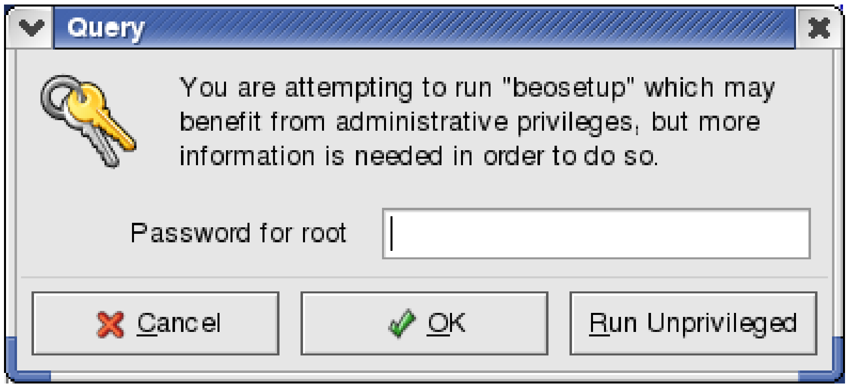
beosetup [-c, --conf=CONFIG_FILE] [-m, --cm=CLUSTER_MANAGEMENT_TYPE] [-n, --new=NEW_NODE_FILE]

The **-c, --conf** option lets you specify an alternate cluster configuration file to read and modify. The default value is `/etc/beowulf/config`, which is the default configuration file used by all Scyld Beowulf tools.

The **-n, --new** option lets you specify an alternate file for the "new node" file. The default value is `/var/beowulf/unknown_addresses`, which is typically written by the **beoserv** or **beopxeserv** daemon when new machines attempt to join the cluster.

Full vs. Limited Privileges

When a single BeoSetup is started by 'root', it automatically starts with full privileges. If you start BeoSetup using a non-root account, you will be presented with the following dialog box:


Query

You are attempting to run "beosetup" which may benefit from administrative privileges, but more information is needed in order to do so.

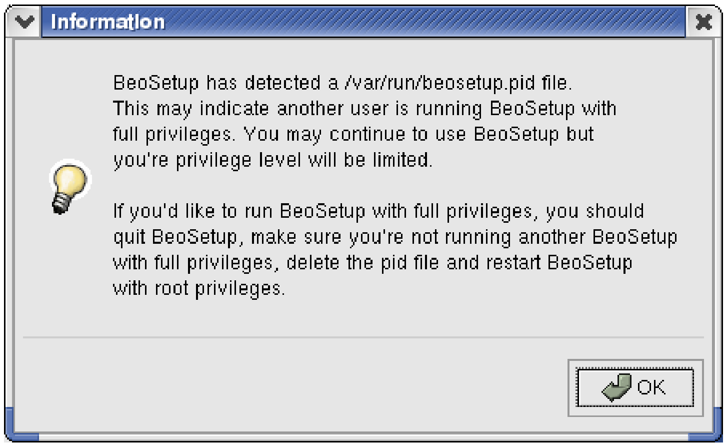
Password for root

If you'd like to run BeoSetup with full privileges and you know the root password, enter the root password in the text entry field labeled *Password for root* and click the **OK** button. You can click the **Run Unprivileged** button to simply run BeoSetup with limited privileges. Click the **Cancel** button if you decide you no longer want to start BeoSetup.

Once BeoSetup has been started, you can look at the title bar to determine whether you're running with full or limited privileges. When running with full privileges, you have complete access to all of BeoSetup's features and functionalities. With limited privileges BeoSetup allows monitoring and using the cluster, but not changing its configuration.

Limiting Full Privileges

To prevent two or more administrators from making changes with BeoSetup simultaneously, the BeoSetup program only allows a single instance of itself to be run with full privileges at any given time. If you attempt to start BeoSetup with full privileges when another fully privileged instance is already running, you will be presented with the following dialog box:



BeoSetup has detected a /var/run/beosetup.pid file.
This may indicate another user is running BeoSetup with
full privileges. You may continue to use BeoSetup but
you're privilege level will be limited.

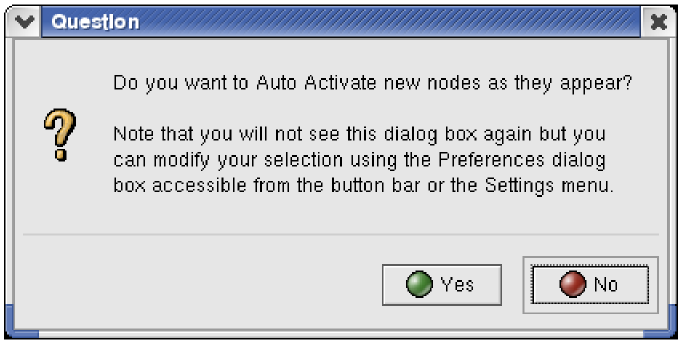
If you'd like to run BeoSetup with full privileges, you should
quit BeoSetup, make sure you're not running another BeoSetup
with full privileges, delete the pid file and restart BeoSetup
with root privileges.

After clicking the **OK** button, BeoSetup will be started with limited privileges. You must shutdown the instance of BeoSetup currently running with full privileges before you can run another instance with full privileges. If you definitely know there's not another fully privileged instance of BeoSetup running yet you keep getting the above dialog box when attempting to start BeoSetup with full privileges, the "pid file" used to provide this protection probably needs to be deleted manually. To delete this file, enter the following command in a terminal window with root privileges:

```
# rm /var/run/beosetup.pid
```

Auto Activate New Nodes

When running BeoSetup with full privileges for the first time after installation, you will be presented with the following dialog box asking if you'd like to enable the *Auto Activate New Nodes* feature:


Question

Do you want to Auto Activate new nodes as they appear?

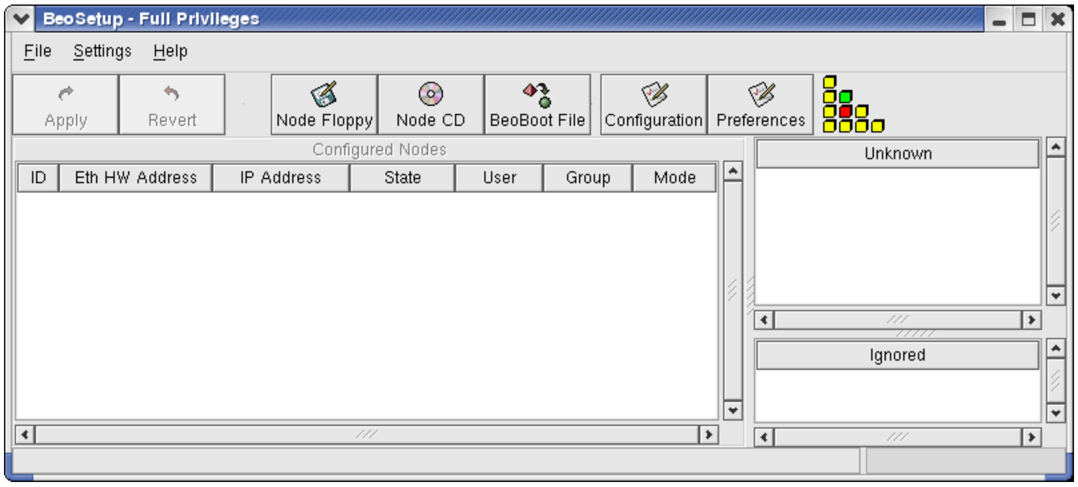
Note that you will not see this dialog box again but you can modify your selection using the Preferences dialog box accessible from the button bar or the Settings menu.

Answering **Yes** or **No** to this dialog box configures the feature appropriately and closes the dialog box. (The Auto New Node Assignment feature is discussed in the section on the **Preferences...** dialog box.) After a button has been selected and the dialog box has been closed, it will not be shown again during subsequent restarts of BeoSetup. Note that if you close the dialog box using the window manager, the auto-activate feature retains its default value of "off" and you will be shown the dialog box each time you start BeoSetup with full privileges until you select one of the buttons.

If you ever want to force the reappearance of this dialog box at startup, delete the line `askautoact=false` from the `[options]` section of the file `/root/.gnome/beosetup`. The next time BeoSetup is started, you'll again be presented with the above dialog box.

BeoSetup Screen Elements

The following is a screenshot of BeoSetup's main window:


BeoSetup - Full Privileges

Working our way from the top of the window down to the bottom, we have the following screen elements: 1) the title bar; 2) the main menu; 3) the toolbar; 4) the node list windows; and, 5) the status bar. The title bar contains the name of the program along with its current privilege level. The **File**, **Settings** and **Help** pull-down menus are all available from the main menu. You can use the keyboard or mouse to open the menus and make your selection. The toolbar button under the main menu provides some additional functionality, as well as shortcuts to some of the items available in the main menu. The largest portion of the main window is occupied with the three "node list" windows: the Unknown Address List, the Configured

Nodes List, and the Ignored Addresses List. These lists contain relevant information about your cluster's compute nodes. Generally, unless you're performing cluster maintenance activities, only the Configured Nodes List contains node data. Finally, the lowest portion of the window contains the status bar. From time to time, when performing certain activities, various informational text messages from BeoSetup will appear in this area of the screen.

The File Menu

The **File Menu** contains the following selections:

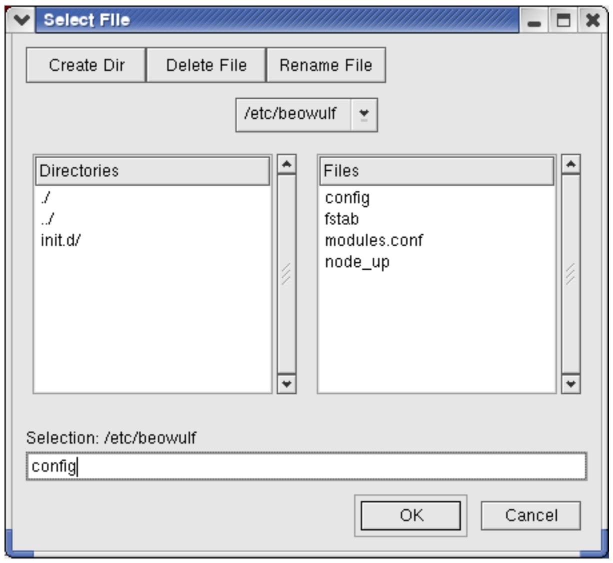
- **Configuration File...**
- **Start Cluster**
- **Service Reconfigure**
- **Shutdown Cluster**
- **Exit**

Refer to the appropriate section as noted above for further details.

Configuration File...

The **Configuration File...** item on the **File Menu** allows you to specify a different filename for the configuration file being used by BeoSetup. The configuration file specified here, or with the **-c, --config** command line option, must be the same one that the **beoserv** daemon is currently using or your cluster will not react properly to the actions performed with BeoSetup. The default configuration filename is `/etc/beowulf/config`.

When choosing this menu item, you are presented with the dialog box shown below. Note that if you're running BeoSetup with limited privileges and you select this menu item, you will be presented with a message box identifying the name of the current configuration file and explaining that you must be running with full privileges to change it.



Selection: /etc/beowulf

config

This is a standard file selection dialog box. Use the dialog's widgets to locate the desired file. Once the appropriate file has been selected, click the **OK** button to finalize your decision and close the dialog box. If you decide you no longer want to change the boot configuration filename, click the **Cancel** button.

Start Cluster

Start Cluster starts or restarts cluster services. If the cluster was previously running, all nodes will reboot and rejoin the cluster.

Service Reconfigure

Service Reconfigure triggers the Beowulf daemons to immediately re-read the configuration file, taking action on any changes from the previous version. Most changes do not require the compute nodes to reboot. Changing the assigned IP addresses or the node count will cause all nodes to restart, while changing the node ordering will only cause the affected nodes to reboot.

Shutdown Cluster

Selecting **Shutdown Cluster** from the **File Menu** stops the cluster daemons, which immediately disconnect from the compute nodes, eventually causing them to reboot.

Exit

Selecting **Exit** from the **File Menu** closes BeoSetup.

The Settings Menu

The **Settings Menu** contains the following selections:

- **Configuration**
- **Preferences...**
- **Reread Cfg Files**

Refer to the appropriate section as noted above for further details.

Configuration: Cluster

The **Configuration** item on the **Settings Menu** displays a multi-paneled dialog box allowing you to modify many of the options found in the cluster configuration file, `/etc/beowulf/config`. Any options that can be changed from this dialog box but which are not stored in this configuration file, are options specific to BeoSetup and are stored in the file `/root/.gnome/beosetup`. The dialog box contains the following four panels: *Network properties*, *Nodes*, *Node File Systems* and *PXE Network Boot*, which are discussed below.

Running along the bottom of the dialog box are four buttons. The **OK** and **Apply** buttons perform somewhat similar operations. When selected, both buttons validate the changes made to the dialog and then act on those changes. The difference between the two buttons is that the dialog box is closed when using the **OK** button, but remains open when you select **Apply** allowing you to continue making additional changes.

If any of your changes are invalid (see particulars below where the widgets are discussed) when you click **OK** or **Apply**, you'll be presented with a message box describing the error. Upon closing the error message box, one of two things will happen depending on which button you first selected. If you selected the **OK** button, after closing the error message box the **Configuration** dialog box closes immediately. If you selected the **Apply** button, after closing the error message box the **Configuration** dialog box remains open giving you the opportunity to correct the errors and try again.

Selecting the **Close** button results in the loss of any changes you've made since you last selected the **Apply** button and closes the dialog box.

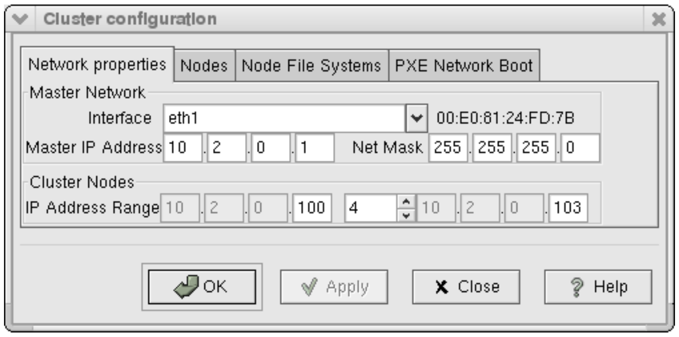
Selecting the **Help** button displays this page in the help browser.

If you're running BeoSetup with limited privileges and you select this menu item, the **Configuration** dialog box will still be displayed but all the widgets on each panel will be grayed out and disabled. With limited privileges, you can view the current settings but you won't be able to modify any settings. To make any changes, you must be running with full privileges.

A toolbar button is available as a shortcut for this menu item. When selecting the toolbar button or choosing the menu item, you are presented with the dialog box shown below, initially displaying the *Network properties* panel. To switch among the panels on the dialog box, simply select the tab for the desired panel.

Configuration: Network properties

The *Network properties* panel of the **Cluster configuration** dialog box is shown below followed by descriptions of the individual widgets. Each of the options discussed below is stored in the configuration file, `/etc/beowulf/config`. In the Network properties panel screenshot, eth1 is the internal network interface.



Master Network
Interface eth1 00:E0:81:24:FD:7B
Master IP Address 10 . 2 . 0 . 1 Net Mask 255 . 255 . 255 . 0
Cluster Nodes
IP Address Range 10 . 2 . 0 . 100 4 10 . 2 . 0 . 103

Interface

This widget is where you select the internal cluster network interface used by the Beowulf server daemon. It defines the communication path between the master node and all the compute nodes in the cluster. This option is stored in the config file as `interface`.

Master IP Address

This widget displays the IP address of the master node on the internal cluster network interface. This is the address used by the compute nodes when communicating back to the master node. This address must be in the same subnet, but may not fall in the IP address range used for the compute nodes (see below). This setting, along with the Net Mask, is typically configured by the underlying Linux system based on the contents of `/etc/sysconfig/network-scripts/ifcfg-eth0`.

Net Mask

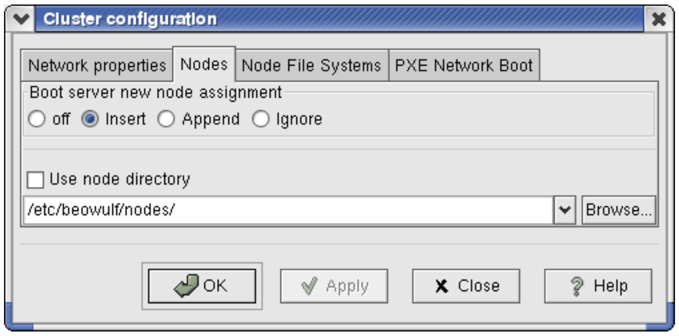
This widget displays the network address mask configuration of the internal cluster network interface. This setting is read from the current interface state.

IP Address Range

This widget is where you specify the range of IP addresses assigned to the compute nodes on the internal cluster network interface. The maximum number of compute nodes in the cluster is defined by this *inclusive* range (the range includes the upper address). This option is stored in the config file as `iprange` and the computed value `nodes`. BeoSetup assists settings a valid range by using the Net Mask to limit which elements of the addresses may be modified.

Configuration: Nodes

The *New Nodes* panel of the **Cluster configuration** dialog box is shown below followed by descriptions of the individual widgets. New nodes may be assigned by either the always-running service daemons, or the GUI ("manually" from the point of view of the daemons, even if it's the GUI automatically moving the addresses between columns). Each of the options discussed below is stored in the configuration file, `/etc/beowulf/config`.



Boot server new node assignment: off [✓ Insert] Append Ignore

Use node directory: ☐

/etc/beowulf/nodes/

Boot server new node assignment

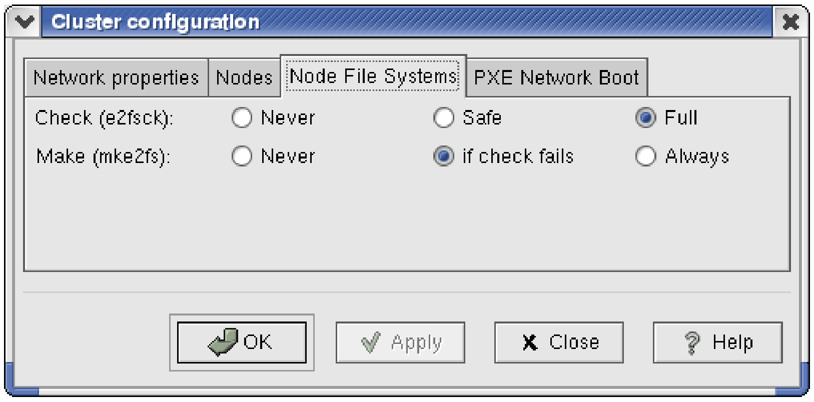
This widget displays the boot server new node assignment used to specify cluster machines by their MAC (Media Access Control) addresses. When set to *off*, the operator can manually configure all new addresses. When set to *insert*, the default, new node addresses are assigned to the first available node number. Nodes are numbered sequentially starting from 0. When set to *Append*, new node addresses are assigned to the end of the configured node list (effectively Insert if the final node number has been assigned). Finally, when set to *Ignore*, newly seen MAC addresses are recorded as machines which may not join the cluster.

Use node directory

When this widget is *checked*, node information is also stored in the specified directory, with one file written per node. The default directory is `/etc/beowulf/nodes/`.

Configuration: Node File Systems

The *Node File Systems* panel of the **Cluster configuration** dialog box is shown below followed by descriptions of the individual widgets. The options on this panel all deal with the later stages of booting. When a compute node is booting, the server will attempt to configure the node's filesystems by running some combination of a filesystem check and a filesystem create. This panel lets you configure how the filesystems are treated during boot. Each of the options discussed below is stored in the configuration file, `/etc/beowulf/config`.



Check (e2fsck): Never Safe [✓ Full]

Make (mke2fs): Never [✓ if check fails] Always

Check (e2fsck)

These mutually exclusive radio buttons specify the type of filesystem check performed during boot. When set to *Never*, no filesystem check is performed. When set to *Safe*, the filesystem check gives up if any "bad" errors are encountered. When set to *Full*, the system will try to answer "y" to all the "should I fix?" questions that may occur during the filesystem check. This option is stored in the config file as `fsck`. The possible values are `never`, `safe` or `full`.

Make (mke2fs)

These mutually exclusive radio buttons specify the type of filesystem create performed during boot. When set to *Never*, no filesystem create is performed. When set to *if check fails*, a blank filesystem is recreated if the filesystem check fails. When set to *Always*, the filesystem is recreated during every boot. With this option selected, the filesystem check will be skipped and thus selection is disabled. This option is stored in the config file as `mkfs`. The possible values are `never`, `if_needed` or `always`.

