Callident Rx User’s Guide

Contributors

Written by Glen Otero, Ph.D.

875 Stevens Ave. #2313
Solana Beach CA 92075

The material in this manual is based on the *User's Guide for NPACI Rocks* version 2.3.2 edition © 2003 by UC Regents. The original material is used with permission of the UC Regents. The content has been extensively revised to describe Callident's Rx Product. The software accompanying this manual, Callident Rx, is based on the NPACI Rocks version 2.3.2 edition © 2003 by UC Regents.

The restrictions accompanying the Copyright 2003 by UC Regents are:

Redistribution in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the San Diego Supercomputer Center and its contributors. 4. Neither the name of the Center nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMTED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABIILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Trademarks

Linux is a registered trademark of Linus Torvalds.

Callident and Callident BioBrew are trademarks of Callident, Inc.

Myrinet is a trademark of Myricom, Inc.

Red Hat and RPM are trademarks or registered trademarks of Red Hat, Inc. in the United States and other countries.

UNIX is a registered trademark of The Open Group.

Intel, Itanium and Pentium are a registered trademarks of Intel Corporation.

AMD is a trademark of Advanced Micro Devices, Inc.

Netscape is a registered trademark of Netscape Communications Corporation in the United States and other countries.

Sun, Sun Grid Engine, and Solaris are trademarks of Sun Microsystems Inc., in the US and in other countries.

SSH and Secure Shell are trademarks of SSH Communications Security, Inc.

The PhpMyAdmin web application (www.phpmyadmin.net) was designed by Tobias Ratschiller in 1998, restarted on SF.net on March 2001 by Olivier Müller, and is released through Sourceforge.net.

The Ganglia scalable distributed monitoring system was developed at University of California at Berkeley by Matt Massie, and is currently developed by an open source partnership between Berkeley, San Diego Supercomputing Center, and others. It is released through Sourceforge.net under a BSD license.

Other trademarks mentioned in this document are the property of their respective owners.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter provides a brief summary of Callident™'s approach to Linux® clusters (more background and terminology can be found in Appendix A), an Rx product feature list and Callident contact information. Rx is a software package that includes Linux and other open source software for running Beowulf clusters.

1.1 Summary of Rx Approach to Linux Clusters

The first cluster computer made from Personal Computer (PC) parts utilizing an open source Operating System (OS) ran at NASA in 1994. Since then, some of the best minds in the computer world have pursued the intuitively appealing goal of building a vastly powerful computer to solve extraordinarily difficult computational problems with:

- Inexpensive, readily available consumer grade PC hardware
- Open source operating system software

The result has been the development of high performance clusters. A computer cluster, or just cluster, is a broad term that means a number of computers networked together. The individual computers are called cluster "nodes". The nodes each run software that pools their resources to work on a common problem or to share a common workload. Specialized clusters were soon developed. The clusters able to handle parallel programming computing tasks were named Beowulf clusters.

As code contributions from talented programmers, expertise and commitment to open source software and the Linux operating system grew, the technical computing community's interest in Beowulf clusters running Linux was sharply focused. The cluster pioneers soon found that their efforts to replace traditional "big iron" or "supercomputers" with Beowulf clusters running Linux were successful. But, despite their multi-million dollar price tags, proprietary hardware and software, special site requirements and expensive maintenance contracts, the big, old "dinosaur" supercomputers turned out to be more difficult to dislodge than initially imagined. The devil for Beowulf clusters turned out to be in the details, node CPU speed increases were lost in slow network transactions between nodes, compilers hadn't been optimized, and worst yet, adding additional nodes caused the aggregate throughput to actually decline. Linux was written for

individual CPUs, not large numbers of nodes, and there was no straightforward method to install, maintain or upgrade clusters with large numbers of nodes. Still, the goal's allure was undeniable. With 149 representatives on the list of the 500 fastest computers in the world, including 17 in the top 50, and 6 in the top 10 (June 2003 ranking), Beowulf clusters are now recognized as the new wave in high performance computing. The list of the top 500 fastest supercomputers in the world can be found at http://www.top500.org, and there is an area dedicated to clusters at http://clusters.top500.org.

By 2003, the original goal of building a supercomputer from PCs running open source software was finally achievable. Better yet, the goal was attainable for folks outside highly technical computing labs. Scientific researchers in communities such as life sciences who have significant computational problems that would benefit from massive computational horsepower can now obtain these resources for a fraction of the price of a supercomputer built only a few years ago. Callident's Rx release supports the following improvements that are now available to researchers building Beowulf clusters running Linux:

- Inexpensive multiple x86 based CPUs running at clock rates greater than 3GHz on a single motherboard
- Inexpensive high performance Gigabit Ethernet interconnects
- Inexpensive hard disk and RAM
- Linux kernel support for symmetric multi-processing

Callident's Rx release has the following features:

- Automated cluster installation
- Cluster management tools
- Myrinet™ high speed, low latency network interconnect support
- MySQL database for storing cluster node information
- Live cluster status monitoring (Ganglia)
- Quality batch processing software (SGE)
- Quality parallel processing software (MPI, PVM)
- RPM software packaging and installation methods

1.2 Conventions

This User's Guide uses the following conventions:

- References to document titles are in *italics*
- Linux commands, names of files, directories, hostnames and RPM paths appear in text in 9.0 pt Helvetica font
- Anything that you type on your keyboard is in `9.0 pt Letter Gothic font`
- User defined variables are displayed in *`9.0 pt Letter Gothic italic font`*. In some instances, user input is shown in brackets
- Anything displayed on the screen is in `Courier`
- Steps to perform tasks are in numbered lists
- A key or combinations of keys that you press on the keyboard are shown in square brackets.