Configuration: PXE Network Boot

The *PXE Network Boot* panel of the **Cluster configuration** dialog box is shown below followed by descriptions of the individual widgets. *PXE* stands for Preboot eXecution Environment, which is a set of protocols and specifications for standardized network boot. Most machines with built-in Ethernet interfaces now include a PXE boot client in firmware, which substitutes for stage one of Scyld BeoBoot.

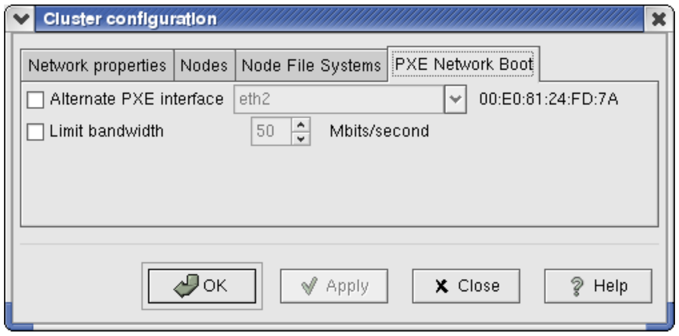
Scyld Beowulf includes our own BeoPXE Server, which makes compute node booting substantially more reliable than other implementations and is directly configured from the standard cluster configuration.

The checkbox for **Alternate PXE Interface** should be selected when the compute nodes must boot from a different network than they use for run-time operation.

Three examples of configurations that require this setting are

- When compute nodes have a Fast Ethernet network interface that supports PXE booting, and a Gigabit Ethernet interface without PXE support. The Gigabit Ethernet is the desired operational network.
- When compute nodes have an Ethernet network interface that supports PXE booting, and a specialized cluster NIC (Myrinet, Infiniband, etc.)
- *RLX compute blades* where the only network interface that supports PXE booting is connected to an internal repeater. Using an Ethernet repeater is very slow during normal operation, so the cluster interfaces should be connected to an Ethernet switch, while the master should use a separate network interface connected to the internal RLX repeater. The second interface, and the internal repeater, is used only for booting.

Each of the options discussed below is stored in the configuration file, `/etc/beowulf/config`.



☐ Alternate PXE interface eth2 00:E0:81:24:FD:7A
☐ Limit bandwidth 50 Mbits/second

Alternate PXE interface

When this widget is *checked*, the BeoPXE Server uses the specified network interface for PXE booting rather than cluster network interface. This alternate network interface should be connected to the compute nodes booting interface. Input the network interface name or choose the down arrow for other network selections.

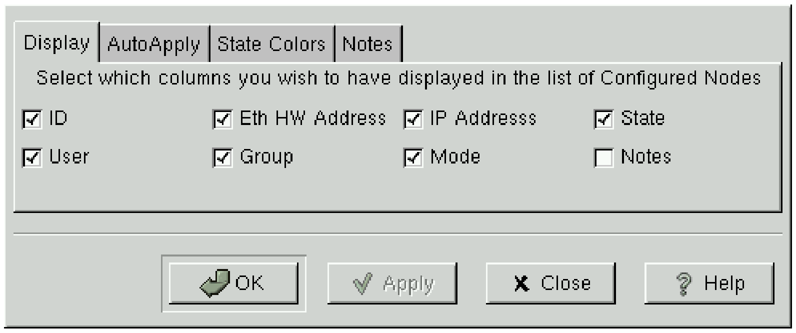
Limit bandwidth

When this widget is *checked*, BeoSetup limits network use of booting machines to avoid timeouts. Input or use the increment/decrement arrows to select the desired PXE service bandwidth limit in Mbps.

Preferences...: BeoSetup Preferences

Preferences: Display

The *Display* panel of the **Preferences...** dialog box is shown below followed by descriptions of the individual widgets. Each of the options discussed below is stored in the file `/root/.gnome/beosetup`.



Select which columns you wish to have displayed in the list of Configured Nodes

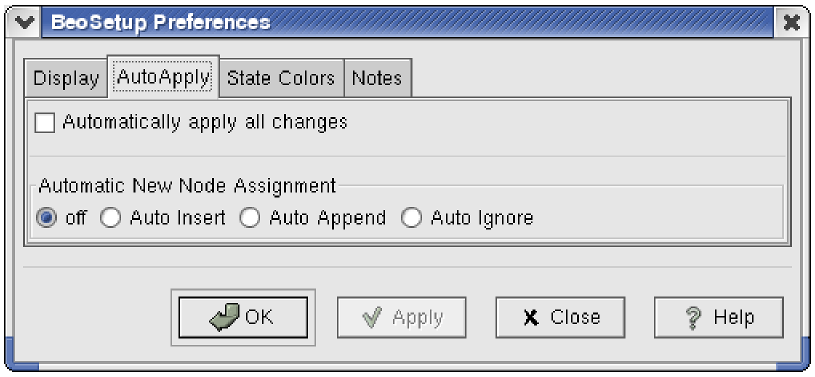
☑ ID ☑ Eth HW Address ☑ IP Address ☑ State
☑ User ☑ Group ☑ Mode ☐ Notes

Columns displayed in the list of Configured Nodes.

This series of buttons specify which columns are displayed in the list of Configured Nodes. When the *ID* box is selected, the number Node ID *(required)* is displayed. When the *Eth HW Address* box is checked, the node's unique hardware address is displayed. When the *IP Address* box is checked, the assigned IP address is displayed. When the *State* box is checked, the current node status is displayed(i.e. Down, Up, Rebooting, etc.). When the *User* box is checked, the user that controls the node is displayed. When the *Group* box is checked, the group that controls the node is displayed. When the *Mode* box is checked, the execute permissions are displayed, i.e. user, group or all. When the *Notes* box is checked, the administrative notes are displayed.

Preferences: AutoApply

The *AutoApply options* panel of the **Preferences...** dialog box is shown below followed by descriptions of the individual widgets. Each of the options discussed below is stored in the file `/root/.gnome/beosetup`.



☐ Automatically apply all changes

Automatic New Node Assignment

[✓ off] Auto Insert Auto Append Auto Ignore

Automatically apply all changes

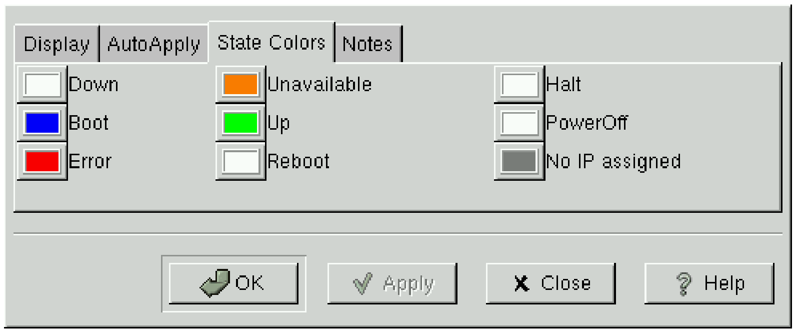
When this widget is *checked*, BeoSetup will automatically apply all changes made in all three node list windows: Unknown Addresses, Ignored Addresses, or Configured Nodes. This option is stored in `/root/.gnome/beosetup` as `autoapply`. It's a boolean data type and its value can be either `true` or `false`.

Automatic New Node Assignment

This series of mutually exclusive radio buttons specifies how BeoSetup treats new compute nodes that normally appear as unknown addresses. When set to *off*, the default, new nodes are left in the Unknown Addresses List. It is up to the operator to make any changes when this option is selected. When set to *Auto Insert*, BeoSetup automatically assigns new nodes to the first available node number displayed in the Configured Nodes List. When set to *Auto Append*, BeoSetup assigns new node addresses to the end of the Configured Nodes List. When set to *Auto Ignore*, BeoSetup automatically moves new nodes to the Ignored Addresses List.

Preferences: State Colors

The *State Colors options* panel of the **Preferences...** dialog box is shown below followed by descriptions of the individual widgets.

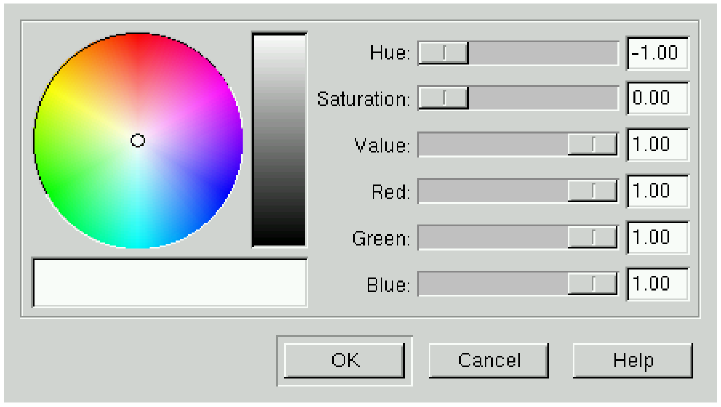


State Colors options

This set of color selection boxes specifies the highlight color of nodes in the Configured Nodes panel, based on the node state. *Down* - Display color for the Node Down state, *Unavailable* - Display color for nodes in the Administratively Unavailable state, *Halt* - Display color for nodes that have been commanded to halt, *Boot* - Display color for the nodes that are booting, *Up* - Display color for fully operational nodes, *PowerOff* - Display color for nodes commanded to power off, *Error* - Display color for the nodes that encountered an error while booting, *Reboot* - Display color for nodes that are rebooting, *No IP assigned* - Foreground color for nodes without an IP address

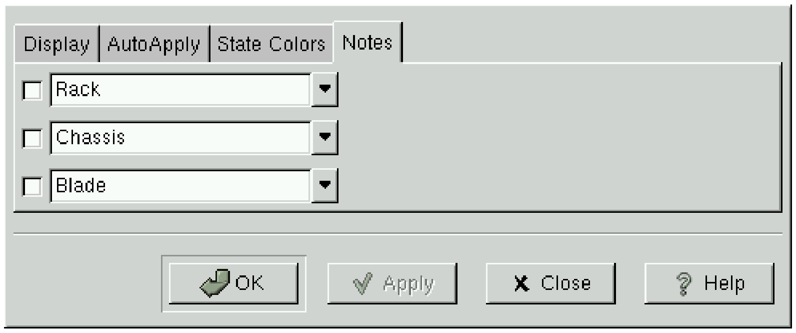
State Colors option custom color wheel panel

The *State Colors option custom color wheel* panel of the **Preferences: State Colors** dialog box is shown below. Select a desired custom color by moving the circle within the color wheel.


Hue: -1.00
Saturation: 0.00
Value: 1.00
Red: 1.00
Green: 1.00
Blue: 1.00

Preferences: Notes

The *Notes options* panel of the **Preferences...** dialog box is shown below followed by descriptions of the individual widgets.



Notes options

When *checked*, either column one, two, or three are visible to a user. Select the corresponding user note column title, default are *Rack, Chassis, or Blade*, with the pulldown arrow or enter a desired user note title.

Reread Cfg Files

The **Reread Cfg Files** item on the **Settings Menu** performs the following commands:

```
# killall -HUP bpmaster
# killall -HUP beoserv
```

The net effect of these two commands is that each of the above mentioned daemons is restarted and rereads the Beowulf configuration file `/etc/beowulf/config`. Strictly speaking, you never really need to perform this operation in response to any of the changes made with BeoSetup. Whenever you apply any of your changes, the appropriate files are automatically rewritten and the appropriate signals are sent to the appropriate daemons. One example of when you might use this function is after you've made edits to any of the configuration files using something other than BeoSetup. For your edits to have an effect, more than likely the daemons will need to reread the configuration files.

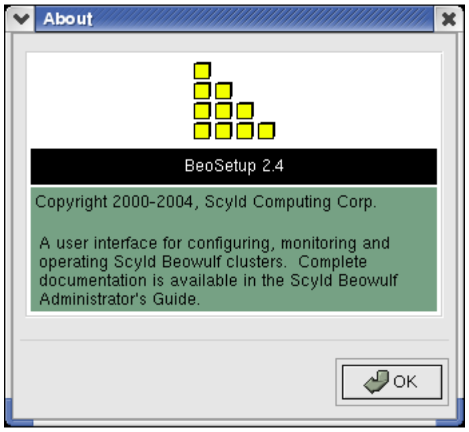
If you select this menu item when running BeoSetup with limited privileges, you will be presented with a message box informing you that you should be running BeoSetup with full privileges to perform this operation.

The Help Menu

The **Help Menu** contains only the **About...** selection item.

About...

Selecting **About...** from the **Help Menu** results in the display of this informational dialog box containing a brief description of BeoSetup and the copyright notice:



Copyright 2000-2004, Scyld Computing Corp.

A user interface for configuring, monitoring and operating Scyld Beowulf clusters. Complete documentation is available in the Scyld Beowulf Administrator's Guide.

Simply click the **OK** button and the dialog box will close.

The Toolbar

BeoSetup's toolbar is the collection of push buttons directly underneath the main menu and directly above the node list windows. Toolbars typically contain shortcuts for performing operations available from the main menu, such as the **Node Floppy** and **Preferences...** buttons. Additionally, they may provide access to other features and / or be used to provide visual feedback to the user, such as the **Apply** and **Revert** buttons. All of the toolbar buttons are discussed below.

Apply

The **Apply** toolbar button is used to activate any changes made in any of the three node list windows: Unknown Addresses, Configured Nodes and / or Ignored Addresses. When selected the *apply* operation writes the latest node information from all three node list windows to the appropriate configuration files and then commands the appropriate daemons to reread these configuration files. For a description on what information from each node list window gets written to which configuration file, see the appropriate sections on the node list windows.

When running BeoSetup with limited privileges, this toolbar button will always be disabled as users with this privilege level are unable to make any changes anyway. When running with full privileges, the button is initially disabled and only becomes enabled after you make a change in one of the node list windows. For example, when you initially drag a node from the Unknown Address List to the Configured Node List, the **Apply** button gets enabled. This *new* node isn't recognized as part of the cluster until you select the **Apply** button. Whenever this toolbar button is enabled, it indicates the information displayed in your node list windows is not synchronized with the data in the configuration files.

The *Automatically apply all changes* feature on the *AutoApply options* panel of the **BeoSetup Preferences** dialog box provides a method for bypassing this extra step when making any changes in the node list windows.

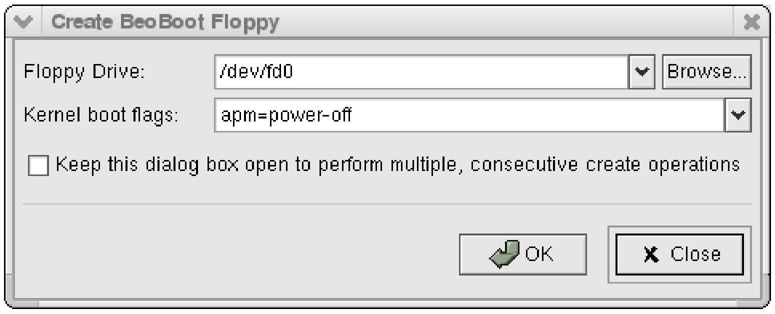
Revert

The **Revert** toolbar button is used to throw away any changes made in any of the three node list windows: Unknown Addresses, Configured Nodes and / or Ignored Addresses. When selected, the *revert* operation deletes the contents of all three node list windows thus undoing all the changes you've made since you last selected the **Apply** button. The three lists are then repopulated by rereading the current configuration files. For a description on what information is read from the configuration files when initializing the contents of the node list windows, see the section on the appropriate node list window. The idea is that this button lets you *revert* back to the original contents of the configuration files before you started making any changes with BeoSetup.

The **Revert** button is initially disabled. Whenever a fully privileged user makes a change in one of the node list windows when BeoSetup's *Auto Apply* feature is disabled, the **Revert** button becomes enabled. Selecting the button performs the operations described above. The **Revert** button also becomes enabled whenever BeoSetup notes a change in the modification time of the configuration file, `/etc/beowulf/config`. The enabling of the **Revert** under this circumstance occurs regardless of the current privilege level. This feature is meant to alert all users to changes to the configuration file occurring *behind their back*. Whenever this button is enabled, it usually indicates the information displayed in your node list windows is not synchronized with the data in the configuration files.

Node Floppy

The **Node Floppy** item on the toolbar button allows you to create a **beoboot** floppy disk image for booting a compute node in your cluster. This phase one, or initial, image is composed of a minimal kernel image and an initial ram disk image. You should not have to regenerate this image unless you make some kind of hardware change to the cluster. When selecting the toolbar button you are presented with the dialog box shown below.


Create BeoBoot Floppy

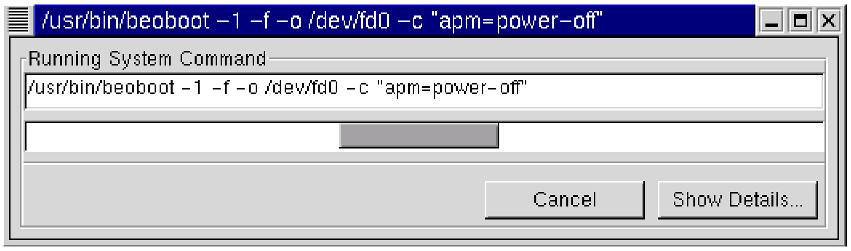
Floppy Drive: /dev/fd0

Kernel boot flags: apm=power-off

Keep this dialog box open to perform multiple, consecutive create operations

The *Kernel boot flags* are read-only regardless of whether you're running with full or limited privileges. These widgets are strictly for informational purposes. (The kernel boot flags are stored in the config file, `/etc/beowulf/config`, as `kernelcommandline` respectively.) The *Floppy Drive* widget is where you specify the device name for the floppy drive you'll be using. The default is `/dev/fd0`, which specifies the first floppy drive in a standard desktop computer. The *Keep this dialog box open...* check box widget provides a means for you to easily create multiple boot floppies without having to reopen the dialog box multiple times.

After the floppy media has been inserted into the drive, click the **OK** button to begin creating your node boot floppy. If you decide you no longer want to create a node boot floppy, click the **Cancel** button to close the dialog box. Once creation of the boot floppy has started, you'll be presented with the following dialog box providing you with visual feedback as the operation progresses.


/usr/bin/beoboot -1 -f -o /dev/fd0 -c "apm=power-off"

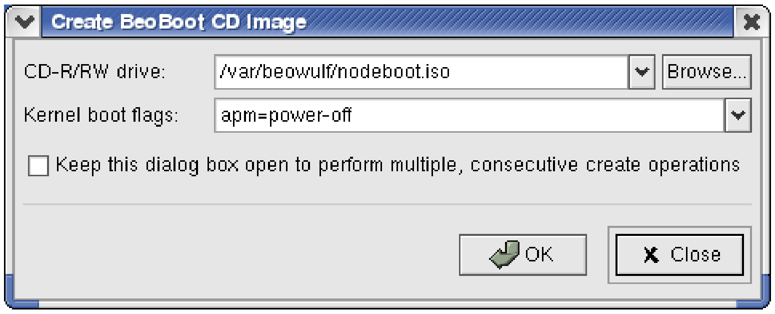
Running System Command

```
/usr/bin/beoboot -1 -f -o /dev/fd0 -c "apm=power-off"
```

If at any time you want to stop the creation of the node boot floppy, simply click the **Cancel** button. When the operation completes, the **Cancel** button is renamed to **Close**, which can then be clicked to close the dialog box and end the operation. At any time, you can click the **Show Details...** button to enlarge the dialog and view the output from the execution of the **beoboot** command shown in the *Running System Command* widget.

Node CD

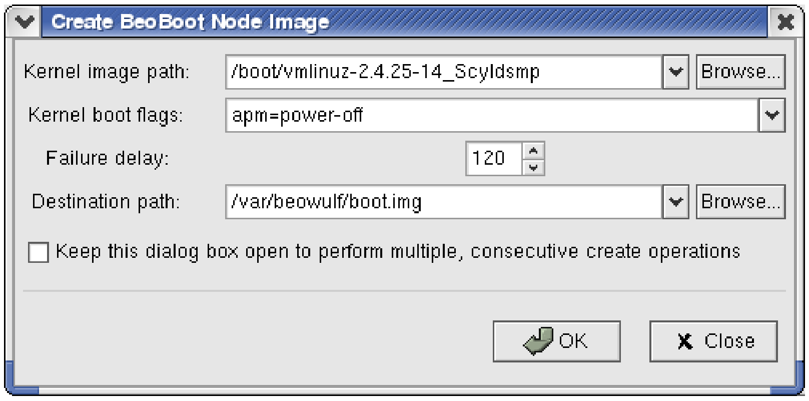
The **Node CD** button on the toolbar creates a beoboot slave node boot image that can be written to a CD-R/RW (ISO-9660 image.)



CD-R/RW drive: /var/beowulf/nodeboot.iso Browse...
Kernel boot flags: apm=power-off
Keep this dialog box open to perform multiple, consecutive create operations

BeoBoot File

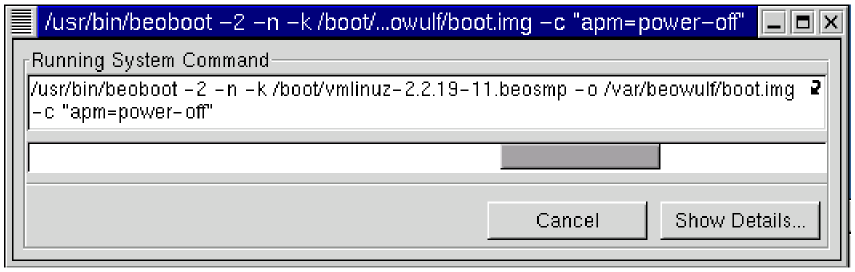
The **BeoBoot File** item on the toolbar button allows you to create a network boot image, which is downloaded from the master node to each compute node during phase two of the boot process. This **beoboot** file contains the final kernel image and modules that the compute node will use. The second phase boot image should be updated whenever you upgrade the kernel or any modules on the front end. Running the same kernel on the master node and the compute nodes is highly recommended. Selecting the **BeoBoot File** item on the toolbar button, you are presented with the dialog box shown below.



Kernel image path: /boot/vmlinuz-2.4.25-14_Scyldsmp Browse...
Kernel boot flags: apm=power-off
Failure delay: 120
Destination path: /var/beowulf/boot.img Browse...
Keep this dialog box open to perform multiple, consecutive create operations

The *Kernel image path* and *Kernel boot flags* widgets are pretty self-explanatory. The first is the kernel image file that will be sent to the compute node by the master during phase two of the boot process. The second is for specifying any boot flags you'd like passed to this phase two kernel when it's booted. (The history of items entered in the *Kernel boot flags* widget is stored in the `[History: Kernel boot flags]` section of the file `/root/.gnome/beosetup`.) The *Destination path* widget is where you specify the location and name for the BeoBoot image file you'll be creating. The default is `/var/beowulf/boot.img`. The *Keep this dialog box open...* check box widget provides a means for you to easily create multiple image files without having to reopen the dialog multiple times.

When you’re ready, click the **OK** button to begin creating your image file. If you decide you no longer want to create an image file, click the **Cancel** button to close the dialog box. Once creation of the image file has started, you’ll be presented with the following dialog box providing you with visual feedback as the operation progresses.



```
/usr/bin/beoboot -2 -n -k /boot/...owulf/boot.img -c "apm=power-off"
Running System Command
/usr/bin/beoboot -2 -n -k /boot/vmlinuz-2.2.19-11.beosmp -o /var/beowulf/boot.img
-c "apm=power-off"
```

If at any time you want to stop the creation of the beoboot file, simply click the **Cancel** button. When the operation completes, the **Cancel** button is renamed to **Close**, which can then be clicked to close the dialog box and end the operation. At any time, you can click the **Show Details...** button to enlarge the dialog and view the output from the execution of the **beoboot** command shown in the *Running System Command* widget.

Configuration

The **Configuration** button on the toolbar is a shortcut to the **Configuration** item on the **Settings Menu**. Clicking this toolbar button using the mouse is equivalent to selecting the aforementioned item on the Settings Menu.

Preferences

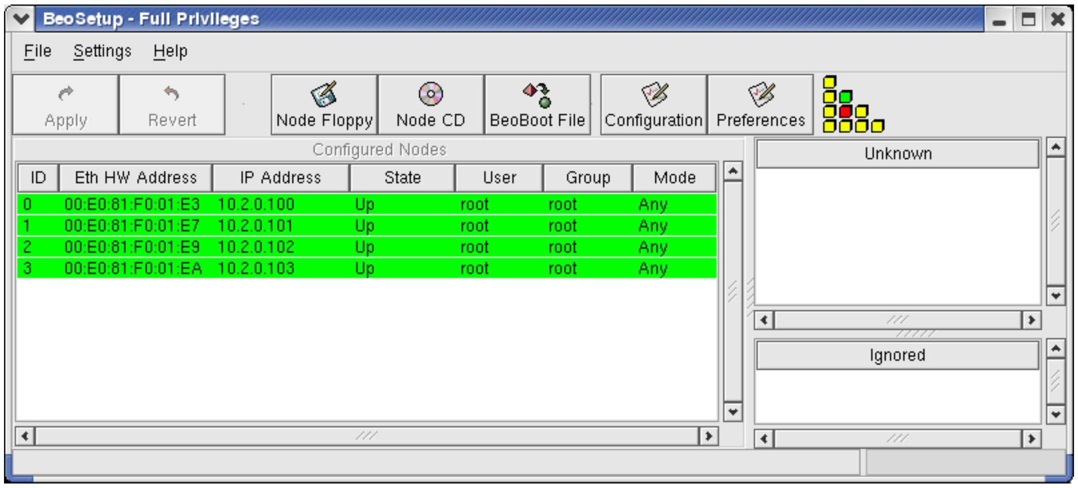
The **Preferences...** button on the toolbar is a shortcut to the **Preferences...** item on the **Settings Menu**. Clicking this toolbar button using the mouse is equivalent to selecting the aforementioned item on the Settings Menu.

The Node List Windows

The central area of BeoSetup’s main window is occupied by the following *node list windows*:

- The Configured Nodes List
- The Unknown Addresses List
- The Ignored Addresses List

These windows are located directly beneath the toolbar buttons and directly above the status bar running along the bottom of the main window. Each list window corresponds to one of the three ways cluster nodes can be classified: *unknown*, *configured* or *ignored*.



ID	Eth HW Address	IP Address	State	User	Group	Mode
0	00:E0:81:F0:01:E3	10.2.0.100	Up	root	root	Any
1	00:E0:81:F0:01:E7	10.2.0.101	Up	root	root	Any
2	00:E0:81:F0:01:E9	10.2.0.102	Up	root	root	Any
3	00:E0:81:F0:01:EA	10.2.0.103	Up	root	root	Any

An *unknown* node is one not formally recognized by the cluster as being either *configured* or *ignored*. When a compute node completes its phase 1 boot process or PXE boot, it begins to send out RARP requests on the cluster's internal network interface. When the master node receives a RARP request from a node not already listed as *configured* or *ignored* in the cluster configuration file, it considers the request to be from an *unknown* node and adds it to the unknown addresses file. When bringing a new compute node online or after replacing an existing node's network interface card, the node will be classified as *unknown* and will subsequently be listed in the Unknown Address List Window. Refer to the section on the *Automatic New Node Assignment* option to see how *unknown* nodes may be automatically classified as either *configured* or *ignored* by BeoSetup.

A *configured* node is one that is listed in the cluster configuration file using the `node` tag. These are nodes that are formally part of the cluster and recognized as such by the master node. Unless you are running BeoSetup with the *Automatic New Node Assignment* feature set to *Auto Activate*, new nodes being brought online are not automatically added to the cluster and classified as *configured*. (These new nodes are classified as *unknown* as described above.) With the *Auto Activate* feature disabled, operator interaction, as described below, is required to officially accept new nodes into the cluster. All nodes classified as *configured* are listed in the Configured Nodes List Window.

An *ignored* node is one that is listed in the cluster configuration file using the `ignore` tag. These are typically nodes that for one reason or another you'd like the master node to simply ignore. They are not considered part of the cluster and will not receive the appropriate responses from the master during their boot process. As an example, you may choose to temporarily classify a node as *ignored* while performing hardware maintenance activities when the node may be frequently rebooted. While BeoSetup's *Automatic New Node Assignment* feature supports an *Auto Ignore* option, you'll most likely end up using the methods described below to identify nodes as *ignored*. All nodes classified as *ignored* are listed in the Ignored Addresses List.

As discussed above, the classification of a node determines how and where the node's data is stored in the cluster configuration files. Unknown nodes are stored in the file `/var/beowulf/unknown_addresses` with the tag `unknown` followed by the node's Ethernet hardware (MAC) address. Configured nodes are stored in the file `/etc/beowulf/config`. They use the tag `node` following by the node's Ethernet hardware (MAC) address and the node's user and group permission data. Ignored nodes are stored in the file `/etc/beowulf/config` with the tag `ignore` followed by the node's Ethernet hardware (MAC) address.

Each node list window provides a pop-up menu, which may be opened by clicking the right mouse button (in some cases you must right click over an item in the list to get the menu to appear). The items presented in each menu are context sensitive, but at a minimum each menu provides items to reclassify a node by moving it from its existing node list window to any of the other two. In addition, nodes may be moved between windows using drag 'n drop mouse operations when running with full privileges. As noted below, after performing any of these operations you must apply your changes for them to take effect.

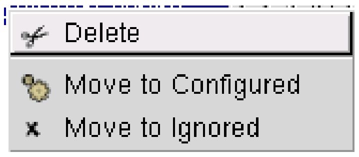
The Unknown Addresses List

The *Unknown Addresses* list window displays the *unknown* nodes currently listed in the unknown addresses file. Each item in this node list window shows the Ethernet hardware (MAC) address of a node listed in this file. A general discussion of how nodes are classified can be found in the section on Node List Windows.

Unknown Addresses Pop-up Menu

To reclassify an *unknown* node, you may drag the node from the *Unknown Addresses List* and drop it over one of the other two node list windows: Configured Nodes or Ignored Addresses. You may perform the same operation using a pop-up menu. If you right-click over any of the nodes listed in this window, you are presented with the pop-up menu shown below. If you are running BeoSetup with limited privileges, all the items in the pop-up menu will be disabled. To use the pop-up menu or the drag 'n drop features, you must be running with full privileges.

For any changes you make in this window to take effect, you must be sure to apply them.



Delete

This menu item deletes the selected node from the *Unknown Addresses* list. Once this change has been applied, the selected node is removed from `/var/beowulf/unknown_addresses`.

Move to Configured

This menu item moves the selected node from the *Unknown Addresses* list to the Configured Nodes list. Once this change has been applied, the selected node is removed from `/var/beowulf/unknown_addresses` and added to `/etc/beowulf/config`. You may also perform this menu operation on multiple, contiguous nodes in the list by simply selecting them prior to your opening the pop-up menu.

Another way to perform this operation is by dragging and dropping nodes using your mouse. Simply select and drag the highlighted node from the *Unknown Addresses* list window and drop it onto the *Configured Nodes* list window. The drag 'n drop operation is functionally equivalent to selecting this menu item.

Move to Ignored

This menu item moves the selected node from the *Unknown Addresses* list to the Ignored Addresses list. Once this change has been applied, the selected node is removed from `/var/beowulf/unknown_addresses` and added to `/etc/beowulf/config`. You may also perform this menu operation on multiple, contiguous nodes in the list by simply selecting them prior to your opening the pop-up menu.

Another way to perform this operation is by dragging and dropping nodes using your mouse. Simply select and drag the highlighted node from the *Unknown Addresses* list window and drop it onto the *Ignored Addresses* list window. The drag 'n drop operation is functionally equivalent to selecting this menu item.

The Configured Nodes List

The *Configured Nodes* list window displays those nodes currently classified as *configured*, with the tag `node`, in the cluster configuration file. This is the list window you will typically be referring to most often when using BeoSetup as these nodes are the ones that are formally treated as members of the cluster. A general discussion of how nodes are classified can be found in the section on Node List Windows. Each node listed in this window contains the items described below, some of which are stored in the cluster configuration file, `/etc/beowulf/config`.

ID

This column displays the node's assigned *number* in the cluster. The node numbers are assigned based on the node's position in the cluster configuration file (the first `node` is given #0, the second is given #1...). While the numbers may appear somewhat arbitrary, they become very important once you consider what it might take to perform maintenance activities on a particular node in a 128 node cluster. If BeoSetup shows an error on node #56, how do you find it? A description of why BeoSetup treats these numbers as important, is found in the section on Numbering Your Cluster's Nodes.

Ethernet Hardware (MAC) Address

This column displays the node's Ethernet hardware (MAC) address. A node's address is stored with its associated entry in the cluster configuration file. As described below, a pop-up menu is available in this list window with an option to modify a node's Ethernet hardware address.

IP Address

This column displays the node's IP address on the cluster's internal network interface. This number is computed from the *node number* described above and the starting address as defined by the cluster's IP address range. For a given row in the list (i.e for a given *node number*), the IP address will remain the same until the IP address range is modified. The IP address range can be changed using the *Network Properties* panel of the **Configuration** dialog box.

State

This column displays the current state of the node as reported by *BProc*, the Beowulf distributed process space software. The possible values for a node's state are: `down`, `unavailable`, `error`, `up` and `boot`. Each state is briefly described below:

- `down` - node is not communicating with the master
- `unavailable` - node has been marked unavailable or off-line
- `error` - node encountered an error during boot
- `up` - node is operating normally and is on-line
- `boot` - node has started, but not yet completed booting

Keep in mind that these states only indicate the condition of the node as reported by the *BProc* daemons. In turn, the *BProc* daemons are only capable of determining a node's state based on the messages communicated (or not communicated, as the case may be) between a node and the master. For example, just because a node's state is not listed as `error` doesn't mean there isn't some undetected hardware problem occurring within the node. These states are only indicative of how the *BProc* software sees the node. As described below, a pop-up menu is available in this list window with options for taking nodes on- and off-line.

User

This column displays the user currently assigned as owner of the node. The value can be a user name, a numeric user id, or the Scyld-defined all encompassing user "any". The user permission setting for a node is analogous to the user ownership feature of files in Linux and basically controls who is allowed to use the node. As described below, a pop-up menu is available in this list window with an option for changing a node's user. A node's user data is stored with its associated entry in the cluster configuration file.

Group

This column displays the group currently assigned as owner of the node. The value can be a group name, a numeric group id, or the Scyld-defined all encompassing group "any". The group permission setting for a node is analogous to the group ownership feature of files in Linux and basically controls who is allowed to use the node. As described below, a pop-up menu is available in this list window with an option for changing a node's group. A node's group data is stored with its associated entry in the cluster configuration file.

Mode

This column displays the execute permissions, i.e. user, group or all.

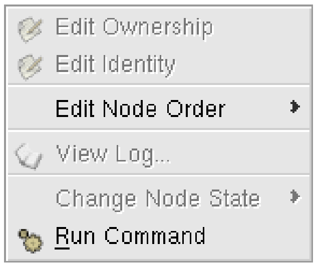
Numbering Your Cluster Nodes

Depending on the size of your cluster, you may or may not pay a great deal of attention to how the individual nodes are numbered. After all, if your cluster is just a handful of commodity PCs sitting on some shelves, what's the big deal. Even a single rack of 8, 16 or 32 nodes may not seem like such a headache. But what about the lab with clusters containing 128 nodes or more? How would you figure out what box to examine if you only knew that node #37 was having problems? Luckily, BeoSetup was designed to make managing clusters of all sizes easy by paying careful attention to how nodes are numbered, and once setup, to maintaining those numbers.

When first setting up your cluster, it's recommended that you power up your compute nodes one-by-one in some logical order relative to how your hardware is arranged. In this way, for example, you'll know that the first box at the top of the rack is node #0, the second box from the top is node #1...and so on. Once you have all nodes in your cluster powered on and numbered (and presumably labeled), BeoSetup will maintain the node numbering of all the `up` nodes in your cluster as other nodes in the cluster are inserted and / or deletion. Examples will hopefully explain this better.

Let's say your cluster contains 3 compute nodes. Node #0 is currently powered off and `down`, while nodes #1 and #2 are `up` and running. If you delete node #0, its address is changed to `off` and it's left in the list as a placeholder so nodes #1 and #2 maintain their current node numbers. As nodes are inserted, moved around or deleted, BeoSetup takes care to ensure the assigned node numbers for all `up` nodes are maintained. Refer to specific operations described below for further details on what effect they have on a cluster's node numbers.

Configured Nodes Pop-up Menu



The *Configured Nodes* list window provides a pop-up menu for performing many operations related to the configuration and maintenance of *configured* nodes. When you right-click anywhere in this window, you are presented with the pop-up menu shown below (beyond the table). Depending on the node being selected, its current state and BeoSetup's current privilege level, some of the items on the menu may be disabled. The following table details the availability of a menu item with respect to the selected node's current state.

Table 2-1. Configured Node List Pop-Up Menu Item Availability vs Node State

Menu Items	Down	Unavailable	Error	Up	Boot
Insert	ena	ena	ena	ena	ena
Edit	ena	dis	dis	dis	dis
Delete	ena	dis	dis	dis	dis
Move to Ignored	ena	dis	dis	dis	dis
Move to Unknown	ena	dis	dis	dis	dis
View Log*	ena	ena	ena	ena	ena
Wake Up	ena	dis	dis	dis	dis
Reboot	dis	ena	ena	ena	ena
Halt	dis	ena	ena	ena	ena
Power Off	dis	ena	ena	ena	ena
Kill	dis	ena	ena	ena	ena
Mark Up	dis	ena	dis	dis	dis
Mark Unavailable	dis	dis	dis	ena	dis

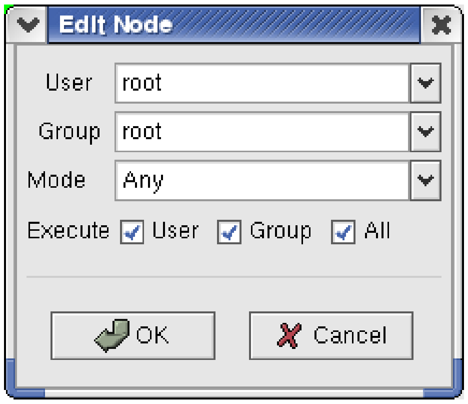
* The **View Log...** menu item is available to all users, regardless of their current privilege level. Of the remaining menu items shown in the table above, those listed as *enabled* are actually only available to users running BeoSetup with full privileges. These menu items will be disabled for those running with limited privileges.

Some of the functionality provided with the pop-up menu shown below, can be performed using drag 'n drop operations with your mouse. These operations are discussed where appropriate. You can also use drag 'n drop to reorder the numbering of the nodes in your cluster. Simply drag a node from one location to another to renumber the list. Note that you are only able to drag a node when its state is anything other than `up`, `unavailable`, `error` or `boot`. Also note that when dragging nodes around the list, the position of `up` nodes is automatically maintained by BeoSetup.

For any changes you make in this window to take effect, you must be sure to apply them.

Edit Ownership

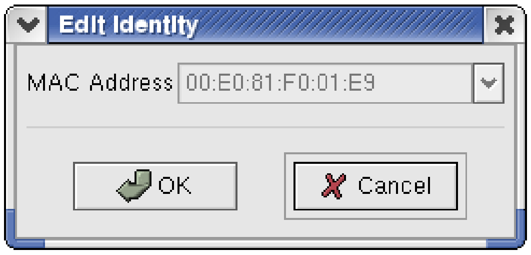
This menu item allows you to edit the user, group, and mode of a node.



User root
Group root
Mode Any
Execute [✓ User] [✓ Group] [✓ All]

Edit Identity

This menu item lets you modify the Ethernet hardware (MAC) address for the currently selected node in the *Configured Nodes* list. When selected, you are presented with the following dialog box:



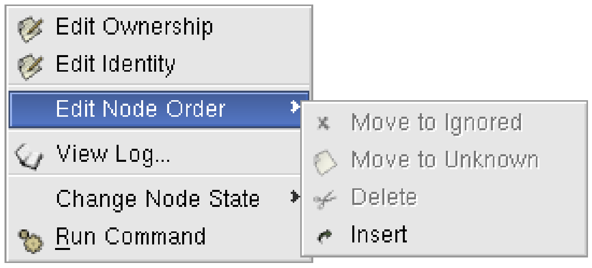
MAC Address 00:E0:81:F0:01:E9

When initially displayed, the edit widget in the dialog box contains the selected node's current hardware address. Simply change the address to the desired value and select the **OK** button. If your changes contain any errors, an error message box will be displayed and your changes will be ignored. Simply reselect the menu item and try again. Once your changes have been applied, the selected node's entry in the cluster configuration file is replaced with the new address. If you decide you no longer wish to change the node's hardware address, simply select the **Cancel** button.

Note that a node's Ethernet hardware (MAC) address is determined by the network interface card plugged in to the node. It should not be arbitrarily changed to another value unless you really know what you're doing. The capability to modify the address is primarily provided to change `off` node's to real addresses and as a convenience when replacing network cards.