1.3 Contact Information

You can contact Callident by phone, mail or email and get more information at the following Web address.

Contact	Glen Otero, Ph.D.
Phone	619-917-1772
Fax	858-481-2826
Web url	www.callident.com
Email address	support@callident.com
Mail address	875 Stevens Ave., #2313 Solana Beach, CA 92075

Chapter 2

Getting Started

This chapter provides the steps to build your cluster and install its software.

2.1 Media Supplied by Callident

If you received a printed copy of the *Callident Rx User's Guide* with your shipment, you also received the CDs shown in Figure 2-1.

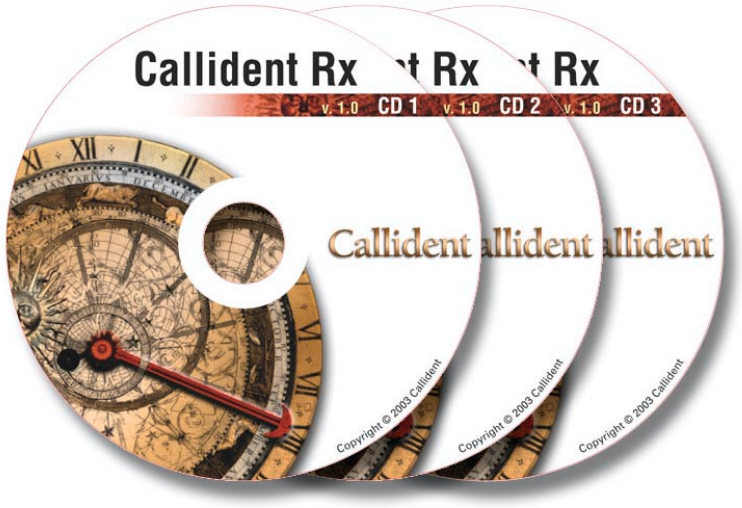

Figure 2-1 Media Supplied by Callident

The media are:

Installation CD #1 Bootable CD. Use this CD to install the Rx Frontend and compute nodes.

Installation CD #2 Rx distribution software for the Rx Frontend.

Installation CD #3 Rx distribution software for the Rx Frontend.

2.2 Assembling the Cluster

The following sections provide information about the hardware that the Rx cluster software supports, basic site preparation information that you will need to evaluate any proposed site for the cluster, and cluster assembly information.

2.2.1 Supported Hardware

Since the Rx cluster software is built on top of Red Hat Linux releases, Rx supports all of the hardware components that Red Hat supports, but only supports the x86 and IA-64 architectures.

The Rx cluster software supports the following microprocessors:

- x86 (Intel® Pentium® and AMD™ microprocessors)
- IA-64 (Itanium™ family of microprocessors)

The Rx cluster software supports the following network cards:

- Ethernet (All flavors that Red Hat supports, including Intel Gigabit Ethernet)
- Myrinet (Lanai 9.x or higher)

Regardless of your Beowulf hardware resources and the performance goals you have in mind, Rx software supports many types of hardware. If you have x86 based CPUs and hardware that is supported by Red Hat Linux 7.3, you will be able to build a Beowulf with Rx software. Check to see if your cluster hardware will work with Red Hat Linux by doing a little research on Red Hat's hardware compatibility list that can be found online at http://hardware.redhat.com/hcl/. You can increase the odds of Red Hat Linux support for your hardware by purchasing name brand, mature computer products. Since many of the latest PC components, like video cards and peripheral controllers, often lag behind in Linux support, check carefully for device driver support before purchasing any new hardware.

2.2.2 Assembling the Nodes

The first step is planning and assembling the Linux cluster. The best strategy for building a Linux cluster has been extensively researched by the Linux community. The book entitled *Beowulf Cluster Computing With Linux* by Thomas Sterling and MIT Press provides the best information currently available on this subject.

Callident favors rack-mounted equipment because of its reliability and density, even though racks are more expensive: they are more reliable than mini-towers. There are Rx clusters, however, that are built from mini-towers. Choose what makes sense for your site and your Rx cluster's intended use.

2.2.3 Site Preparation

Carefully consider the following check list before choosing a site for your Rx cluster. Clusters aren't easy to move, and you will want to make sure that any prospective site has enough power, cooling capability and space.

- ❑ Route to intended site has sufficient clearance and access. If you must move large boxes or racks, make sure that you can get them into elevators or over piles of cables without damaging equipment or causing personal injury. If you are adding computers into racks, make sure that the racks are far enough apart so that you can slide individual computers all the way out for repair on their rails. For 19" racks, you need a minimum of 24" and up to 36" of clearance.
- ❑ Conditions at site are satisfactory. The site has no unusual EMI, ESD or environmental (excess dust or vibration for example) factors. If you have densely loaded racks, and the site is on an upper floor, check the floor loading limits carefully before beginning the installation.
- ❑ Plan for current and future power requirements. See the following section.
- ❑ Plan for current and future cooling requirements. See the following section.
- ❑ Gather all applicable monitor, serial, Ethernet cables, routers, switches, monitors, cards, PCs and AC power cords. Label all cables.

2.2.3.1 Basic Information About Calculating Power and Cooling Requirements

The following section provides basic information about calculating power and cooling requirements for your cluster. It is intended to provide you with basic information so that you can evaluate a site effectively, and you can have an informed discussion with electrical contractors and site administrators (if applicable).

A wall electrical socket in North America can continuously deliver about 15 amps of current at 120 volts (maximum). A typical PC used in an Rx cluster contains from one to four CPUs. PCs vary greatly in the amount of power they use per CPU and in the efficiency rating of their power supplies. However, typical PC power usage is about 300 watts per PC.