Edit Node Order

This group of menus will allow you to move, delete, and insert from the list of configured nodes.



Move to Ignored

This menu item moves the selected node from the *Configured Nodes* list to the Ignored Addresses list. Once this change has been applied, the selected node is re-tagged in `/etc/beowulf/config` from `node` to `ignore`. As described above in the section on Numbering Your Cluster Nodes, when moving a node out of this list an `off` node may be automatically inserted by BeoSetup to maintain the current numbering of `up` nodes.

Another way to perform this operation is by dragging and dropping nodes using your mouse. Simply select and drag the highlighted node from the *Configured Nodes* list window and drop it onto the *Ignored Addresses* list window. The drag 'n drop operation is functionally equivalent to selecting this menu item.

Move to Unknown

This menu item moves the selected node from the *Configured Nodes* list to the Unknown Addresses list. Once this change has been applied, the selected node is removed from `/etc/beowulf/config` and added to `/var/beowulf/unknown_addresses`. As described above in the section on Numbering Your Cluster Nodes, when moving a node out of this list an `off` node may be automatically inserted by BeoSetup to maintain the current numbering of `up` nodes.

Another way to perform this operation is by dragging and dropping nodes using your mouse. Simply select and drag the highlighted node from the *Configured Nodes* list window and drop it onto the *Unknown Addresses* list window. The drag 'n drop operation is functionally equivalent to selecting this menu item.

Delete

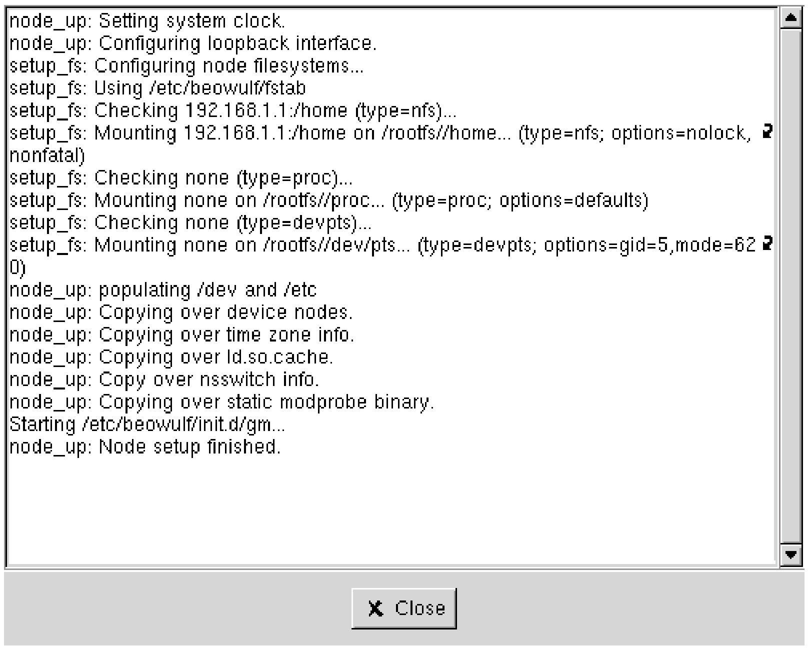
This menu item deletes the selected node from the *Configured Nodes* list. Once this change has been applied, the selected node is removed from `/etc/beowulf/config`. As described above in the section on Numbering Your Cluster Nodes, when moving a node out of this list an `off` node may be automatically inserted by BeoSetup to maintain the current numbering of `up` nodes.

Insert

This menu item allows you to insert a node into the *Configured Nodes* list directly above the node you selected when clicking your right mouse button. (If you happened to right click in the white space at the bottom of the list window, the new node is simply added to the end of the list.) The new node is initially given an address of `off` that can be changed using the **Edit...** menu item discussed below. Once changed and applied, the newly inserted node is written to `/etc/beowulf/config` using the `node` tag. As described above in the section on Numbering Your Cluster Nodes, BeoSetup will automatically insert the new node where appropriate so as to maintain the current numbering of `up` nodes.

View Log...

This menu item lets you view the boot / status log for the selected node. When selected, the dialog box shown below is displayed. Its main window contains the contents of the node's activity log file. The name of the log file is shown in the window's title bar: `/var/log/beowulf/node.#`, where '#' is replaced with the selected node's number. If new information should be added to the log file while this window is displayed, the window will automatically update with the new information. This menu item is available to users of all privilege levels, all the time.

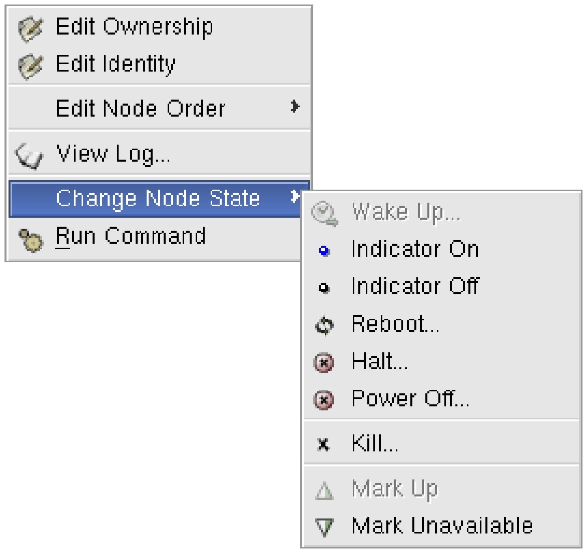


```
node_up: Setting system clock.
node_up: Configuring loopback interface.
setup_fs: Configuring node filesystems...
setup_fs: Using /etc/beowulf/fstab
setup_fs: Checking 192.168.1.1:/home (type=nfs)...
setup_fs: Mounting 192.168.1.1:/home on /rootfs//home... (type=nfs; options=nolock,
nonfatal)
setup_fs: Checking none (type=proc)...
setup_fs: Mounting none on /rootfs//proc... (type=proc; options=defaults)
setup_fs: Checking none (type=devpts)...
setup_fs: Mounting none on /rootfs//dev/pts... (type=devpts; options=gid=5,mode=62
0)
node_up: populating /dev and /etc
node_up: Copying over device nodes.
node_up: Copying over time zone info.
node_up: Copying over ld.so.cache.
node_up: Copy over nsswitch info.
node_up: Copying over static modprobe binary.
Starting /etc/beowulf/init.d/gm...
node_up: Node setup finished.
```

Simply select the **Close** button to remove the dialog box from the screen.

Change Node State

This group of menus will allow you to change the state of configured nodes.



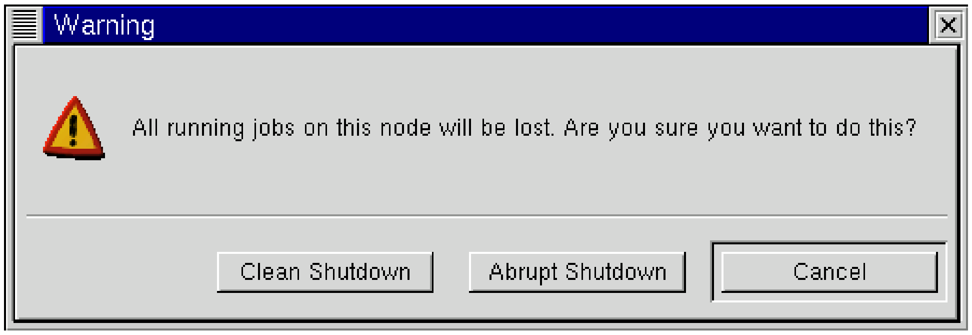
Wake Up...

This menu item can be used to send a *Wake On LAN* message to the selected node. *Wake-On-LAN* is the generic name for the AMD "Magic Packet" technology. It's very similar to the PCMCIA modem "wake on ring" signal line. The basic idea is that the network adapter has a very-low-power mode to monitor the network for special packet data that will wake up the machine. When this menu item is selected, the script `/usr/lib/beoboot/bin/node_wake` is executed, which uses the `ether-wake` utility to generate and send the special data packet to the node.

Indicator on
Indicator off
Reboot...
Halt...
Power Off...
Kill...

These four menu items all perform somewhat related functions. In each case, the master node sends commands to the selected compute node using the script `/usr/lib/beoboot/bin/node_down`. The selected operation is passed to the script as a parameter. **Reboot** will cycle power off and then on again to the node. **Halt** will suspend the node. **Power Off** will shut off a node completely. Finally, **Kill** will totally disavow the node by changing its state to **down**.

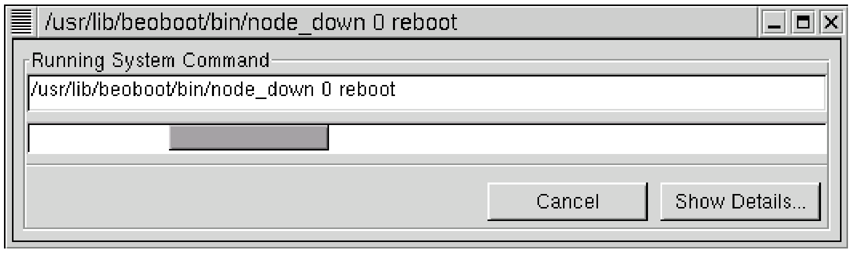
After selecting any one of these items, you are presented with the following message box:



All running jobs on this node will be lost. Are you sure you want to do this?

If you select **Clean Shutdown**, the selected operation is performed by running the `node_down` script as noted above. If you select **Abrupt Shutdown**, the script is not run but instead the new state is directly passed on to the Beowulf distributed process space software and the node responds accordingly. Selecting the **Cancel** button closes the message box and the node is left alone.

If you selected **Clean Shutdown**, the following progress dialog box is displayed:



```
/usr/lib/beoboot/bin/node_down 0 reboot
```

Though the dialog box contains a **Cancel** button, it is not recommended you select it for any of these operations. The button is included by default in this Scyld-created *System Command* dialog but using it with these operations is very likely to leave the selected node in an ambiguous state. When the operation completes, the **Cancel** button is renamed to **Close**, which can then be selected to close the dialog box and end the operation. At any time, you can click the **Show Details...** button to enlarge the dialog and view the output from the execution of the `node_down` script.

Mark Up
Mark Unavailable

These two menu items allow you to change the selected node's state. When the node's state is `up`, you can mark it as `unavailable`. Not surprisingly, when the node's state is `unavailable`, you can mark it as `up`. As described above, the `up` and `unavailable` states are analogous to the familiar computer terms on-line and off-line. You may want to temporarily mark a node as `unavailable` to take if off-line for performing some routine maintenance. While the node is off-line, it will not be automatically used by the scheduler for running any jobs. Once you've completed your maintenance activities, simply mark the node as `up` and it's back on-line again.

The Ignored Addresses List

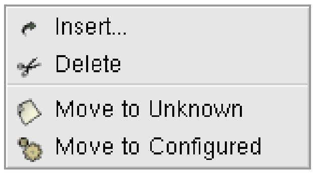
The *Ignored Addresses* list window displays the *ignored* nodes currently listed in the cluster configuration file. Each item in this node list window shows the Ethernet hardware (MAC) address of a node listed as `ignore` in this file. A general discussion of how nodes are classified can be found in the section on Node List Windows.

Ignored Addresses Pop-up Menu

To reclassify an *ignored* node, you may drag the node from the *Ignored Addresses List* and drop it over one of the other two node list windows: Unknown Addresses or Configured Nodes. You must be running BeoSetup with full privileges to use the drag 'n drop features.

You may perform the same operation using a pop-up menu. When you right-click anywhere in this window, you are presented with the pop-up menu shown below. If you are running BeoSetup with limited privileges, all the items in the pop-up menu will be disabled. With full privileges, the sensitivity of the menu items depends on where in the window you click. When right-clicking over a node, all items in the menu are enabled. When right-clicking over empty space in the list window, only the **Insert...** item is enabled.

For any changes you make in this window to take effect, you must be sure to apply them.



Insert...

This menu item adds another node to the bottom of the *Ignored Addresses* list. The item is initially given an address of `off`. Once changed and applied, the newly inserted *ignored* node is written to the file `/etc/beowulf/config` using the `ignore` tag. Any *ignored* nodes in the list window with an address of `off`, are not written to the cluster configuration file when changes are applied.

Delete

This menu item deletes the selected node from the *Ignored Addresses* list. Once this change has been applied, the selected node is removed from `/etc/beowulf/config`.

Move to Unknown

This menu item moves the selected node from the *Ignored Addresses* list to the Unknown Addresses list. Once this change has been applied, the selected node is removed from `/etc/beowulf/config` and added to `/var/beowulf/unknown_addresses`. You may also perform this menu operation on multiple, contiguous nodes in the list by simply selecting them before opening the pop-up menu.

Another way to perform this operation is by dragging and dropping nodes using your mouse. Simply select and drag the highlighted node from the *Ignored Addresses* list window and drop it onto the *Unknown Addresses* list window. The drag 'n drop operation is functionally equivalent to selecting this menu item.

Move to Configured

This menu item moves the selected node from the *Ignored Addresses* list to the Configured Nodes list. Once this change has been applied, the selected node is re-tagged in `/etc/beowulf/config` from `ignore` to `node`. You may also perform this menu operation on multiple, contiguous nodes in the list by simply selecting them before opening the pop-up menu.

Another way to perform this operation is by dragging and dropping nodes using your mouse. Simply select and drag the highlighted node from the *Ignored Addresses* list window and drop it onto the *Configured Nodes* list window. The drag 'n drop operation is functionally equivalent to selecting this menu item.

Chapter 3. Configuring the Cluster Manually

This chapter covers how to configure a cluster without using the graphical setup tool **beosetup**. It is highly recommended that the beginning administrator use the **beosetup** GUI tool for configuring the cluster. However, if you either can't get X-windows running on the master node after installation or you want to remotely set up the cluster and X forwarding is too slow for you, then this chapter is for you.

Configuration Files

All setup information about the cluster is stored in configuration files. The **beosetup** tool simply edits these files. You may prefer to edit these files directly in your favorite editor.

`/etc/beowulf/config`

This is the main configuration file for the cluster. Some of the information in this file is used to create the stage 1 boot image. Stage 1 images are most commonly placed on floppy disks to boot the slave, but may also be used during PXE boot. See the *Reference Guide* for more details. Other values that this file sets for configuration of the cluster include:

- the name of the network interface connected to the internal network
- the IP address and netmask of the network interface
- the IP address range to assign to the compute nodes
- the list of shared libraries to cache on the compute nodes
- compute node filesystem startup policy
- the name of second stage boot file to send to the compute nodes at boot
- the name of network ports used by the file transport services
- the default kernel and command line to use when creating a boot file
- the hardware addresses of all the identified nodes in the cluster

The details of these settings are described in the *Reference Guide*, but we'll briefly show some of the highlights here.

When new nodes boot up they send Reverse Address Resolution Protocol (RARP) packets to the network in order to get an IP address assigned to them. The master will detect these RARP packets and record the hardware address that the packets are coming from in the file `/var/beowulf/unknown_addresses`. At any time one can copy addresses out of this file and into the `/etc/beowulf/config` file with the keyword *node* prefixing the hardware address. The order of the *node* lines in the `config` file dictate the node number and IP address.

Another thing you might want to do is add a shared library to the list of libraries cached on the compute nodes. Just add the name to the end of the line that starts with the keyword *libraries*. The space character is the delimiter. You may also specify entire directories of shared libraries to be cached on this line (by just inserting the directory path). Note only shared libraries in these directories will be cached, not binaries or other data files.

The IP address range should be kept to a minimum. Having a few spare addresses is a good idea to allow growth in the cluster, but don't add too many as all the cluster utilities are based on looping through this range. Having a large number of addresses that will never be used will be an unnecessary waste of resources.

See the *Reference Guide* for more options and details about this file.

`/etc/beowulf/fdisk`

This file contains the partition table information as read by **beofdisk**. Normally, this file should not be edited by hand. For details see the *Reference Guide*.

You may create separate versions of this file that end with the node number (e.g. `/etc/beowulf/fdisk.3`). The master's beoboot software will look for these files before using the general `/etc/beowulf/fdisk` file.

`/etc/beowulf/fstab`

This is the file system table for the mount points of the partitions on the compute nodes. It should be familiar to anyone who has dealt with a `/etc/fstab` file in a standard UNIX system. For details see the *Reference Guide*.

You may create separate versions of this file that end with the node number (e.g. `/etc/beowulf/fstab.3`). The master's beoboot software will look for these files before using the general `/etc/beowulf/fstab` file.

Command Line Tools

bpstat

The command **bpstat** can be used to quickly check the status of the cluster nodes and/or see what processes are running on the compute nodes. See the *Reference Guide* for details on usage.

bpctl

To reboot or set the state of a node via the command line, one can use the **bpctl** command. For example, to reboot node 5:

```
[user1@cluster]$ bpctl -S 5 -R
```

As the administrator, you may at some point have reason to prevent other users from running new jobs on a specific node, but you do not want to shut it down. For this purpose we have the *unavailable* state. When a node is set to *unavailable* non-root users will be unable to start new jobs on that node, but existing jobs will continue running. To do this, set the state to *unavailable* using the **bpctl** command. For example, to set node 5 to unavailable:

```
[user1@cluster]$ bpctl -S 5 -s unavailable
```

node_down

If you are mounting local filesystems on the compute nodes, you should shut down the node cleanly so that the filesystems on the hard drives stay in a consistent state. The **/usr/lib/beoboot/bin/node_down** script does exactly this. It takes two arguments, the first is the node number, and the second is the state to which you want the node to go. For example, to cleanly reboot node 5:

```
[user1@cluster]$ /usr/lib/beoboot/bin/node_down 5 reboot
```

or to instead cleanly poweroff node 5:

```
[user1@cluster]$ /usr/lib/beoboot/bin/node_down 5 pwroff
```

node_down works by first setting the node's state to unavailable, then remounting the filesystems on the compute node read-only, then calling **bpctl** to change the node state. This can all be done by hand, but the script saves some keystrokes.

Interconnects

There are many different types of network fabric one can use to interconnect the nodes of your cluster. The least expensive and most common is Fast (100Mbps) and Gigabit (1000Mbps) Ethernet. Other cluster-specific network types, such Infiniband, offer lower latency, higher bandwidth and features such as RDMA (Remote Direct Memory Access).

Ethernet

Switching fabric is always the most important (read: expensive) part of any interconnected sub-system. Ethernet switches up to 48 ports are extremely cost effective; however, anything larger becomes expensive quickly. Intelligent switches (those with software monitoring and configuration) add little value in a Beowulf cluster, and can be extremely expensive.

How do I add a new Ethernet driver?

Drivers for most Ethernet adapters are included with Scyld Beowulf and are supported out of the box for both the master and the compute nodes. If you find that your card is not supported and there exists a Linux source code driver for it, you need to compile it against the master's kernel and the BeoBoot kernel (for compute node boot). For details on adding new kernel modules see the Section called *Adding New Kernel Modules*.

Gigabit Ethernet vs. Specialized Cluster Interconnects

Surprisingly, the packet latency for Gigabit Ethernet is approximately the same as for Fast Ethernet. In some cases the latency may even be slightly higher, as the network is tuned for high bandwidth with low system impact utilitization. Thus Gigabit Ethernet will not give significant improvement over Fast Ethernet to fine grained communication-bound parallel applications, where specialized interconnects have a significant performance advantage. However, Gigabit Ethernet can be very efficient when doing large I/O transfers, which may dominate the overall run time of a system.

Other Interconnects

Infiniband (IB) is a new, standardized interconnect for system area networking. While the hardware interface is an industry standard, the details of the hardware device interface are vendor specific and rapidly change. Contact Scyld support for details on which IB host adapters and switches are currently supported.

With the exception of unique network monitoring tools for each, the administrative and end user interaction is unchanged from the base Scyld Beowulf system.