How many PCs in a cluster can an individual socket support? Watts are calculated by multiplying amps times volts. A wall socket can supply about 1800 watts (120 volts times 15 amps). A typical

wall socket could support about six x86 PCs (not Itanium CPUs) in a rack. Since the average cluster has a least 32 processors, you will need to coordinate installing additional power circuits for the site.

Power consumption generates heat. Sites remove heat by suppling air conditioners, and air conditioners are rated by their ability to remove tons of air conditioning load. Follow these steps to determine the total air conditioning load:

1. Identify all the wattages of each item associated with the cluster. Include monitors, routers, and any additional equipment.
2. Multiply the total wattage times 0.000285 to calculate the tons of air conditioning load. One ton of air conditioning removes 12,000 Btu/hour. For example, a new cluster with 64 PCs in a rackmount chassis with 128 x86 based CPUs (with each PC consuming about 300 Watts per hour) uses about 19200 watts of power each hour. By multiplying 19200 times 0.000285, we calculate that this new cluster adds 5.4 tons of additional air conditioning load to the site.

Finally, plan with a large margin of additional capability for future expansion, CPU upgrades and unexpected problems. For example, an Itanium based cluster would need far more cooling capability than x86 based cluster. As an example of unexpected problems, sites sometimes find that cooling hot PCs encased in metal boxes with small fans can lead to stagnant pools of hot air that end up being re-circulated. This can lead to heat building up to levels that exceed the technical specifications for motherboards. So, make sure that the site has sufficient AC capacity to remove more heat if this problem occurs.

2.3 Installing Your Frontend and Compute Nodes

The Rx cluster contains the following node types:

- Frontend
- Compute

The Rx cluster architecture dictates that the cluster be networked as illustrated in Figure 2-2. Figure 2-2 illustrates that the master-slave hierarchical schema of the Rx cluster architecture sets it apart from a Network of Workstations (NOW) or other generic LAN-based topologies. As this configuration illustrates, the Frontend is viewable to an outside LAN. Anyone with access to the LAN can remotely access the Frontend provided they have an account. The compute nodes, on the other hand are protected behind the Frontend on a private, dedicated network typically comprised of non-routable IP addresses (`10.x.x.x`, `192.168.x.x`) that are invisible to the public LAN.

This architecture prevents users from accessing compute nodes without first logging into the Frontend, and this design is implemented for Rx cluster security, control, and cluster administration. The master-slave hierarchy allows security and control measures to be focused on a single point of access: the Frontend. Authorization, authentication, and access policies are set and enforced at the Frontend.

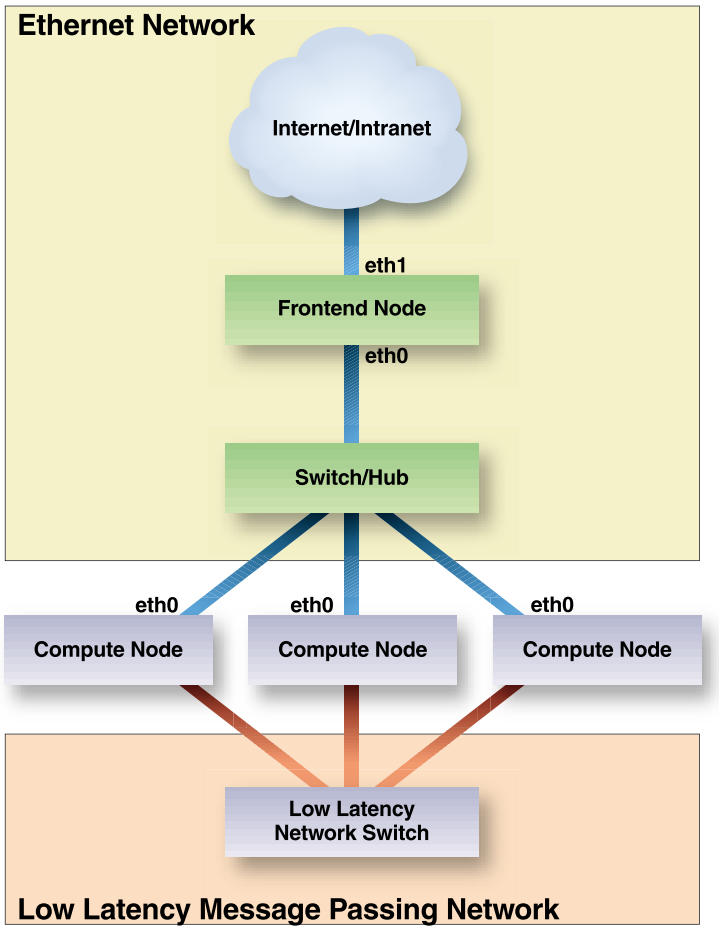

Figure 2-2 Node Connections in Rx Cluster Architecture

2.3.1 Frontend Node

Frontend nodes are connected directly to the outside world. This is typically a local LAN or the Internet. Many services (for example NFS, NIS, DHCP, NTP, MySQL, and HTTP) run on the Frontend node or nodes. Cluster users log into the Frontend node, submit jobs, and compile code. Your Rx cluster needs a competent system administrator to manage these network services for the cluster's users. Frontend nodes have the following characteristics:

- Two Ethernet interfaces: one public, one private
- Minimum of 6 GB of hard drive storage space available

Hint: The Frontend node must provide many services for the cluster, so it's a good idea to use the PC with the best performance for the Frontend. It's not uncommon for the Frontend to have two CPUs, ample memory and lots of disk space.

The Frontend directs the actions of the cluster, and all the compute nodes are subservient to the Frontend. The following cluster activities and tasks are initiated and managed on the Frontend:

- User login
- Software development
- Compiling code
- Launching and managing jobs
- File serving
- Monitoring node health
- System administration

All of the cluster monitoring command software (described later in this chapter) require that the server components be installed on the Frontend and the client components installed on the compute nodes.

2.3.2 Compute Node

These are the workhorse nodes. They are expendable. Our cluster management scheme allows the complete OS to be reinstalled on every compute node in a short amount of time (about 10 minutes). These nodes are not visible to the public LAN or Internet.

Compute nodes have the following characteristics:

- Power cable
- Ethernet connection for administration
- Disk drive for storing base operating system environment (OS and libraries)
- Optional high-performance network (Myrinet for example)
- PC motherboard BIOS capability to boot without a keyboard

2.3.2.1 Private Ethernet Network

All compute nodes are connected via Ethernet on the private network. This network is used for administration, monitoring, basic file sharing and running jobs.

On the compute nodes, the Ethernet interface that Linux maps to `eth0` must be connected to the cluster's Ethernet switch. This network is considered private, that is, all traffic on this network is physically separated from the external public network (shown as the Internet/intranet as shown in Figure 2-2).

On the Frontend, two Ethernet interfaces are required. The interface that Linux maps to `eth0` must be connected to the same Ethernet network as the compute nodes. The interface that Linux maps to `eth1` must be connected to the external network (for example, the Internet or your organization's intranet).

2.3.2.2 Application Message Passing Network

All nodes can be connected with an optional low-latency, high-bandwidth network and the required switches that enable high-performance message passing for parallel programs.

Note: Once the cluster has been assembled, each node must be set to boot without a keyboard. This procedure requires setting BIOS values and, unfortunately, the procedure is different for every motherboard. Worse yet, there are some motherboards whose BIOS won't allow the OS to boot without a keyboard being attached. Callident recommends avoiding the use of older PCs that won't boot without a keyboard because this isn't an issue for newer model PCs.

2.4 Configure the Frontend and Compute Nodes

In this section, you will have to input several screens of information that the Frontend needs for cluster installation and management.

2.4.1 Frontend Configuration

After making sure that the BIOS boot order on the Frontend has been configured to boot from the CD-ROM first, and the hard drive second, follow these steps:

1. Insert Rx Installation CD Disk 1 into your Frontend node.
2. Reset the Frontend.
3. When you see the boot prompt, type:

```
frontend
```

Hint: The `boot` prompt arrives and departs from the screen quickly. It is easy to miss it if you are not careful. If you do miss it, the node will assume it is a compute node, and the Frontend installation fails. You can use Linux's virtual console capability to switch between the installation prompts, system log, system messages and a shell prompt to troubleshoot problems during installation and use. Press the combination of keystrokes shown in Table 2-1 to switch consoles as necessary.

Table 2-1 Accessing Virtual Consoles

Console Display	Console Number	Keystroke Combination to Access
Installation dialog	1	[Crtl]-[Alt]-[F1]
Shell prompt	2	[Crtl]-[Alt]-[F2]
Installation log	3	[Crtl]-[Alt]-[F3]
System messages	4	[Crtl]-[Alt]-[F4]
Other messages	5	[Crtl]-[Alt]-[F5]
X Windows	7	[Crtl]-[Alt]-[F7]

After typing `frontend`, the Red Hat installer called `anaconda` starts running. You will see a screen similar to the following shown in Figure 2-3:

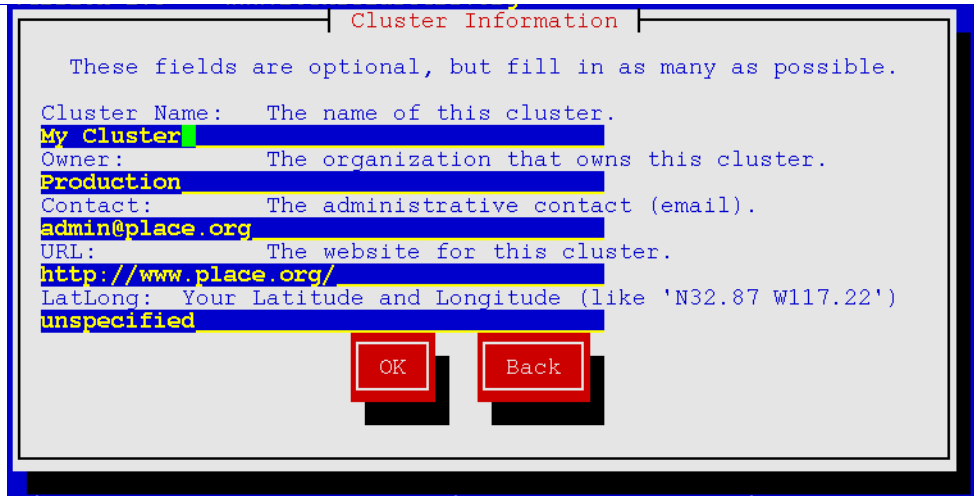

Figure 2-3 Cluster Information Screen

4. Enter the information appropriate to your cluster. This information is used by Ganglia to uniquely identify this cluster. Entries are optional.

Next, the disk partitioning screen shown in Figure 2-4 allows the user to select automatic or manual partitioning for the Frontend.