Useful Kernel Command Line Options

There are a large number of different command line options that you can pass to the kernel. This section covers some of them. For information on how to feed these kernel command line options to the compute node kernels, see the section on `/etc/beowulf/config` and the chapter on booting.

mem=

One of the most frequently used command line arguments is setting the memory size. There are some old BIOS's out there that do not correctly report to Linux how much memory you have, so Linux ends up seeing only 64M of RAM. If your compute nodes come up and **beostatus** shows them only having 64M of RAM, but you know the compute nodes have more, you will want to use this option. You specify it as `mem=XXXM` where XXX is how much RAM you have. For example, you might use `mem=128M` if your compute nodes have 128 megs of RAM.

apic

This turns on APIC support on the compute node. APIC is disabled by default as it causes some machines to have problems booting. Intel has two different mechanisms for invoking interrupts. The old style is called XT-PIC. The new style, which works better with SMP systems, is called APIC. However, not every motherboard and chipset works correctly with the new APIC. If you find that your cluster nodes kernel panic or crash immediately upon boot, you probably want to turn off APIC. If you have many devices that generate interrupts (e.g. hard disk controllers, network adapters) you may want to try turning on APIC to see if there's any performance advantage for your cluster.

panic=

This specifies that after the kernel panics, it should reboot. After the equals sign place a number specifying how long to wait before rebooting. So if you specify `panic=30`, the kernel will wait 30 seconds after a panic, then reboot. BeoBoot automatically adds `panic=30` to phase 2 boot images.

apm=

This option allows you to specify APM options on the compute node. After the equals sign, you can put `on` to turn APM fully on, `off` to turn it completely off, `debug` to turn on debugging, and `power-off` to only turn on the power off part of APM. APM is not SMP-safe in the kernel and will auto-disable itself if it is turned on on an SMP box. However, the power-off functionality of it is SMP safe, so if you want to be able to power off SMP boxes, you can do so by specifying `apm=power-off`. `apm=power-off` is the default kernel command line in `/etc/beowulf/config`.

console=

This option is used to select which device(s) to use for console output.

```
The format of this option is:

        console=device,options

        device:         tty0 for the foreground virtual console
                        ttyX for any other virtual console
                        ttySx for a serial port

        options:        For the serial port this defines the
                        baud rate/parity/bits of the port in the
                        format BBBBPN, where BBBB is the speed,
                        P is parity (n/o/e), and N is bits. The default
                        setting is 9600n8. The maximum baud rate is 115200.
```

For example, to use the serial port at the maximum baud rate, the option would be specified as `console=ttyS0,115200`.

Adding New Kernel Modules

Many device drivers are included with Scyld Beowulf and are supported out-of-the-box for both the master and the compute nodes. If you find that your device, such as an Ethernet adapter, is not supported and there exists a Linux source code driver for it, then you will need to build the driver modules for the master's kernel and the BeoBoot kernel (for compute node boot). To do this, if you haven't already done so, you'll need to install the RPM of kernel source code. Next, compile the source code of the driver two times (once for the master kernel (Uniprocessor or SMP)) and once for the beoboot kernel using these different set of extra GNU C Compiler (gcc) options:

- For Uniprocessor: `-D__BOOT_KERNEL_SMP=0 -D__BOOT_KERNEL_UP=1`
- For SMP: `-D__BOOT_KERNEL_SMP=1 -D__BOOT_KERNEL_UP=0`
- For BeoBoot: `-D__BOOT_KERNEL_SMP=0 -D__BOOT_KERNEL_UP=0 -D__module__beoboot`

The compiled modules must be installed in the appropriate directories under `/lib/modules`. For example, if you are currently running under the 2.4.25-13 kernel version, the compiled module for an ethernet driver for a uniprocessor kernel would be put in the `/lib/modules/2.4.25-13_Scyld/kernel/drivers/net` directory, the module for a SMP kernel would be put in `/lib/modules/2.4.25-13_Scyldsmp/kernel/drivers/net` directory, and the module built for the beoboot kernel would be put in the `/lib/modules/2.4.25-13_Scyldbeoboot/kernel/drivers/net` directory.

If you use the Scyld Beoboot stage 1 system to boot (rather than network booting), and the updated driver is required to support new hardware, you must generate new stage 1 boot media (floppy disk, CD-ROM or DVD). Edit the file beowulf configuration file `/etc/beowulf/config`. Add the name of the driver to the `bootmodule` list (you can add more bootmodule lines if needed.) You may find that an updated driver list results in a boot image too large to fit onto the boot media. If that happens you will need to delete some of the unused drivers listed in the `bootmodule` lines to allow the image to fit.

Next, you need to configure how the device driver gets loaded. You can set it up so that the device driver only loads if the specific device is found on the compute node. To do this, you need to add the PCI vendor/device ID pair to the PCI table information in the `/usr/share/kudzu/pcitable` file. You can figure out what these values are by using a combination of **/sbin/lspci** and **/sbin/lspci -n**. You can also set it up so that your new kernel module is always loaded on the compute nodes. You do this by adding a `modprobe` line. The line should look like:

```
modprobe <module>
```

where <`module`> is the kernel module in question.

Finally, you can regenerate beoboot images by running **beoboot -1 -f** (for stage 1) and **beoboot -2 -n** (for stage 2). For more details see the Section called *Creating Boot Images* in Chapter 8.

Chapter 4. Monitoring the Status of the Cluster

Scyld Beowulf provides several methods to monitor cluster performance and health, with GUI, command line, web, and "C" language interfaces. In general, these tools provide easy access to the information available through the Linux */proc* system as well as BProcinformation for each of the cluster nodes. Within each of these interfaces is also the ability to customize the display format. The monitoring programs are available to both administrators and regular users, as they provide no cluster commanding capabilities.

Overview

The cluster monitoring interfaces, **beostatus**, **beostat**, and **libbeostat** are installed during the normal installation procedure.

The Beostatus cluster monitoring utility displays CPU utilization, memory usage, swap usage, disk usage, and network utilization. It defaults to a bar graph X-window GUI, but can display the information in several text formats. And for large clusters, a small footprint GUI can be selected, with colored dots depicting the overall status on each node.

For a detailed command line display, **beostat** can be used. With no options **beostat** will list */proc/cpuinfo*, */proc/meminfo*, */proc/loadavg*, */proc/net/dev*, and */proc/stat* . Or you can use the arguments to select any combination of those statistics.

For users that wish to create their own custom displays or create more sophisticated resource scheduling software, Scyld provides "C" language interfaces through the *libbeostat* library. These functions are the same as those used by **beostatus**, **beostat** and the batch scheduler *BBQ*. They provide convenient access to the data structures in */proc* and BProc node state information.

Web enabled monitoring is also available, but is not installed by default. This facility provides the same information as the **beostatus** GUI, but in HTML format. To enable, an Apache web server must be installed on the master, and the **beowebenable** script run. See Reference Guide for more information.

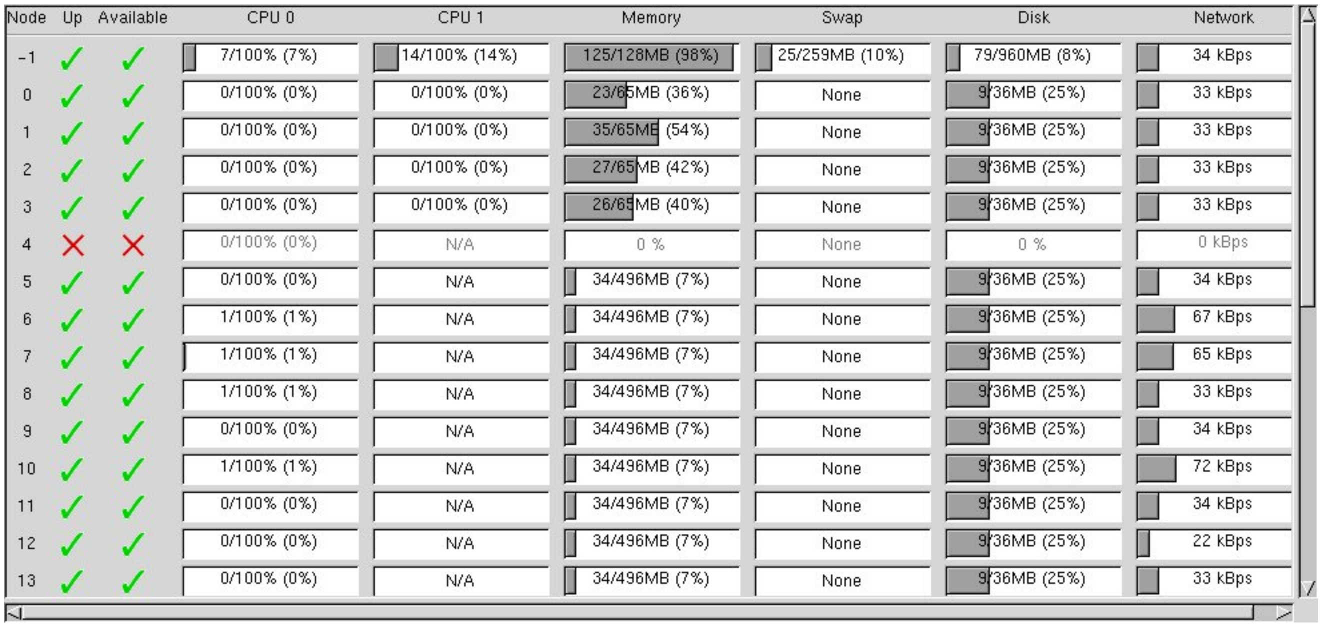
Underlying the monitoring facility are two daemons, *sendstats*, and *recvstats*. You can check that these are running with the **ps -aux |grep stats** command. The *sendstats* daemon, as the name implies, sends the status information from each of the nodes as well as the master to the *recvstats* daemon running only on the master. On the master these daemons are started in the */etc/rc.d/init.d/beowulf* script. The nodes start the *sendstats* daemon in the *boot.c* file. Nominally the daemons operate in a command/response fashion, where the *recvstats* daemon sends out multicast packets on port 3040 to request updates from all of the nodes. But the nodes also perform an auto-transmit on 59 seconds intervals to assist in trouble shooting. For more information on all of the daemon options, check the reference guide.

Using the Data

The outputs from the monitoring utilities can provide insights into obtaining the best performance from your cluster. If you are new to cluster computing, you will want to note the relationship between the different machine resources including CPU utilization, swap usage, and network utilization. Low CPU usage, but high network traffic, might indicate that your system is I/O bound, and could benefit from faster network components. Low network load and high CPU usage could improve with faster CPUs. Medium to high swap usage is always bad. This indicates that memory is oversubscribed and application pieces must be moved to the much slower disk subsystem. This can be a substantial bottleneck, and is a sure sign that additional RAM is needed. Any of these issues could be helped with application optimization, but sometimes it's more economical to add resources than to change working software. Also note that the *BBQ* batching software will try to keep the CPUs at the 80% level. You might choose to lower that threshold if the other elements are out of balance. Monitoring memory usage is not as valuable because the Linux kernel correctly tries to keep as much information cached or buffered as possible and will tend to keep memory full. Disk usage is more of a hard limit than a performance issue. If you see disk usage approaching 95%, you are headed for trouble, as the Linux kernel does not degrade gracefully when the disk is full.

Beostatus

The **beostatus** GUI display is started during Gnome X-window initialization with an entry in the `/usr/share/gnome/default.session` file. Gnome is started with the **startx** command, or as part of Linux runlevel 5 as defined in the `/etc/inittab` file. **beostatus** supports four different types of display generation, all of which can be operated simultaneously. Output in bar graph mode is the default and is provided by a Gnome/GTK+ GUI display. This display is updated once every 5 seconds by default, but may be changed using the **-u** option.



Node	Up	Available	CPU 0	CPU 1	Memory	Swap	Disk	Network
-1	✓	✓	7/100% (7%)	14/100% (14%)	125/128MB (98%)	25/259MB (10%)	79/960MB (8%)	34 kBps
0	✓	✓	0/100% (0%)	0/100% (0%)	23/65MB (36%)	None	9/36MB (25%)	33 kBps
1	✓	✓	0/100% (0%)	0/100% (0%)	35/65MB (54%)	None	9/36MB (25%)	33 kBps
2	✓	✓	0/100% (0%)	0/100% (0%)	27/65MB (42%)	None	9/36MB (25%)	33 kBps
3	✓	✓	0/100% (0%)	0/100% (0%)	26/65MB (40%)	None	9/36MB (25%)	33 kBps
4	✗	✗	0/100% (0%)	N/A	0 %	None	0 %	0 kBps
5	✓	✓	0/100% (0%)	N/A	34/496MB (7%)	None	9/36MB (25%)	34 kBps
6	✓	✓	1/100% (1%)	N/A	34/496MB (7%)	None	9/36MB (25%)	67 kBps
7	✓	✓	1/100% (1%)	N/A	34/496MB (7%)	None	9/36MB (25%)	65 kBps
8	✓	✓	1/100% (1%)	N/A	34/496MB (7%)	None	9/36MB (25%)	33 kBps
9	✓	✓	0/100% (0%)	N/A	34/496MB (7%)	None	9/36MB (25%)	34 kBps
10	✓	✓	1/100% (1%)	N/A	34/496MB (7%)	None	9/36MB (25%)	72 kBps
11	✓	✓	0/100% (0%)	N/A	34/496MB (7%)	None	9/36MB (25%)	34 kBps
12	✓	✓	0/100% (0%)	N/A	34/496MB (7%)	None	9/36MB (25%)	22 kBps
13	✓	✓	0/100% (0%)	N/A	34/496MB (7%)	None	9/36MB (25%)	33 kBps

Figure 4-1. Scyld Beostatus Monitor Tool

Beostatus File Menu

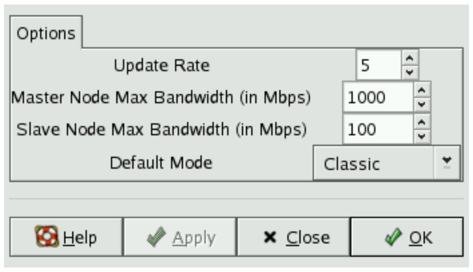
The **Beostatus File Menu** contains the following selections:

- **Preferences**
- **Quit**

Refer to the appropriate section as noted below for further details.

Beostatus Preferences

The **Preferences** item on the **Beostatus File Menu** displays the Beostatus Options panel. This panel permits modification of `Update Rate, Master Node Max Bandwidth (in Mbps), Slave Node Max Bandwidth (in Mbps)` or `Default Modes` values.


Options

Update Rate 5

Master Node Max Bandwidth (in Mbps) 1000

Slave Node Max Bandwidth (in Mbps) 100

Default Mode Classic

Figure 4-2. Scyld Beostatus Options panel

Quit

Selecting **Quit** from the **Beostatus File Menu** results in the closing of the BeoStatus panel.

Beostatus Mode Menu

The **Beostatus Mode Menu** supports seven different types of display generation, all of which can be operated simultaneously. Output in bar graph mode, (Classic,) is the default and is provided by a Gnome/GTK+ GUI display. This display is updated once every 5 seconds, but the refresh rate may be changed using the **-u** option.

Beostatus dots mode

Output in dots mode, **-d**, provides a Gnome/GTK+ GUI display. Each node is represented by a colored dot. This output is intended for quick overviews and for situations where the screen size needed for the full display for large clusters is unavailable.

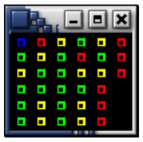


Figure 4-3. Scyld Beostatus Monitor Tool small footprint

In dots mode,

`red    - No access, node state` **`down`**`yellow - Admin access only, node state` **`unavailable`**`,`**`boot`**`,`**`error`**

```
green  - Ready, node state up and node load less than/equal 48%
blue   - Busy, node state up and  node load greater than 48%

Note - SMP is considered for node load calculation as load(CPU1) + load(CPU2) > 48%.
```

Output in curses mode, **-c**, prints a column header and a line for each node without a linefeed. This continuous output provides a method to monitor the system over text only connections, such as the installed **ssh** server.

```
       Beostatus - 3.0
Node     State   CPU 0  Memory  Swap   Disk   Network
 -1         up   2.5%  91.7%   0.0%   9.2%   1 kBps
  0         up   0.2%  20.5%   0.0%  25.0%   1 kBps
  1         up   0.1%  20.5%   0.0%  25.0%   1 kBps
  2         up   0.1%  20.5%   0.0%  25.0%   1 kBps
  3         up   0.2%  20.4%   0.0%  25.0%   1 kBps
  4         up   0.1%  20.3%   0.0%  25.0%   1 kBps
  5         up   0.1%  20.3%   0.0%  25.0%   1 kBps
  6         up   0.2%  20.6%   0.0%  25.0%   1 kBps
  7         up   0.1%  20.4%   0.0%  25.0%   1 kBps
```

Output in text mode **-t** prints a line for each node followed by a linefeed. This continuous output provides a method to collect usage and performance statistics by directing the output to a file, with a command such as **beostatus -t > /tmp/stat.log**. This file could then be parsed to generate long term usage or trending data. Note that the display format provides for ease of parsing with a fixed string prior to each variable data element.

```
Node: -1  CPU0: 7.2%  Mem: 91.3%  Swap: 0.0%  Disk:  9.2% Net: 1 kBps
Node:  0  CPU0: 0.2%  Mem: 20.5%  Swap: 0.0%  Disk: 25.0% Net: 1 kBps
Node:  1  CPU0: 0.1%  Mem: 20.4%  Swap: 0.0%  Disk: 25.0% Net: 1 kBps
Node:  2  CPU0: 0.3%  Mem: 20.3%  Swap: 0.0%  Disk: 25.0% Net: 1 kBps
Node:  3  CPU0: 0.2%  Mem: 20.3%  Swap: 0.0%  Disk: 25.0% Net: 1 kBps
Node:  4  CPU0: 0.3%  Mem: 20.5%  Swap: 0.0%  Disk: 25.0% Net: 1 kBps
Node:  5  CPU0: 0.3%  Mem: 20.3%  Swap: 0.0%  Disk: 25.0% Net: 1 kBps
Node:  6  CPU0: 0.2%  Mem: 20.6%  Swap: 0.0%  Disk: 25.0% Net: 1 kBps
Node:  7  CPU0: 0.3%  Mem: 20.4%  Swap: 0.0%  Disk: 25.0% Net: 1 kBps
```

Beostat

The **beostat** utility is both a command line program as well as a set of "C" language library functions. The command line, **beostat**, provides a text listing of the information from the */proc* structure on each node. The output from a single node follows:

```
============== Node: .0 (index 0) ==================

 *** /proc/meminfo *** Thu Apr 22 15:53:39 2004
        total:    used:    free:  shared: buffers:  cached:
Mem:  2095665152 20357120 2075308032        0  2265088        0
Swap:        0        0        0
MemTotal:   2046548 kB
```

```
MemFree:    2026668 kB
MemShared:        0 kB
Buffers:       2212 kB
Cached:           0 kB
SwapTotal:        0 kB
SwapFree:         0 kB

 *** /proc/loadavg *** Thu Apr 22 15:53:39 2004
0.00 0.00 0.00 0/14 0

 *** /proc/net/dev *** Thu Apr 22 15:53:39 2004
Inter-|   Receive                                                |  Transmit
 face |bytes    packets errs drop fifo frame compressed multicast|bytes    packets errs drop fifo co
  eth0:18446744072884698819 40447277    0    0    0     0          0         0 18446744072841335190
  eth1:18446744072884698819 40447277    0    0    0     0          0         0 18446744072841335190

 *** /proc/stat ***
cpu0 521158 0 84226 53055975            Thu Apr 22 15:53:39 2004
cpu1 803313 0 33899 52824147            Thu Apr 22 15:53:39 2004

 *** statfs ("/") *** Thu Apr 22 15:53:39 2004
path:           /
f_type:         0x1021994
f_bsize:        4096
f_blocks:       255818
f_bfree:        253575
f_bavail:       253575
f_files:        255818
f_ffree:        255743
f_fsid:         000000 000000
f_namelen:      255
```

The **beostat** library, *libbeostat* contains the "C" language functions listed below. You compile with the header files `sys/bproc.h` and `sys/beostat.h` adding the linker commands **-lbproc -lbeostat**.

```
beostat-get-cpu-count
beostat-get-name
beostat-get-time
beostat-get-cpuinfo-x86
beostat-get-meminfo
beostat-get-loadavg
beostat-get-net-dev
beostat-get-stat-cpu
beostat-get-MHz
beostat-get-statfs-p
beostat-get-last-multicast
beostat-set-last-multicast
beostat-get-cpu-percent
beostat-get-net-rate
beostat-get-disk-usage
beostat-count-idle-cpus
beostat-count-idle-cpus-on-node
beostat-get-avail-nodes-by-id
beostat-is-node-available
```

Bpstat

bpstat displays a text only snapshot of the current cluster state/configuration.

```
[root@cluster]# bpstat
Node(s)                         Status       Mode       User       Group
16-31                            down         ---------- root       root
0-15                             up           ---x--x--x root       root
```

Using the **-p** flag you can view the ghosted PIDs for each user process running on the nodes. This ghost PID is the I/O forwarding process on the master. You can then pipe the **ps** command into **grep** to get the command string associated with it. For example, **ps -aux |grep 8370** . Normal process signaling will work with these PIDs such as **kill -9 8369**.

```
PID Node Ghost
8367 0 -1
8368 1 -1
8369 2 -1
8370 3 -1
```

See the reference guide for more details on all of the command options.

Chapter 5. Managing Users on the Cluster

In order for someone to gain access to the cluster, they must first be given a user account. Once the user has an account, you can control what cluster resources he or she will have access to. Accounts are managed using the same tools that are available with most Linux distributions.

Managing User Accounts

Adding/Removing users

The **useradd** command is used to add a new user to the system. It is good practice to also give each user their own unique home directory. Note that a home directory is required when using the batch queuing utility, *BBQ*. The **useradd** takes an argument that is the new user's login name and creates a home directory named `/home/<username>` as shown below:

```
useradd <username>
```

where <username> is the login name of the new user. After you add the user, you will want to give them a default password using the **passwd** command. If you don't do this, they won't be able to log in! To use the **passwd** command, just give it a single argument that is the username as shown below:

```
passwd <username>
```

If you ever want to remove a user from your cluster, you should use the **userdel** command. This command takes a single argument that is the username. Common usage is:

```
userdel <username>
```

By default, **userdel** does not remove the users home directory. If you wish it to remove the user's home directory, you will have to give it the -r option as shown:

```
userdel -r <username>
```

NOTE: **userdel** will never remove any files that are not in the users home directory. To fully remove all of a user's files, you should make sure to remove their mail file from `/var/spool/mail/`, as well as any files they may have in `/tmp/`, `/var/tmp/`, and any other places you have given them permission to write to.

Adding and Modifying Groups

In addition to users, you can also create groups. Groups can be very powerful as they allow you to put an arbitrary set of users in a group, then assign resources that can only be used by users in that group. Typically groups are primarily used for file permissions on UNIX systems, but with *BProc* system, you can also limit the access to a given node to a single group. This section covers creating and modifying groups so that you assign nodes to a certain set of users.

Creating a Group

Before you can add users to a group, you must first create the group. Groups can be created with the **groupadd** tool. This command takes a single argument of the groupname as shown below:

```
groupadd <groupname>
```

That command will create a group named <groupname>.

Adding a User to a Group

Adding a user to a group is a little trickier than just creating the group. In order to add a user to a group you have to use the **usermod**. Unfortunately, **usermod** doesn't allow you to just add a group, but requires you to give it a list of all the groups a user should be in. In order to make sure you don't remove any of the user's groups, you should first use the **groups** command to get a list of all their current groups.

```
[root@cluster /root]# groups smith
smith : smith src
[root@cluster /root]# usermod -G smith,src,<newgroup> smith
```

where `smith` is the name of the user you are adding a group for.

Removing a Group

If you wish to remove a group, you only need to run the **groupdel** command with the groupname as an argument:

```
groupdel <groupname>
```

Controlling Access to Cluster Resources

By default, anyone who can log onto the master node of the cluster, can send a job to any of the compute nodes they wish. This is not always desireable. For this reason, node ownership and mode has been added so that the use of each node can be restricted to a certain user or group.

What Node Ownership Means

Each node has user, group and mode bits assigned to it which indicates who is allowed to run jobs on that node. The user and group can be set to any user ID or group ID on your system. In addition, the use of a node can be unrestricted by setting the user and group to 'root'. For BProc node permissions 'root' and 'any' are equivalent. Node user access follows the normal Linux convention that the most restictive access rule is the one used. Examples: With, user root, group test, and mode 101 (u=1, g=0, o=1), anyone in group test will not be able to access the node. With, user tester, group root, and mode 011 (u=0, g=1, o=1), user tester will not be able to access the node

Note, in Linux "other" is defined as anyone not listed in the user or group.

Checking Node Ownership

You can display the current node ownership of the nodes by either running the **beosetup** program or typing **bpstat** on the command line as shown:

```
[root@cluster]# bpstat
Node(s)                    Status       Mode       User       Group
16-31                       down         ---------- root       root
0-15                        up           ---x--x--x root       root
```

The ’User’ column shows the user for each node and the ’Group’ column shows the group for each node.

Setting Node Ownership

Setting node ownership can be done using the command **bpctl**. You must give **bpctl** a *-S* option to specify which node to change, and either a *-u* option to change the user, *-g* option to change the group, or *-m* to change the mode. The only bit utilized for the mode is the execute bit.

To set the user for node 5 to be ’root’, then you would want to run:

```
bpctl -S 5 -u root
```

To set all the nodes to be in the group ’beousers’, you would do:

```
bpctl -S all -g beousers
```

To allow only group ’beousers’ access to the nodes, you would do both:

```
bpctl -S all -g beousers
bpctl -S all -m 010
```

You can check the *Reference Guide* for more details on **bpctl**.

Using **bpctl** does not permanently change the node ownership. Whenever the master node reboots or **beosetup** causes the BProc daemon to be restarted, the node ownership settings will go back to what they were. To make sure your changes are picked up, you will need to change the setting in the `/etc/beowulf/config` file. The *Reference Guide* details the layout of the file so that you can see exactly what needs to change.

Chapter 6. Job Batching

Beowulf Batch Queue (BBQ)

Overview

The *BBQ* package provides a simple job batch spooler. It is based on the standard **at** package that has been enhanced to handle the Scyld cluster environment. The package supports starting user jobs based on time, and handles load balancing for jobs that are ready to run. The package consists of three programs, the daemon **atd**, a job submission tool **at** with an alias **batch**, and a queue viewer **bbq**. *BBQ* allows users to submit programs for execution when the cluster might normally be under utilized, such as overnight and on weekends; but, the primary goal is to keep the cluster busy. To accomplish this the *BBQ* software looks at individual processor utilization on each node and attempts to keep that level at the default 80 percent mark. Note, in this discussion bbq is used both as a command and as a software package. The convention followed is, *BBQ* for the package, and **bbq** for the command.

Installation and Configuration

BBQ is installed as part of the normal installation procedure, with updates available as standard RPMs. *BBQ* is configured with the **atd** startup script, */etc/rc.d/init.d/atd*. During installation, two directories are created with user and group set to *daemon* with read, write, and execute permission for owner only, */var/spool/at* and */var/spool/at/spool*.

The */var/spool/at* directory will contain the job files. These files are ASCII text shell scripts containing user identification, and the user's environment with commands to be executed. Note that the user must have a home directory. The listing below shows a typical file:

```
!/bin/sh
# atrun uid=1001 gid=1001
# mail  joeuser 0
umask 2
LESSOPEN=\|/usr/bin/lesspipe.sh\ %s; export LESSOPEN
USERNAME=; export USERNAME
HISTSIZE=1000; export HISTSIZE
HOSTNAME=dyn84; export HOSTNAME
LOGNAME=joeuser; export LOGNAME
SSH_TTY=/dev/pts/4; export SSH_TTY
MAIL=/var/spool/mail/joeuser; export MAIL
HOSTTYPE=i386; export HOSTTYPE
PATH=/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/joeuser//bin; export PATH
HOME=/home/joeuser/; export HOME
INPUTRC=/etc/inputrc; export INPUTRC
XAUTHORITY=/tmp/ssh-VSaL2232/cookies; export XAUTHORITY
USER=joeuser; export USER
BASH_ENV=/home/joeuser//.bashrc; export BASH_ENV
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass; export SSH_ASKPASS
LANG=en_US; export LANG
SSH_CLIENT=127.0.0.1\ 1067\ 22; export SSH_CLIENT
OSTYPE=Linux; export OSTYPE
SHLVL=1; export SHLVL
LS_COLORS=no=00: ... truncated ... export LS_COLORS
cd /home/joeuser || {
         echo 'Execution directory inaccessible' >&2
         exit 1
}
#CMDS
NP=32
userProgram -options
[EOF]
```

It is possible for *root* to edit these files, but this should be done with care. If you need to do this, check that your editor of choice does not open temporary files in the spool directory. Job files are named with fixed length strings containing the queue, job number, and time to start. A typical name, *b0002100fd7eb0*, is broken down as queue *b*, job number in hex *00021*, and minutes since epoch *00fd7eb0*. Again it is possible with caution to rename the file and change job parameters. For example, you may change the start time by adding or subtracting minutes from the 32 bit time field. Note, by default, users do not even have read access to these files.

The */var/spool/at/spool* directory contains the user email with the jobs stdout and stderr. This directory should be checked on occasion to insure that dead files are not collecting. These files can occur when jobs are executing and the *atd* daemon has been killed or exited with an error. Dead files may also be possible if the user program has died in an unusual way that *atd* is unaware of. These files are also ASCII text, but it is not recommended that they be edited. The cluster administrator may want to instruct users not to use the email output for large amounts of data. If the user needs a large amount of data, they should open a file.

Each time a job is released for execution *BBQ* will print a message to the syslog. By default these messages are sent to */var/log/cron*. A typical message follows, it contains the start time of job execution (may not be the start time as requested), job number, number of requested processes, number of CPUs allocated to this job, and the total number of CPUs available to the user.

```
Jun  6 11:12:41 dyn84 atd[516]: Released job       46  Requested 3
processes, 2 CPUs ready of 2 possible
```

There is currently no built in command to pause the batch queue. If the administrator needs to do this, the **atd** daemon may be suspended with **killall -STOP atd**, and resumed with **killall -CONT atd**. Running jobs are unaffected by this, but their email may be lost. The daemon may also be stopped and started with **/etc/rc.d/init.d/atd stop** and **/etc/rc.d/init.d/atd restart**.

There are six arguments to the **atd** daemon, **-l**, **-b**, **-d**, **-s**, **-r**, and **-n**.

- **-l** - allows setting of the load factor. This factor is the threshold below which a cpu is considered idle. Before a job is started all of the mapped CPUs must be below this threshold. Note for a Scyld cluster, the loading factor, aka, *loadavg*, is a value from 0% to 100%. Each cpu is treated as a separate computational resource, with the *loadavg* applied to it individually. This is different than the standard **at** *load average*, where SMP machines are set to numbers higher than number of processors - 1.
- **-b** - allows setting the time in seconds for scans of the */var/spool/at* directory. During each scan all files in the directory are checked for the start time. This time is set to 5 seconds by default, but administrators may want to set this higher if system loading on the master is also high.
- **-d** - allows starting the **atd** program as a non-daemon. All error messages will be printed to stdout. In daemon mode stdin and stdout are closed. This mode is only useful for developer debugging.
- **-s** - starts **atd** once and exits. This is retained for **at** heritage, but is not useful in the cluster environment.
- **-r** - restricts user jobs to compute nodes with the *daemon* group id. This is a powerful feature that forces all user jobs to go through using *BBQ* as their cluster interface. This allows for job logging and better load management. To implement this feature, the administrator should start the daemon with the **-r** option and set the group id of all compute nodes to *daemon* with the **bpctl -S a -g daemon** command.

- **-n** - bypass the Scyld enhancements and run **atd** in non-cluster mode. Note that **bbq** may generate errors in this mode, but all regular *at* commands will operate as normal. If you require the non-cluster *at* package, you should remove *BBQ* and install the original *at*.

Job Mapping

Users have some control over the processor mapping by using the *MPI* options and arguments seen below.

```
-all-local   ALL_LOCAL=1
-all-cpus    ALL_CPUS=1
-allcpus     ALL_CPUS=1
-np x        NP=x
-nolocal     NO_LOCAL=1
-map         BEOWULF_JOB_MAP=x1:x2...
-exclude     EXCLUDE=x1:x2...
```

These are treated as requests for cluster resources. Nominally, **bbq** will use one processor for each primary process the user requests, as this maximizes the job throughput. Only when **bbq** is constrained otherwise will it fall back to multiple processes per processor. These constraints are reached when the user selects a limited processor list or the cluster administrator has limited this user’s access to the full cluster. For example, the user sets NP=32, but the user only has permission to run on 16 uniprocessor nodes. **bbq** will map 2 processes per processor. With the same example but with dual processor nodes, **bbq** will map 1 process per processor. The administrator could set up a cluster of 64 nodes as four sub-clusters up to 16 nodes each. This is accomplished by assigning each sub-cluster a different group id and giving that id to different groups of users. The mapping software, **beomap** only assigns users to permissioned nodes. Using this approach would preclude using the **-r** option with **atd**. Administrator may also remove nodes from use by setting a node to *unavailable* with **bpctl -S x -s unavailable**. *BBQ* will only map nodes that are in the *up* state, and that the user has permission to access.

Cluster administrators should encourage users not to specify individual nodes through the BEOWULF_JOB_MAP or -map options as **BBQ** will honor these requests and wait for those exact nodes to be available, ignoring other idle nodes. This defeats some of the load balancing attributes of using the *BBQ* package. Using the hard map options is only useful for non-homogeneous clusters, where either the network fabric or the machines themselves are unequal.

Queue Management

Cluster administrators use the same options as the user for viewing the job queue. The command **bbq**, lists the pending and running batch jobs to stdout. There is also a GUI display **beoqstat** that constantly monitors the queue and updates its display every 5 seconds. This GUI also allows jobs to be sorted and deleted. In addition, whenever **bbq** detects the environment variable *GATEWAY_INTERFACE*, **bbq** will generate HTML to standard out. This environment variable will only be set when **bbq** is called via an APACHE web server. Details on these options and the available queue sorting options are available in the reference sections of this document.

Jobs may be deleted from the queue at anytime via the command line or the **beoqstat** GUI. To do this, run **atrm** with the job number, or right click on the job in **beoqstat** and select delete. Both pending and running jobs may be removed, but it should be noted that **bbq** does not have knowledge of all the individual processes that an application has started. To effectively remove a running job, you should delete the job from the queue, and then kill each of the associated processes.

Chapter 7. Remote Administration and Monitoring

Scyld Beowulf provides a variety of tools for remotely monitoring and administering clusters. These include traditional shell and X window based tools, along with web based tools. Some utilities are available for users to monitor the system, while others are for administrators to configure the cluster.

Command Line Tools

Scyld Beowulf includes **openssh** by default. This can be used to securely login to your server remotely, and use the command line tools. These tools allow modification of all options on your cluster, but do not provide the easy user interface that the graphical configuration tools, such as beosetup, provide Another option for configuring the cluster is to hand edit the configuration files. For more information, on the configuration files and command line utilities, see Chapter 3.

X forwarding

SSH can also be configured to do X forwarding, which allows the GUI applications to be run on a remote machine. This allows you to use the full functionality of the convenient graphics tools, but can be slow, especially if the connection to the cluster is not via a local area network. In order to activate X forwarding, you may need to use the -X option to ssh from your client. Once the X forwarding is setup, you can use any of the GUI tools described throughout this manual.

Chapter 8. Compute Node Boot Options

One of the unique advantages of Scyld Beowulf is the fast and flexible boot procedure for compute nodes. The Scyld BeoBoot system is a combination of unified booting and a carefully designed light-weight compute node environment. The BeoBoot system allows compute nodes to initialize with a very small boot image that may be stored on a wide range of boot media. This small boot image never has to change, yet with Scyld Beowulf's boot setup, you will still be able to change what kernel the compute nodes run, what modules are loaded and every aspect of the application environment, all by just changing a few files on the master node.

This chapter details the boot process for the compute nodes, gives instructions on how to setup different boot media for the compute nodes, and how to change other settings involved with booting the compute nodes.

Boot Process

The process of booting the cluster starts with booting the master node. As part of the master node boot process a set of Beowulf support daemons are started. These daemons are required by the compute nodes to boot properly. Once the Master node has finished booting, the compute nodes can boot. There are two phases to the boot process of the compute nodes.

Phase 1 Boot

This phase boots a simplified kernel configured with the minimum device support needed to boot and communicate on the network. In this boot phase, the PCI bus is scanned for network devices. It will then send out RARP requests on all of the network devices that it finds. **beoserv** on the master should see the requests and respond, giving the compute node an IP address. The compute node will then bring up the network interface for which the IP address was assigned and immediately request a phase 2 boot image from **beoserv** on the master node. After downloading the image, it will then use the two-kernel monte to boot the phase 2 boot image.

The two-kernel monte process is intended so that you can have boot media on all the compute nodes that will always boot the same phase1 image, but allow you to upgrade the kernel on all of the compute nodes by merely changing a few files on the master node.

Phase 2 Boot

In this phase, the kernel that the compute node will actually be running is booted. As in phase 1, the PCI bus will be scanned for network devices and RARP requests will be sent out on all of the network devices found. Once **beoserv** responds with an IP, the network device will be brought up with that IP. At this point, **sendstats** and **bpslave** will be started on the compute node. **sendstats** sends information to **recvstats** on the master node to give information for cluster status monitoring (viewable with the **beostatus** program). **bpslave** is the BProc slave daemon. It connects to **bpmaster** on the master node. These daemons are what allow BProc to move jobs between nodes. Once **bpslave** connects to **bpmaster**, the node's state is changed to 'boot' (node states can be monitored with **beostatus**, **bpstat**, or **beostat** program). Once the compute node is in the 'boot' state, **bpmaster** will run the **/usr/lib/beoboot/bin/node_up** script, with an argument passed to it specifying what the node number is. (NOTE: This script is actually run on the master node and uses **bpsh** and **bpcp** to set things up on the compute node.) The default setting for this script will cause a filesystem to be setup on the compute node, libraries to be copied over for caching, and a few other files that will be needed, and finally a chroot to the filesystem it created.

Adding Steps to the node_up Script

If you wish to add more steps to be executed during the node_up script, you can do it without actually editing the script. Instead, you create a script in the `/etc/beowulf/init.d` directory. All scripts in this directory will be executed for each node that boots up. This script will be sourced by the **node_up** script when the specified node boots, therefore it must be written in standard sh. When your script is sourced, the variable $NODE will be set to the node number that is booting.

Error Logs

During the phase1 boot and first part of the phase 2 boot, any error messages are sent to the console of the compute nodes. However, any output that is generated by the node_up script is sent to a log file in `/var/log/beowulf/`. The actual log file is named node.<nodenumber>, where <nodenumber> is the node number for the node in question. If the compute node ends up in the 'error' state, then there was a problem during the node_up script and you should check the error log in `/var/log/beowulf/` (this log can also be checked by right-clicking on the node in **beosetup** and choosing 'View Log' from the menu). If the node never seems to get to the `boot` state, then you need to check on the console of the compute node and see what is happening there.

Compute Node Boot Media

There are several ways that you can boot a compute node for a Scyld Beowulf cluster. Below are outlined a few of the possible ways. The following methods are all interchangeable and work seamlessly with each other such that if you want you can have some of your compute nodes booting using one method, and other booting with another method.

Hard Disk

This option allows you to boot your compute node off the hard disk, much like a normal Linux box does. An advantage of this is that you have the speed of loading from the hard drive, but this also requires you to have a hard drive, and it cannot be setup until after you already have your nodes up.

In order to set this up you must first partition the hard drive of the compute node to include a beoboot partition (see the chapter on disk partitioning). Once you have the beoboot partition on the compute node, you use **beoboot-install** to create a phase 1 image, install it on the compute node, on the beoboot partition, and setup lilo on the compute node to boot the phase 1 image.

To run **beoboot-install** just on node 0, you run **beoboot-install 0 <device>**. To run it on all of the compute nodes, you do **beoboot-install -a <device>**. In both cases, <device> should be the hard drive that has the beoboot partition on it. (ie `/dev/hda` if it's the IDE primary master, or `/dev/sda` if its the first SCSI disk)

In order for this to work, you must also make sure the BIOS of the compute node is configured so that it will boot from the hard drive.

Floppy Disk

Booting from floppy disks is easy to setup, however it slows down the boot process as it takes a while to load the image off the floppy disk.

There are several ways to create a boot floppy for a compute node. The first is to go into **beosetup**, and click on 'Node Floppy' in the toolbar button. You can also run **beoboot -1 -f -o /dev/fd0** which does the same thing. On the CD, there is an **images/nodeboot.img** file which is a floppy image of a boot floppy. You can write this image to a floppy disk using **dd** or **rawrite** if you like.

After creating the floppy disks and putting them in the compute nodes, you will need to make sure that the BIOS's on the compute nodes are all setup to boot from floppy disk.

CD-ROM

All of the Scyld Beowulf CDs can be used to boot an x86 compute node. The x86 CD will automatically boot a x86 compute node if you let it timeout, or if you type 'beoboot' at the boot prompt.

PXE

PXE is a protocol that defines a standard way to netboot x86-based machines. In order for PXE to work, your compute nodes must have network adapters that support PXE booting as well as BIOS's that support it. The option to PXE boot must also be turned on in the BIOS. This is the preferred method of booting nodes in a Scyld Beowulf cluster.

Linux BIOS

Linux BIOS is a project to replace the BIOS of a machine with Linux. This greatly speeds up the boot process as most of the actual work done by the BIOS is designed to make things like DOS work, but which aren't really needed by Linux.