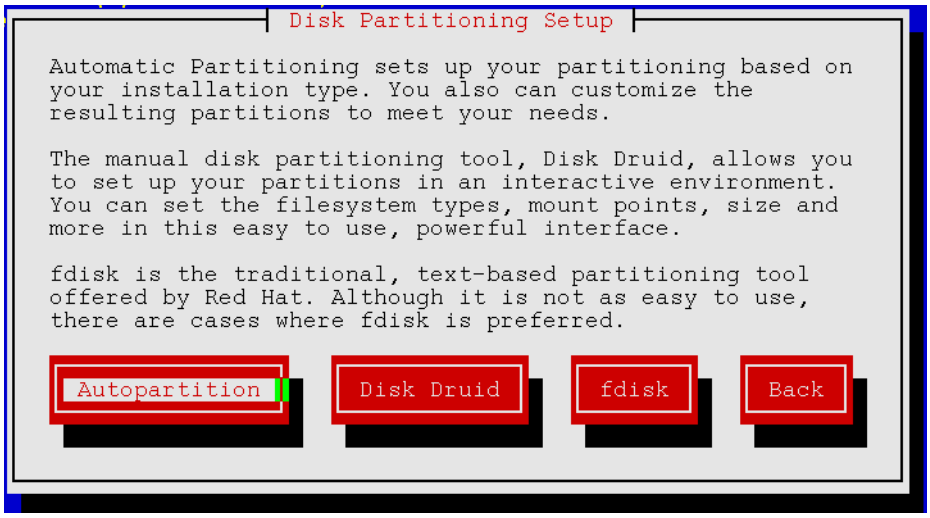

Figure 2-4 Disk Partitioning Screen

Automatic partitioning is the default (and recommended). If you plan to manually partition your Frontend, select either Disk Druid or fdisk.

Note: Callident recommends that you install the default values throughout this section of the installation. To do that, press [Tab] until the OK button is highlighted, then press [Enter] for each of the following steps.

5. Select automatic partitioning, by pressing the Autopartition button. This will partition the Frontend as shown in Table 2-2.

Table 2-2 shows the names and sizes of the partitions on the hard disk.

Table 2-2 Frontend Default Root Partitions

Partition Name	Size
/	4 GB
swap	1 GB
/export	remainder of disk

The private cluster network configuration screen shown in Figure 2-5 allows you to set up the Ethernet network that connects the Frontend to the compute nodes.

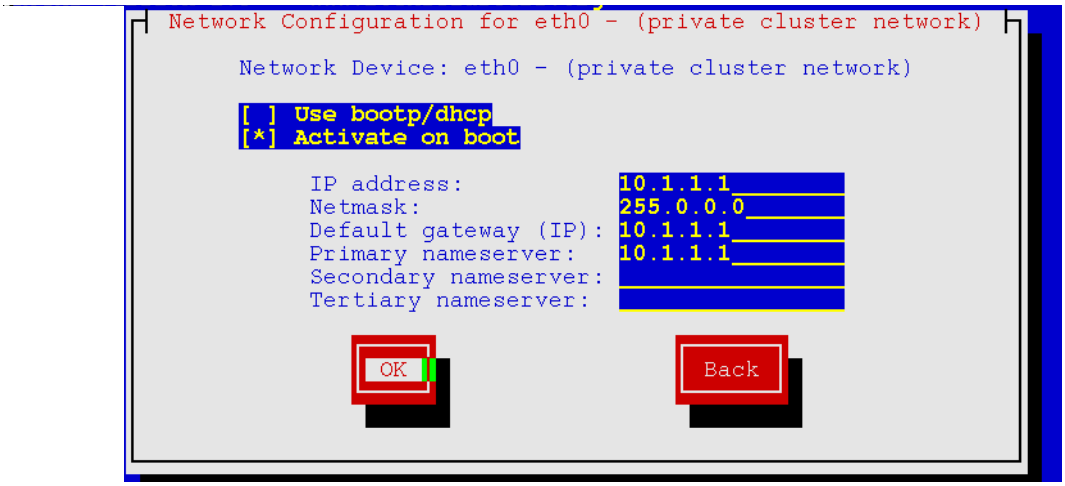

Figure 2-5 Network Configuration for eth0

Note: It is recommended that you accept the defaults.

6. To install the default values, press [Tab] key until the OK button is highlighted, then press [Enter].

The public cluster network configuration screen shown in Figure 2-6 allows you to set up the networking parameters for the Ethernet network that connects the Frontend to the outside network (the Internet, for example).

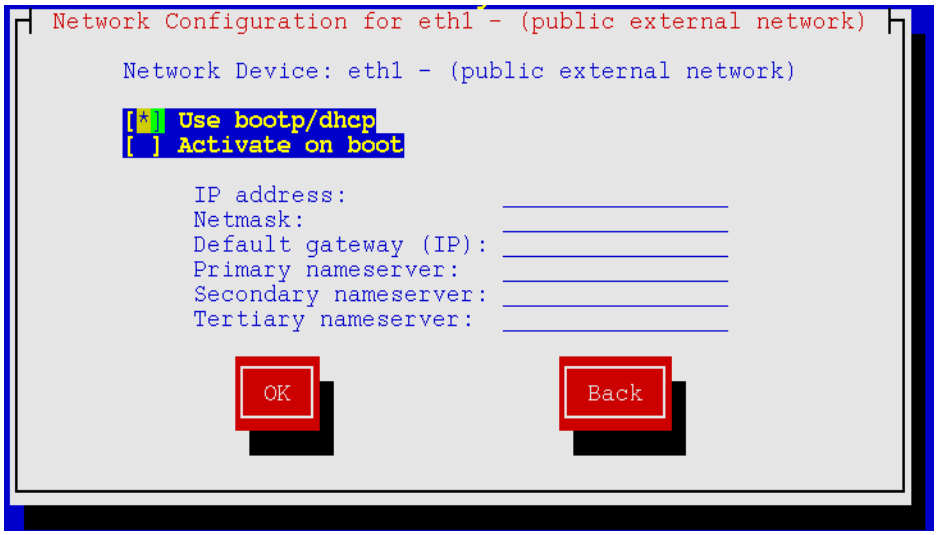

Figure 2-6 Network Configuration for eth1

If you want to specify a static IP address, unselect the `Use bootp/dhcp` option using the [spacebar] key, check the `Activate on boot` option, then fill out the remaining fields. You may need to contact the local network administrator to get some of these values.