There has been work done by third parties so that it is a Scyld Beowulf phase 1 image that replaces the BIOS. This has the advantage that all you need for a compute node is a motherboard with ram, processor, built-in network adapter, and a power supply.

Linux BIOS is not supported by Scyld Software, however you can see http://www.linuxbios.org/ for more information if you are interested.

Flash Disk

Although not Beowulf specific, using a flash disk is mentioned as it can increase cluster reliability. A flash disk is a solid state device using an Electrical Eraseable PROM (EEPROM). The devices are seen by the BIOS as an IDE or SCSI harddrive, and support all normal drive operations, including running **beofdisk** and installing the phase 1 boot image. This allows a node cluster configuration with no moving parts other than cooling fans, and is an alternative to using the Linux BIOS. These devices are faster and cheaper than harddrives, and are currently limited to 4 mbytes to 512 mbytes. But, for booting, less than 2 mbytes would be needed.

Creating Boot Images

Every once in a while you might want to recreate your phase 1 or phase 2 boot images. The most likely reason for this is adding new kernel modules or changing some of the network driver settings. For more details on adding kernel modules, please see Chapter 3. This section details how to recreate the phase1 and phase2 boot images.

Phase 1 Image

For the most part, phase1 boot images are created as part of the process of putting them on the bootmedia that is used to boot the compute nodes. See the above section on compute node boot media to see how to recreate these images.

Phase 2 Image

There are two ways to recreate the phase 2 image. The first is to go into **beosetup**, click 'BeoBoot File' on the toolbar. The second is to run **beoboot -2 -n**. These will both create the phase 2 image as `/var/beowulf/boot.img`.

If you wish to create the phase 2 image as a different file, the dialog box in **beosetup** has a field to change that, or you can add the options **-o filename** to the **beoboot** command to specify a different filename. If you change which filename the file is created as, you will need to change the bootfile directive in `/etc/beowulf/config`.

Chapter 9. Disk Partitioning

The disk of each compute node does not need to be partitioned before the node can be successfully added to the cluster as Scyld Beowulf uses ramdisks by default. However, using hard disks can make booting faster and allow for more scratch space for applications. Partitioning allows a disk drive's storage space to be broken up into segments that are then accessible by the operating system. Each *partition* is a segment that can be accessed as if it was a complete disk drive.

Introduction to Disk Partitioning

Disk partitioning on a cluster is essentially no different than on any stand-alone computer with a few exceptions.

On a stand-alone computer or server, the disk drive's file system(s) divide the storage available on the disk into different sections that are configured in ways and sizes to meet your particular needs. Each partition can be accessed independently as if it were a separate disk. The partitions are configured and determined by the partition table contained on each disk.

Each partition table entry contains information about the the locations on the disk where the partition starts and ends, the state of the partition (active or not), and the partition's type. Many partition types exist such as Linux native, AIX, DOS, etc. and are up to the administrator to determine the appropriate types for his/her own system.

Note that your compute nodes do not require hard disks in order to function as nodes on the cluster. In fact, a RAM disk is created on the compute node by default during phase 1 of the boot process. This RAM disk is used to hold the phase 2 boot image downloaded from the master node. If you have diskless nodes, then this chapter does not pertain to you.

Disk Partitioning on Scyld Beowulf

Master Node

On the Master Node of the Scyld Beowulf cluster, the disk partitioning administration is identical to that on any stand-alone Linux server. As part of installing Scyld Beowulf, you are requested to select how you would like to partition the master node's hard disk. After installation, the disk's partitioning can be modified, checked, and utilized via traditional Linux tools such as fdisk, sfdisk, cfdisk, mount etc.

Compute Nodes

The compute nodes of a Scyld Beowulf cluster are slightly different than a traditional, stand-alone Linux server. However, not too many people would enjoy partitioning 64 or more nodes manually.

The compute node hard disks however, still need to be able to be formatted and partitioned to be useful to the applications running on the cluster. In order to do this, Scyld Beowulf provides the **beofdisk** command which allows remote partitioning of the compute node hard disks. It is very similar in operation to fdisk but allows many nodes to be partitioned at once. Beofdisk and Compute node partitioning information is covered in more detail further in this chapter.

In order to boot your compute node from its hard disk, rather than floppy or CD-ROM, a small partition (minimum 2MB) must be set aside for beoboot, the initial boot image. The partition type should be set to type 89. The *partition type* indicates the type of filesystem the partition is expected to contain. This partition should exist at or near the beginning of the disk to avoid problems that some BIOS's can have with large disks (i.e. disks with more than 1024 sectors).

Disk partitioning is very much determined by the cluster system hardware and the requirements of the application(s) that will be running on the cluster. Some applications are very process intensive but not very data intensive. In such instances, the cluster may best utilize a RAM disk in the default partitioning scheme. The speed of the RAM will provide better performance and not having a hard drive will provide some cost savings. Some applications are very data intensive but not very process intensive. In these cases, a hard disk is either required given the size of the data set the application is working with and/or is a very inexpensive solution over purchasing an equivalent amount of memory. The hard drive partitioning scheme is very dependent on the applications needs, what other tools will be interfacing to the data, and what are the preferences of the end-user.

Default Partitioning

This section addresses the default partitioning schemes for a Scyld Beowulf cluster

Master Node

The default Scyld partition table allocates 4 partitions. Most administrators will want to change this to meet their site's needs.

- /boot partition
- /home partition
- / partition
- Swap partition = 2 times physical memory

Compute Node(s)

The default partition table allocates three partitions

- BeoBoot partition = 2MB
- Swap partition = 2 times node's physical memory
- Single root partition equal to remainder of disk

Diskless Operation: The default method of configuration of the compute nodes at boot time is to run off of a ram disk. This "diskless" configuration is appropriate for many applications but not all. Typical usage requires configuration and partitioning of the compute node hard disks which is covered later in this chapter.

Partitioning Scenarios

This section will cover how to implement the three most common partitioning scenarios:

- Applying Scyld recommended default partitioning to all disks in the cluster
- Applying user specified partitioning to all disks in the cluster
- Applying unique partitioning to specified disks in the cluster

beofdisk can read an existing partition table on a compute node, it sequentially queries compute nodes beginning with node 0, and for each new type/position/geometry it finds, it looks for an existing partition table file in `/etc/beowulf/fdisk`. If no partition table is present, a new one is generated that uses the default scheme. When you query, it will create files in `/etc/beowulf/fdisk/`, one for each device/drive geometry it finds. These files can then be modified by hand, or with the default options, then written back to the hard drives. Note: Compute nodes with unpartitioned disks will be tagged with the *error* state when they are booted if the `/etc/beowulf/fstab` refers to those disks.

Default Compute Node Partition Scheme

If you want to apply the recommended disk partitioning to the slave nodes, it only takes a few easy steps. The first step queries all the nodes hard drives and writes out partition table files for them that contain the suggested partitioning:

```
> beofdisk -d
Creating a default partition table for hda:2495:255:63
Creating a default partition table for hda:1222:255:63
```

Next, you need to run the step that will read the partition table files, and partition the hard drives on the compute nodes so that they match:

```
> beofdisk -w
```

In order to use the new partitions you created, you must modify the `/etc/beowulf/fstab` in order to specify how the partitions on the compute node should be mounted. The contents of `/etc/beowulf/fstab` should be in the standard fstab format. Change

```
mkfs never
```

to

```
mkfs always
```

in the `/etc/beowulf/config` file to format the disk(s) on reboot. To try out the new partitioning, reboot the compute nodes with

```
bpctl -S all -s reboot
```

Caution

Change the

```
        mkfs always
```

back to

```
        mkfs never
```

in `/etc/beowulf/config` after the nodes have booted to prevent disks from being reformatted on subsequent reboots.

Manual, but Homogenous Partitioning Scheme

If you wish to specify the partition tables by hand instead of taking the suggested defaults, there are two ways of doing this. The recommended one involves running **fdisk** on the first node (node 0) of the cluster, and then on every *first* node that has a unique type of hard disk. The other method is to manually edit the partition table text file retrieved by the **beofdisk** query.

An example will help. Let’s say the cluster has 6 compute nodes. Also assume all disks of 255 heads and 63 sectors which is most common. Nodes 0, 1, & 5 have a single IDE hard disk with 2500 cylinders. Nodes 2, 3, & 4 have a first IDE disk with 2000 cylinders and node 2 has a SCSI disk with 5000 cylinders.

First we must partition node 0's disk:

```
> bpsh 0 fdisk /dev/hda
```

Follow the steps through the standard fdisk method of partitioning the disk.

Now we need to partition node 2's disk manually with **fdisk**.

```
>  bpsh 2 fdisk /dev/hda
```

Follow the same procedure of using **fdisk**.

Next we do the same with node 4's SCSI disk.

```
> bpsh 4 fdisk /dev/sda
```

Now you query the compute nodes and get all the partition table files for their hard drives written. You do this by running the following command:

```
> beofdisk -q
```

At this point all three partition tables will be translated in to text descriptions and three files would be put in the directory `/etc/beowulf/fdisk`. The file names would be: `hda:2500:255:63`, `hda:2000:255:63`, and `sda:5000:255:63`, They represent the way the compute node hard drives are currently partitioned. If you're brave you can skip the **fdisk** command and just edit these files manually. The danger is that there are lots of rules about what combinations of values are legal so it's easy to make an invalid partition table. Most of the these rules are explained as comments at the top of the file.

Now write out the partitioning scheme using the command **beofdisk -w**.

When specifying unique partitioning to specified nodes, you will also want to specify a unique fstab for each node that has a unique partition table. You do this by creating `/etc/beowulf/fstab.<nodenumber>`. If this file exists, the node_up script will use that as the fstab for the compute node, otherwise it will default to using `/etc/beowulf/fstab`. `/etc/beowulf/fstab.<nodenumber>` should be in the same format as `/etc/beowulf/fstab`.

Change

```
mkfs never
```

to

```
mkfs always
```

in the `/etc/beowulf/config` file to format the disk(s) on reboot. To try out the new partitioning, reboot the compute nodes with

```
bpctl -S all -s reboot
```

Caution

Change the

```
        mkfs always
```

back to

```
        mkfs never
```

in `/etc/beowulf/config` after the nodes have booted to prevent disks from being reformatted on subsequent reboots.

Chapter 10. Filesystems

Filesystems on a Cluster

Filesystems on a cluster consist of two types of filesystems, local filesystems and network filesystems.

Local Filesystems

Local filesystems are the filesystems that exist locally on each machine. In the Scyld Beowulf setup, the master node has a local filesystem, typically ext3, and each node also has a local filesystem. The local filesystems are used for storing data that is local to the machines.

Network/Cluster Filesystems

Network filesystems are used so that files can be shared across the cluster and every node in the cluster can see the exact same set of files. The default network filesystem for Scyld Beowulf is NFS. NFS allows the contents of a directory on the server (in this case the master node) to be accessed by the clients (the slave nodes). The default Scyld Beowulf setup has `/home` exported through NFS so that all the user home directories can be accessed on the slave nodes. NOTE: root's home directory is not in `/home` and thus cannot access its home directory on the slave nodes. This should not be a problem as normal compute jobs should not be run as root.

Scyld Beowulf also includes the software to setup a PVFS filesystem. The PVFS filesystem does not have a central store of files on the master node like NFS does. Instead it spreads its files out across all of the nodes. If all the nodes will be needing to access the same large set of files, this should speed things up as there will no longer be a bottleneck reading from the master node.

NFS

NFS is the standard way to have files stored on one machine, yet be able to access them from other machines on the network as if they were stored locally.

NFS on clusters

NFS in clusters is typically used so that if all the nodes need the same file, or set of files, they can access the file(s) through NFS. This way, if one changes the file, every node sees the change, and there is only one copy of the file that needs to be backed up.

Configuration of NFS

The Network Filesystem (NFS) is what Scyld Beowulf uses to allow users to access their home directories on slave nodes. (The *User's Guide* has a small discussion on good and bad ways to use NFS.) By default, `/home` and nothing else is mounted on the slave nodes. There are two files that control what directories are NFS mounted on the slave nodes. The first is `/etc/exports`. This tells the nfs daemon on the master node what directories it should allow to be mounted and who can mount them. The default `/etc/exports` file looks like this:

```
/home  @cluster(rw)
```

The ’/home’ says that `/home` can be nfs mounted, and the @cluster(rw) says who can mount it. @cluster is a netgroup. It uses one word to represent several machines. In this case, it represents all your slave nodes. ’cluster’ is a special netgroup that is setup by beonss that automatically maps to all of your slave nodes. This makes it easy to specify something can be mounted by your slave noddes. The (rw) part specifies what permissions the slave node has when it mounts `/home`. In this case, all user processes on the slave nodes have read-write access to `/home`. There are more options that can go here, and you can find them detailed in ’man exports’.

The second file is `/etc/beowulf/fstab`. (Note it is possible to setup an individual fstab for every node. For this, we will be assuming that you are using a single fstab for all nodes.) The last line in the default `/etc/beowulf/fstab` is:

```
$MASTER:/home /home  nfs nolock  0 0
```

This is the line that tells the slave nodes to try to mount `/home` when they boot. The $MASTER is a variable that will automatically be expanded to the IP of the master node. The first ’/home’ is the directory location on the master node, and the second ’/home’ is where it should be mounted on the slave node. The ’nfs’ is just specifying that it’s an nfs filesystem. The nolock specifies that locking should be turned off with this nfs mount. We turn off locking so that we don’t have to run daemons on the slave nodes. (If you need locking, take a look at the ROMIO with NFSv3 section for details on how to turn on locking.) The two 0’s on the end are there to make the fstab like the standard fstab in `/etc`.

If you were to want to add an nfs mount of `/foo` to all your slave nodes, then you would first have to add the following line to the end of your `/etc/exports` file:

```
/foo @cluster(rw)
```

You would then have to run ’/usr/sbin/exportfs -a’ as root. For the mount to take place the next time your slave nodes reboot, you have to add the following line to the end of `/etc/beowulf/fstab`:

```
$MASTER:/foo /foo nfs nolock 0 0
```

You can then reboot all your nodes to make the nfs mount happen. If you wish to keep your nodes running, you can issue the following two commands:

```
bpsh -a mkdir -p /foo
bpsh -a mount -t nfs -o nolock master:/foo /foo
```

`/foo` will have to be adjusted for the directory you actually want.

If you wish to stop mounting a certain directory, you can either remove the line from `/etc/beowulf/fstab` or just comment it out by putting a ’#’ at the beginning of the line. At this point, you can leave the line referring to it in `/etc/exports`, or remove it, whichever you feel more comfortable with.

If you wish to unmount that directory on all the slave nodes without rebooting them, you can then run:

```
bpsh -a umount /foo
```

where ’/foo’ is the directory you no longer wish to have nfs mounted.

File locking over NFS

By default, the slave nodes mount `/home` over NFSv3, however locking is turned off. This was done as running the proper daemons to get locking working would require two daemons to be started per slave node, so for each slave node you would lose two pids from your global pid space. This can become quite large loss if you have a large cluster. If you have a program that requires locking, then you will want to do the following steps to turn on NFSv3 locking:

```
mv /usr/lib/beoboot/bin/nfs.daemons /usr/lib/beoboot/bin/nfs
```

Edit the /etc/beowulf/fstab so that the following line:

```
$MASTER:/home   /home           nfs     nolock          0 0
```

looks like:

```
$MASTER:/home   /home           nfs     defaults        0 0
```

Then reboot all your nodes for it to take affect.

PVFS

PVFS provides a high-performance parallel filesystem that allows serial and parallel applications to access data distributed amongst the filesystems of multiple nodes as one large logical partition. The primary goal is to provide such applications a high performance global scratch space.

This section documents the steps required to configure PVFS on a Scyld Beowulf cluster consisting of more than one node. It assumes that the PVFS rpms have already been installed on your system.

It is very important that you use a static collection of I/O nodes. Scyld Beowulf allows you to easily add and remove slave nodes, but you cannot permanently add or remove nodes that serve PVFS data without rebuilding the PVFS file system. However, you can add and remove PVFS file system clients as often as you wish.

Machine Configuration

Prior to configuration, determine the role that each machine, or node will play inside of the PVFS filesystem. The construct can be defined by three roles, Each node can play one or more of these roles.

- metadata server
- I/O server
- client

The metadata server maintains the filesystem, including permissions and time stamps and data locations. There maybe only one metadata server. Clients will talk through this server to manipulate data in the parallel filesystem. An I/O server stores a portion of the PVFS data. An I/O server may hold this data on a local disk or in a ramfs depending on configuration. Clients manipulate data stored in the parallel filesystem. Applications accessing PVFS files run on client machines.

A machine can fill one, two, or three of these roles simultaneously. Each role requires a specific set of binaries and configuration information. There will be one metadata server for the PVFS file system. There can be many I/O servers and clients. In this section we will discuss the components and configuration files needed to fulfill each role.

We will configure our example system so that the "master" node provides metadata service. This node has an internal IP address of 192.168.1.1. There are eight slave nodes, all of which will provide I/O service. The slave nodes are numbered 0 through 7, and possess IP addresses 192.168.1.100 through 192.168.1.107. All nodes will be configured to act as clients. Note that in the following examples we refer to all nodes using the cluster-centric hostnames recognized by Scyld Beowulf. These hostnames follow the pattern .#, where # identifies a particular node in the cluster (.-1 for the master: .0, .1, .2... for the slaves).

Configuring the metadata server

On the machine that will act as your metadata server, you must create a directory that will be used to store file system metadata. You can then run a script called **mkmgrconf** that will configure this directory and provide information about the layout of your system. Take special note of the portion of the script that asks you to list each I/O node on the system.

```
[root@cluster /root]# mkdir /pvfs-meta
[root@cluster /root]# cd /pvfs-meta/
[root@cluster /pvfs-meta]# /usr/bin/mkmgrconf
This script will make the .iodtab and .pvfsdir files
in the metadata directory of a PVFS file system.

Enter the root directory (metadata directory):
/pvfs-meta
Enter the user id of directory:
root
Enter the group id of directory:
root
Enter the mode of the root directory:
777
Enter the hostname that will run the manager:
.-1
Searching for host...success
Enter the port number on the host for manager:
(Port number 3000 is the default)
3000
Enter the I/O nodes: (can use form node1, node2, ... or
nodename{#-#,#,#})
.0, .1, .2, .3, .4, .5, .6, .7
Searching for hosts...success
I/O nodes: .0 .1 .2 .3 .4 .5 .6 .7
Enter the port number for the iods:
(Port number 7000 is the default)
7000
Done!
[root@cluster /pvfs-meta]# ls -al
total 16
drwxr-xr-x    2 root     root         4096 Jul 31 16:01 .
drwxr-xr-x   19 root     root         4096 Jul 31 16:00 ..
-rwxr-xr-x    1 root     root          139 Jul 31 16:01 .iodtab
-rwxr-xr-x    1 root     root           40 Jul 31 16:01 .pvfsdir
[root@cluster /pvfs-meta]#
```

The `.iodtab` and `.pvfsdir` files were created by the **mkmgrconf** script and contain information about the configuration of the file system.

Configuring the I/O server and clients

Next prepare all nodes by building the PVFS data structure, access point an device files. The script `pvfs_slave_create.sh` will create `/mnt/pvfs` as the client access point and reserve `/pvfs-data` to store the filesystem data. If the clients have not already mounted a physical partition as `/pvfs-data`, the directory will be created on the ram disk and will need to be reconfigured after reboot. The same scenario holds if the compute nodes have been booted from read-only media, the physical disks are rebuilt, or additional nodes added to the filesystem. Otherwise setup on physical disks is only necessary once.

To configure physical disks refer to the Chapter 9 section of the administration guide. Make certain that entries to `/etc/beowulf/fstab` mount the desired partition to `/pvfs-data`.

```
[root@cluster /root]# /usr/bin/pvfs_slave_create.sh a
```

The "a" argument specifices that you want to configure all of the currently running nodes as well as the head node. Alternatively, you can just setup specific nodes by passing the node number as an argument, or just setup the local head node by using the "l" argument.

This setup only needs to be performed once if your slave nodes are using the local hard drive and do not reformat on reboot. However, if you are booting from read only media, you must perform this step every time the machine is rebooted.

Starting the PVFS daemons

At this point, everything is configured and all that is left is to start the file system daemons and mount the file system locally on each node. To start up the PVFS daemons on every node you can run the following script (which again should be modified if needed):

```
[root@cluster /root]# /usr/bin/pvfs_servers_start.sh
```

This script assumes that you are running an I/O server on every node, which of course may not have been your desired configuration. Please modify the script if you wish to only start I/O server on particular slave nodes. This script needs to be executed any time the cluster is rebooted.