Here (Figure 2-7) is an example of how we configured the external network on a Frontend node:

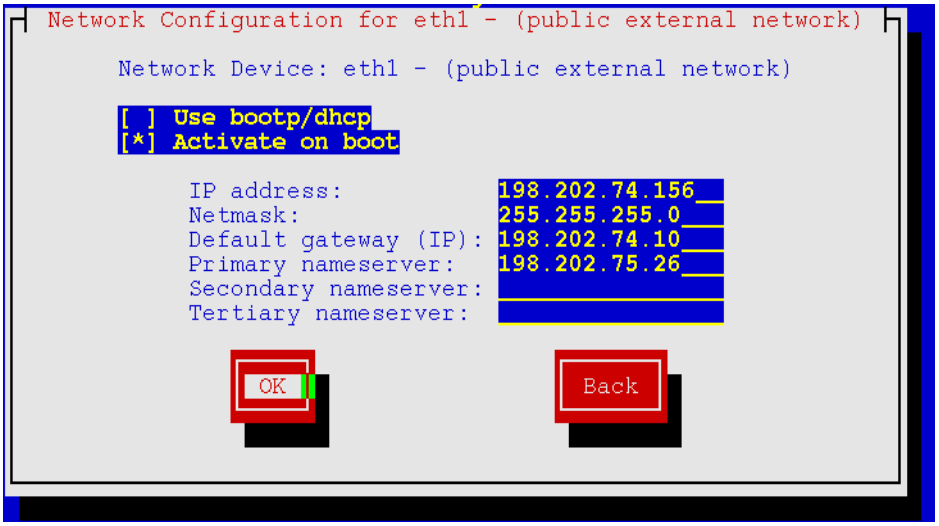

Figure 2-7 Example of External Network Configuration

7. Next, the Frontend's name is chosen. As shown Figure 2-8, the default name is `frontend-0`. Choose the name for your Frontend:

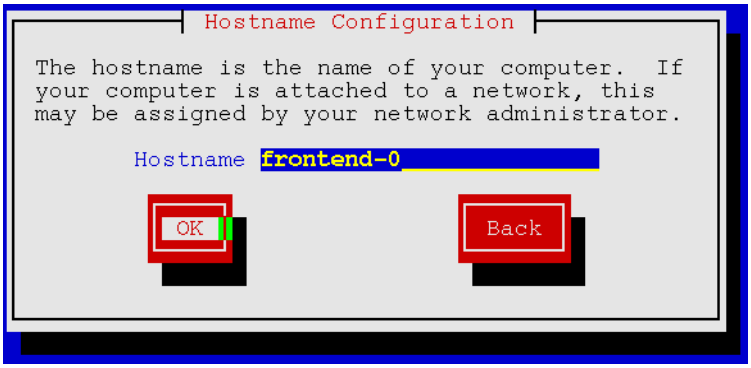

Figure 2-8 Hostname Configuration

8. Input the root password as shown in Figure 2-9:

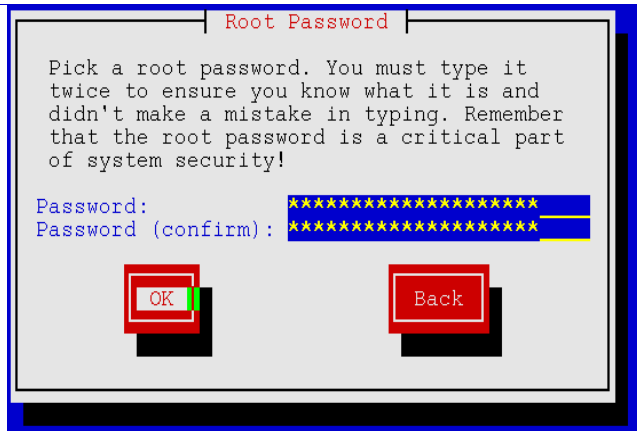

Figure 2-9 Entering Root Password

Note: The authentication screen allows a user authentication scheme choice. It is recommended that you accept the defaults rather than LDAP and Kerberos which have not been tested.

9. To install the default values (recommended), press the [Tab] until the OK button is highlighted, then press the [Enter].