The server side portion of PVFS is now ready for access. If at any point you wish to verify that one of the iod's or mgr is running correctly on your system you may check it by running the iod-ping or mgr-ping utilites, respectively. These test programs establish communications with the daemons in question and return a status report:

```
[root@cluster /root]# /usr/bin/iod-ping -h .0
.0:7000 is responding.
[root@cluster /root]# /usr/bin/iod-ping -h .1
.1:7000 is responding.
[root@cluster /root]# /usr/bin/mgr-ping -h .-1
.-1:3000 is responding.
```

Client Configuration

There are two primary methods for accessing a PVFS file system. The first is the native PVFS interface which is made available through libminipvfs. This library offers the highest performance method of accessing the file system. It is therefore also the interface used by ROMIO if you configure your system to use MPI-IO. The second method relies on a kernel module to provide standard Linux file system compatability. The interface allows the user to use existing binaries and system utilities on PVFS without recompiling, although at somewhat of a performance penalty over the native library.

First we will cover the kernel module interface, and then cover the steps needed to use the native pvfs interface.

In order to mount the PVFS file system with the kernel interface, you must load the `pvfs.o` module, start the **pvfsd** daemon, and then mount the file system using the **mount.pvfs** command. An example of how to do this can be found in the **pvfs_client_start.sh** script. This script will cause all of the nodes to mount the file system on `/mnt/pvfs`. Notice that you must specify the name of the pvfs-meta directory created earlier back on the manager node.

```
[root@cluster /root]# /usr/bin/pvfs_client_start.sh /pvfs-meta
```

The filesystem is now setup and can be accessed through `/mnt/pvfs` in the same manner as any other file system. For example

```
[root@cluster /root]# dd if=/dev/urandom of=/var/tmp/pvfs-test.img bs=1M count=100
```

This will create a 100 MB file called pvfs-test.img filled with random data. Now copy this file to the access point, `/mnt/pvfs` and then check the files existence remotely.

```
[root@cluster /root]# cp /var/tmp/pvfs-test.img /mnt/pvfs
[root@cluster /root]# bpsh -ap ls -l /mnt/pvfs/ | grep pvfs | sort -n
  0: -rw-r--r--    1 root     0        104857600 Feb 28 12:38 pvfs-test.img
  1: -rw-r--r--    1 root     0        104857600 Feb 28 12:38 pvfs-test.img
  2: -rw-r--r--    1 root     0        104857600 Feb 28 12:38 pvfs-test.img
  3: -rw-r--r--    1 root     0        104857600 Feb 28 12:38 pvfs-test.img
  4: -rw-r--r--    1 root     0        104857600 Feb 28 12:38 pvfs-test.img
  5: -rw-r--r--    1 root     0        104857600 Feb 28 12:38 pvfs-test.img
  6: -rw-r--r--    1 root     0        104857600 Feb 28 12:38 pvfs-test.img
  7: -rw-r--r--    1 root     0        104857600 Feb 28 12:38 pvfs-test.img
```

The following shows example output from the mount command:

```
[root@cluster /root]# mount
/dev/hda5 on / type ext2 (rw)
none on /proc type proc (rw)
/dev/hda1 on /boot type ext2 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
192.168.1.1:/pvfs-meta on /mnt/pvfs type pvfs (rw)
```

When you wish to unmount the file system and remove the modules (this is probably a good idea before rebooting), you may run the following example script:

```
[root@cluster /root]# /usr/bin/pvfs_client_stop.sh
```

Client native library interface

If you wish to use the native PVFS library functions, or if you wish to use the ROMIO MPI-IO implementation, you must perform one further step. Each of the nodes (including slave nodes) must have a file in the `/etc/` directory called `pvfstab`. This file informs the pvfs library where to look for the file system. In our case, for example, it would contain:

```
.-1:/pvfs-meta /mnt/pvfs pvfs port=3000 0  0
```

This file should be copied out to each node that will be participating using either the **bpcp** command or some other means. Remember again to specify the manager node with an IP address rather than hostname.

ROMIO

ROMIO is a high-performance, portable implementation of MPI-IO, the I/O chapter in MPI-2: Extensions to the Message Passing Interface, and is included in the Scyld Beowulf distribution. ROMIO is optimized for noncontiguous access patterns, which are common in parallel applications. It has an optimized implementation of collective I/O, an important optimization in parallel I/O.

What does ROMIO do for me?

ROMIO gives you an abstraction layer on top of high performance input/output. The details for the file system maybe implemented in various ways but ROMIO prevents you from caring. Your binary code will run on a NFS filesystem here and a PVFS filesystem there without changing a line or recompiling. You may think, “But POSIX open(), read(), ... calls already do this for me!”. However, the virtual filesystem code to handle this abstraction is deep in the kernel. You may have to use ROMIO to take advantage of new special and experimental filesystems. PVFS is a good example, while there’s now a kernel module to support VFS access to a PVFS filesystem that was not always the case. It’s easier and more portable to implement a ROMIO module for a new filesystem than a Linux specific VFS kernel layer.

Since ROMIO is an abstraction layer, it has the freedom to be implemented arbitrarily. For example, it could be implemented on top of the POSIX Asynchronous and List I/O calls for real-time performance reasons. The end-user application is shielded from caring and benefits from careful optimization of the I/O details by experts.

Installation and Configuration of ROMIO

ROMIO over NFS

To use ROMIO on NFS, file locking with fcntl must work correctly on the NFS installation. First, since file locking is turned off by default, you need to turn on NFSv3 locking by following the directions in the *Configuration of NFS* section. Now, to get the fcntl locks to work, you must mount the NFS file system with the "noac" option (no attribute caching). This is done by modifying the line for mounting `/home` in `/etc/beowulf/fstab` to look like:

```
$MASTER:/home   /home           nfs     noac            0 0
```

Turning off attribute caching may reduce performance, but it is necessary for correct behavior.

ROMIO over PVFS

Once you have configured and setup PVFS, there is nothing special you need to do in order to use ROMIO over PVFS.

Other Cluster Filesystems

There are various other network filesystems that may be used on a cluster. The list includes:

- AFS (Andrew File System) - Used in many universities, not known for great performance.
- Coda - Explicitly cachable NFS with disconnect. Users may specify which files they want to be permanently cached. Able to run without an active connection to the server. E.g. it works on laptops. Will try to autosync up with the server upon reconnection. Not considered production quality by many who see it as an academic experiment.

- Intermezzo - Shares many qualities of Coda as it’s written by some of the same people. Source code is mostly written in Perl.
- GFS (Global File System) - Written and supported by Sistina Software. Looks to be the best candidate for an advanced network filesystem. The Scyld kernel does include the patches needed to support GFS, but the GFS product must be purchased from Sistina Software.

Chapter 11. Failover

An important concept in computing today is failover. Failover is other nodes compensating for the loss of another node by acquiring its workload. This is commonly used for servers, such as web servers.

What happens when compute nodes fail?

When a compute node fails, all jobs running on that node will fail. If there was an MPI job running that was using that node, the entire job will fail on all the nodes on which the MPI program was running.

Even though the running jobs running on that node failed, jobs running on other nodes that weren't communicating with jobs on the failed node will continue to run without a problem.

If the problem with the node is easily fixed and you want to bring the node back into the cluster, you need merely to plug it back into the cluster and turn it on. It will immediately boot and as soon as it gets to the 'up', new jobs can be spawned that will use it.

If you wish to switch out the node for a new machine, you need merely go into **beosetup**, delete the node that went down, then bring the new node up and move it to the same node number as the node that went down. (See the chapter on **beosetup** for more information on how to do this.) Switching out the node like this will not cause any problems for jobs that are running on other nodes.

Compute Node Data

What happens to data on a slave node after the node goes down depends on how you have the filesystem on the slave node setup. If you are only using a ramdisk on your slave nodes, then all data stored on your slave node will be lost when it goes down.

If you are using the harddisk on your slave nodes, there are a few more variables to take into account. If you have your cluster configured to run mke2fs on every slave node boot, then all data that was stored on ext2 filesystems on the slave nodes will be destroyed. If you do not have it set to run mke2fs on every slave node boot, then it will try to recover the ext2 filesystems with fsck, however there are no guarantees that the filesystem will be recoverable. NOTE: Even if fsck is able to recover the filesystem, there is a chance that files you were writing to may be in a corrupt or unstable state.

How do I Protect my Application from Node Failure?

There is only one good solution for protecting your application from node failure, and that is checkpointing. Checkpointing is where at regular intervals your application writes to disk what it has done so far, and at startup checks the file on disk so that it can start off where it was when it last wrote the file.

The way to checkpoint that gives you the highest chance of recovering is to send the data back to the master node and have it checkpoint there, and also make regular backups of your files on the master node.

When setting up checkpointing, it is important to think carefully about how often you want to checkpoint. Some jobs that don't have much data that needs to be saved can checkpoint as often as every 5 minutes, whereas if you have a large data set, it might be smarter to checkpoint every hour, day, week, or longer. It depends a lot on your application. If you have a lot of data to checkpoint, you don't want to do it often as that will drastically increase your run time. However, you also want to make sure that if you only checkpoint once every two days, that you can live with losing two days worth of work if there is ever a problem.

How do I Prevent Node Failure?

Some people like to run hardware through extensive testing before they use it for a production machine so that they feel confident that the hardware is solid and will not have problems. Unfortunately, this is an option that is not viable for everyone. Hardware failure is a fact of life and something that is for the most part unavoidable unless you get lucky. As such, there is no way to make sure your nodes will never go down, so instead we recommend you concentrate on making sure that, if a node does go down, you can quickly recover.

Chapter 12. Load Balancing and Scyld Beowulf

You have made some rather significant investment in your cluster. It is also evident that it depreciates at a rather frightening rate. Given these two facts it should be obvious you want your cluster busy 100% of the time if possible.

However, timely results of output are also important. If the memory requirements of programs running on the cluster exceed the available physical memory, swap memory (hard disk) will be used severely reducing performance. Even if the memory requirements of many processes still fit within the physical memory, results of any one of the programs may take significantly longer to achieve if many jobs are running on the same nodes simultaneously.

Thus we come to concept of the *Load Balancing*. *Load Balancing* is maintaining that delicate balance between overburdened and idle. Load balancing is when you have multiple servers that can perform the same task and you automatically switch which server performs that task based on which server is currently doing the least amount of work. This helps to spread a heavy work load across several machines and does it intelligently so that if one machine is more heavily loaded than the others, new requests will not be sent to it. By doing this, a job is always run on a machine that has the most resources to devote to it, and therefore gets finished sooner. Generally, it is believed that constant load of one 100% CPU bound process per CPU is ideal. However, not all processes are CPU bound, many are I/O bound on either the hard drive or the network. The act of *Load Balancing* is often described as *Scheduling*.

Optimal *Load Balancing* is almost never achieved, hence, it is a subject of study for many researchers. The optimal algorithm for scheduling the programs running on your cluster is probably not the same as it might be for others. So, you may want to spend time on your own *Load Balancing* scheme.

How does a Scyld Beowulf Load Balance

Scyld Beowulf supplies a general *Load Balancing* scheme via the **beomap** and **bbq** subsystems. The job of scheduling is divided between mapping (beomap) and queuing (bbq). Mapping is the assignment of processes to nodes based upon current CPU load. Queuing is the holding of jobs until the cluster is idle enough to let the jobs run. Both of these are covered in detail in other sections of this guide and the *User's Guide*. In this section, we'll just discuss the scheduling policy that is used.

It is also important to understand that by default the use of the BBQ system is optional. There are ways to change this to try to enforce all users use of the BBQ system (see the mapping chapter). But BBQ was not designed for hostile users and a clever user could find ways to bypass BBQ.

Mapping Policy

The current default mapping policy consists of the following steps:

- run on nodes that are idle
- run on CPUs that are idle
- minimize the load per CPU

Each proceeding step is only performed if the number of desired processes (NP) is not yet satisfied. The information required to perform these steps comes from the **beostat** sub-system.

Queuing Policy

The current default queuing policy is to attempt to determine the desired number of processes (NP) and other mapping parameters from the job script. Next, the **beomap** command is run to determine which nodes would be used if it ran immediately. If every node in the returned map is below 0.8 CPU usage the job is released for execution.

How can I implement my own Schedule Policy?

Right now the queuing portion of the schedule policy is hard-coded into BBQ. Other queuing policies can be achieved by modifying the source code.

The mapping portions, however, are already modularized. There are a number of ways to override the default. They include:

- substitute a different program for the **beomap** command and use **mpirun** to start jobs (which uses beomap).
- create a shared library that defines the function get_beowulf_job_map() and use the environment variable LD_PRELOAD to force the pre-loading of this shared library.
- create the shared library and replace the default `/usr/liblibbeomap.so`.

These methods are in order of complexity. We can't actually highly recommend the first method as your mileage may vary. The second method is the most recommended followed by the third method of replacing the Scyld source code when you're happy that your scheduler is better.

It is highly recommended that you get the source code for the **beomap** package. It will give you a head start on writing your own mappers. For more information on developing your own mapper, see the Programmer's Guide

Load Balancing Example for non-MPI Program

Each time the queue is scanned (this interval is set to 5 seconds by default), **bbq** determines (through beomap) which node is the least busy. It checks that least busy node to see if its CPU utilization is under the specified threshold (80% by default). If it's not, then, at the next batch interval it repeats the process of finding the least busy node and checking its CPU utilization. This continues until it finds an available node to run the pending job on. When **bbq** finds a node whose CPU utilization is under the threshold, the job is released. When bbq releases a job, it starts up a shell with the user's environment that was captured when the job was submitted, and executes the command(s) that was entered. `stdout` and `stderr` are captured in an email and sent to the user when the execution of the shell has completed.

Now, if the submitted job is MPI-aware , the MPI program determines at run-time (via MPI initialization which uses beomap) which node to run on (i.e. the least busy node). But, since **bbq** does not spawn jobs on the compute nodes - it starts them on the master, non MPI-aware programs must be expicitly bpsh'ed out to the nodes. This is where the **mpprun** command comes in handy. **mpprun** uses beomap to determine the current least busy node. In this case, **bbq** can be used to queue up a **mpprun** job to wait until the node (or group of nodes) that meets the specified criteria for the job becomes available. Then **mpprun** decides at runtime which node to run the job on. Here's an example of how one might batch up non MPI-aware jobs:

```
NP=16 mpprun myprog arg1 arg2 >&/dev/null | batch now
```

bbq will hold the mpprun job until 16 processors are under the 80% utilization threshold. Then when the **mpprun** job starts, it determines which 16 nodes to run the `myprog arg1 arg2` job on and then uses **bpsh** to get the jobs out to those nodes. In this example we suppressed the **bbq** mail by redirecting stdout to `/dev/null`.

Chapter 13. Extra Tools

Included in the Scyld Beowulf distribution are extra tools that may be of interest to some users, but they are not part of the default installation. This chapter describes the tools and how to install them.

Linux System Hardware Monitoring

Hardware monitoring on Linux is done through a program called **lm_sensors**. This program provides support for monitoring all types of hardware status, but the most widely known is the CPU temperature.

How do I use lm_sensors?

Getting lm_sensors setup is a relatively simple process. On the Scyld Beowulf CD is an lm_sensors rpm. You need to install this rpm by running ’rpm -Uhv <package>’.

Scyld Beowulf’s lm_sensors package contains special scripts that the **node_up** script knows to look for. When the package is installed, the **node_up** script will automatically run a configure script on the slave nodes when they boot. This script will probe the hardware on the slave nodes and load whatever kernel modules are required by **lm_sensors** to monitor the hardware that is present on the slave nodes.

In order to view the status of the hardware on the slave nodes, you need to use the program **sensors** that comes with **lm_sensors**. You need merely **bpsh** this program over to a slave node and it will output the status of the slave node. Here is a sample command for checking the status on node 1:

```
bpsh 1 sensors
```

What are possible problems with lm_sensors?

There are many different types of hardware out there that are used to give the status of computer hardware. **lm_sensors** makes an attempt to support as much of this hardware as possible, but unfortunately there are times when a piece of hardware will look like something **lm_sensors** knows about, but really isn’t. This can cause some problems, the most common of which is the slave node freezing when the kernel modules are trying to be loaded. If you install the lm_sensors package and your nodes suddenly stop half-way through the boot, you’ve probably run into this problem and you will need to remove the **lm_sensors** package by running this command:

```
rpm -e lm_sensors
```

Chapter 14. Updating software on your cluster

Scyld may, from time to time, release updates and add-ons to Scyld Beowulf in the form of errata. These may be downloaded from the support section of Scyld's website at http://www.scyld.com. Each errata will include instructions for installation, and information on why it was released and what bug(s) it fixes. Please be sure to thoroughly read each announcement, as it may contain specific instructions pertaining to certain requirements and potential conflicts with other software.

What can't I update

Some packages are specifically patched to include support for the Scyld Beowulf system. While other Linux distributions may also have these packages, their packages may not work properly on Scyld Beowulf, causing the cluster to stop working. Most notably, Scyld adds BProc support to the kernel. Without this, the cluster will not boot nor will it run jobs. Additionally, some libraries, such as mpich and pvm, have certain features that must be enabled to properly run on Scyld Beowulf.

Appendix A. Special Directories, Configuration Files, and Scripts

Scyld Beowulf adds some special files and directories on top of the standard Linux install that help control the behavior of the cluster. This appendix contains a summary of those files and directories, and what is in them.

What Resides on the Master Node?

`/etc/beowulf` Directory

All the config files for controlling how BProc and beoboot behave are stored here.

`/etc/beowulf/config`

This file contains the settings that control the **bpmaster** daemon for BProc, and the **beoserv** daemon that is part of beoboot. It also contains part of the configuration for how to make beoboot boot images.

`/etc/beowulf/fdisk/`

This directory is used by **beofdisk** to store files detailing the partitioning of the slave nodes' hard drives, and is also read from when it rewrites the partition tables on the slave nodes. See the Disk Partitioning chapter for more information.

`/etc/beowulf/fstab`

Refer to the disk partitioning chapter for details on using multiple fstabs

`/usr/lib/beoboot` Directory

This directory contains files that are used by beoboot for booting slave nodes.

`/usr/lib/beoboot/bin`

This directory contains the **node_up** script and several smaller scripts that it calls.

`/usr/lib/beoboot/boot`

This directory contains the files needed by **beoboot** to make it so that the phase1 boot image is actually booted.

`/usr/lib/beoboot/pxe`

This directory contains files that are necessary for the master node to serve as a PXE server for the slave nodes.

`/var/beowulf` Directory

This directory contains the booting images for the slave nodes and the list of unknown MAC addresses.

`/var/beowulf/boot.img`

This is the phase2 boot image. It is fed to the booting slave nodes by the **beoserv** daemon. See the booting chapter for more information on creating this file.

`/var/beowulf/nodeboot.img`

This is the phase1 boot image. This copy of the image can be written to a floppy disk using **dd**. By default, this file does not exist, but can be created using the **beoboot** command. See the booting chapter for more information on creating this file.

`/var/beowulf/unknown_addresses`

This file contains a list of MAC addresses that **beoserv** has seen RARP requests from, yet doesn't recognize.

`/var/log/beowulf` Directory

This directory contains the boot logs from slave nodes. These logs are the output of what happens when the **node_up** script runs. The files are named `node.number` where number is actually the node number.

What Gets Put on the Compute Nodes at Boot Time?

- The `/dev` directory contains all the files that were in the `/dev` directory on the master node when the slave node was booted.
- The `/etc` directory exists solely to contain the `ld.so.cache`, `localtime`, `mtab`, and `nsswitch.conf` files.
- The `/home` directory exists as a NFS mount of the `/home` directory from the master node. In this way, all the home directories can be accessed by jobs running on the slave nodes.
- The `/lib` directory contains the shared libraries that were cached over from the master node. By caching shared libraries beforehand, it speeds up the transfer process when you are trying to run jobs. The list of libraries to cache, is controlled by the `libraries` lines in `/etc/beowulf/config`.
- The `/usr/lib` directory may exist if your setup caches any libraries that exist in `/usr/lib` on the master node. Refer to the `libraries` lines in `/etc/beowulf/config`.
- The binary file `/sbin/modprobe` is a statically linked copy of **modprobe**. It exists on a slave node in case the kernel wishes to run it and so it can be run by the lm_sensors configuration script. NOTE: The version of **modprobe** that is shipped with Scyld Beowulf is modified so that if it is run from a slave node, it will use BProc to move itself to the master node and grab the module from there, then move itself back to the slave node and load the module.
- The `/tmp` and `/scratch` directories are both world-writable directories that can be used as temporary space for compute jobs.