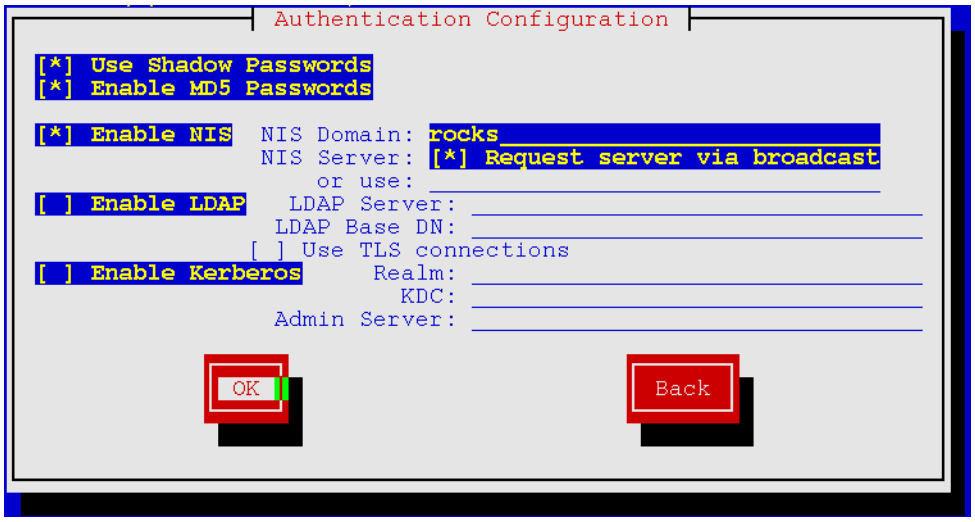

Figure 2-10 Authentication Configuration

2.4.2 Installation Status Screens

The following screens show what you should see throughout the remainder of a typical installation. First, the Frontend formats its file system (Figure 2-11):

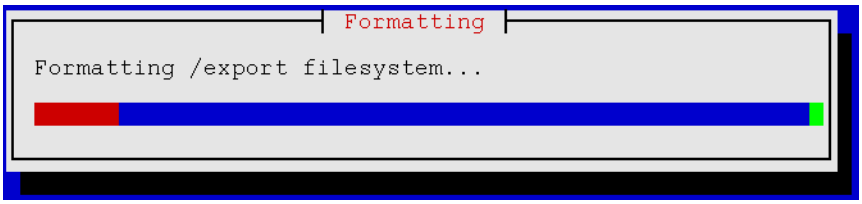

Figure 2-11 Formatting File System

Next, it installs packages (Figure 2-12):

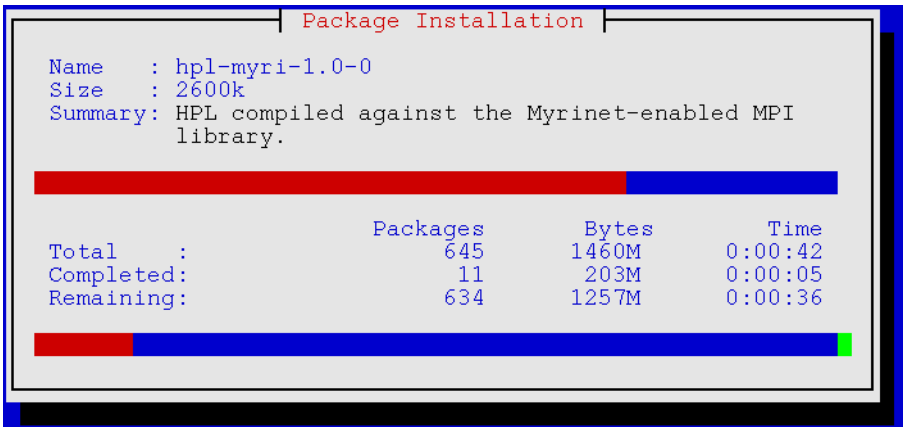

Figure 2-12 Installing Packages

The next screen (Figure 2-13) shown indicates that services are being configured.

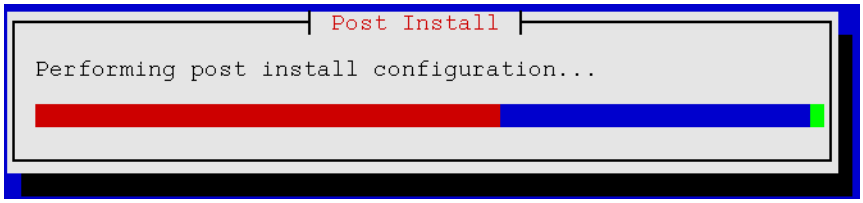

Figure 2-13 Configuring Services

As shown in Figure 2-14 below, the Frontend copies the Rx cluster distribution from the CD to the hard disk. The distribution on the hard disk is used to install the compute nodes in the final step.

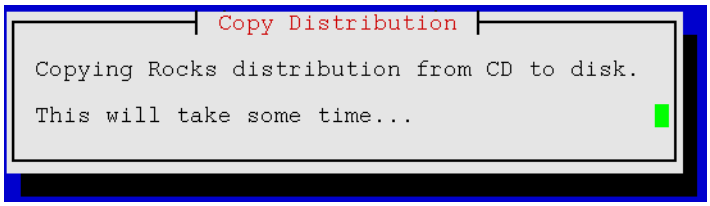

Figure 2-14 Copying Rx Distribution

When the installation completes, the CD is ejected and the Frontend reboots.

Note: In order to start the X Window System on the Frontend, you will need to create the appropriate `XFree86` configuration file for the video card. Use the `Xconfigurator` program to do this as described in the Red Hat documentation. If you do not know anything about the Frontend's video card select "4MB" of video RAM and "16 bit color 800x600" when using `Xconfigurator`. This video mode should work on any modern VGA card. If you choose not to boot into the X Window environment by default, type `startx` at the command line to launch `XFree86` after rebooting.

10. Remove Rx Installation CD Disk 1.
11. Login as root.
12. Press [Enter] three times to generate `ssh` keys without a passphrase.

The following steps are required to complete the Rx distribution on the Frontend before installing the compute nodes:

13. Insert the Rx Installation CD Disk 2 into the Frontend.
14. As root, change to the install directory:

    ```
    # cd /home/install
    ```

15. Mount the CD-ROM:

    ```
    # mount /dev/cdrom /mnt/cdrom
    ```

16. Copy the files from Rx Installation CD Disk 2. This takes some time.

    ```
    # rocks-dist copycd
    ```

17. Unmount Rx Installation CD Disk 2 using the umount command.

    ```
    # umount /mnt/cdrom
    ```

18. Repeat steps 15 through 17 for the Rx Installation CD Disk 3.
19. From /home/install, execute the following command to build all of the Rx software on the Frontend:

    ```
    # rocks-dist dist
    ```

The Frontend installation is now complete.

2.4.3 Compute Nodes

After making sure that the BIOS boot order on each of the compute nodes has been configured to boot from the CD-ROM first, hard drive second, and PXE third, follow the steps in this section to install the compute nodes.

1. Login to the Frontend node as root.
2. Run the following program. It captures the compute node DHCP requests and puts their information into the Rx MySQL database. Input:

    ```
    # insert-ethers
    ```

This presents a screen similar to Figure 2-15:

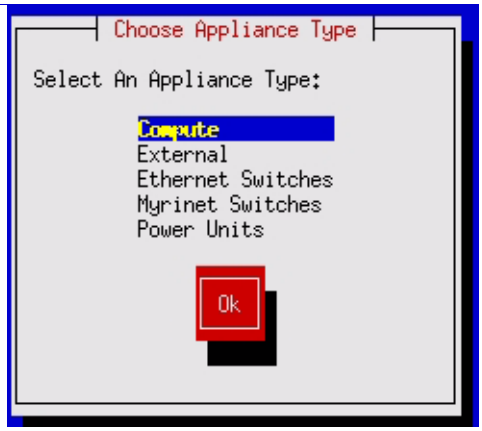

Figure 2-15 Choosing Cluster Node Type

Note: Before powering on any compute nodes, check to see if your managed Ethernet switch issues DHCP requests by default. If it does, you'll want to select "Ethernet Switches" from the list above. Many managed Ethernet switches issue DHCP requests by default so that they can receive an IP address.

When insert-ethers captures the DHCP request for the managed switch, it will configure it as an Ethernet switch and store that information in the MySQL database on the Frontend.

Hint: You may have to wait several minutes before the Ethernet switch broadcasts its DHCP request. If after 10 minutes (or if insert-ethers has correctly detected and configured the Ethernet switch), then you should quit insert-ethers by pressing [F1] .

3. Restart insert-ethers and continue.
4. Select the default selection, Compute, and press [Enter].

Figure 2-16 Listening for DHCP Requests

Figure 2-16 indicates that insert-ethers is waiting for new compute nodes.

5. Take the CD Disk 1 (the same one you used to install the Frontend node) and put it in your first compute node.

Hint: If you don't have a CD drive in your compute nodes, you can use PXE or a boot floppy.

6. Power up the first compute node.

When the Frontend receives the DHCP request from the compute node, you will see a display similar to Figure 2-17.

Figure 2-17 Compute Node Successfully Identified on Frontend

The appearance of this display indicates that insert-ethers received the DHCP request from the compute node, inserted it into the database and updated all configuration files such as /etc/hosts and /etc/dhcpd.conf. You will see this display for each compute node that is successfully identified by insert-ethers.

At this point, you can monitor the compute node installation by using telnet on a separate virtual console on the Frontend. Extract the name of the installing compute node from the insert-ethers output. The first number displayed shows the number of the cabinet (rack) and the second number shows the compute node's number within that rack.

Note: The initial use of the insert-ethers command begins counting the compute nodes in the first rack as cabinet *0* as the default setting, the second use must specify the number of the next rack beginning with *1*.

For example, if the compute node name is compute-*0*-*0*, this indicates the first compute node housed in the first rack. To monitor the installation of this compute node, type the following at the Frontend:

```
# telnet compute-0-0 8000
```

When the Rx installation completes on each compute node, CD #1 is ejected.

7. Take the CD out of the tray, and place it into the next compute node and press the power button.

After you've installed all the compute nodes in the initial cabinet or rack, you can install the compute nodes in additional racks by specifying the rack's number as `--cabinet=x`. If you aren't installing any additional racks, quit insert-ethers by pressing [F1].

To continue installing the compute nodes in a second cabinet, quit insert-ethers by pressing [F1] on the Frontend. Restart it with the following command:

```
# insert-ethers --cabinet=1
```

8. Take the CD out of the last compute node in the first rack, and put it into the first compute node in the second rack, and press the power button.
9. When all compute nodes have been successfully installed using insert-ethers, end the installation by pressing [F1].

2.4.3.1 Default Compute Node Disk Partitioning

The default root partition is 4 GB. The default swap partition is 1 GB. The remainder of the root disk is configured as the partition /state/partition1.

Table 2-3 and Table 2-4 show the names and sizes of the partitions on a compute node's single or multiple hard disks.

Table 2-3 Compute Node: Default Root Disk Partitions on Single Hard Drive

Partition Name	Size
/	4 GB
swap	1 GB
/state/partition1	remainder of root disk

All remaining disk drives have one partition with the name /state/partition2, /state/partition3, and so on. Table 2-4 shows an example of the device names, mount points and partition sizes for a file system that uses three SCSI hard drives.

Hint: After the initial installation, all data in the file systems labeled /state/partition*X* will be will be preserved through reinstallations.

Table 2-4 Compute Node: Default Root Disk Partitions on Three SCSI Hard Drives

Device Name	Mount Point	Size
/dev/sda1	/	4 GB
/dev/sda2	swap	1 GB
/dev/sda3	/state/partition1	remainder of root disk
/dev/sdb1	/state/partition2	size of disk
/dev/sdc1	/state/partition3	size of disk

2.4.3.2 Modifying Compute Node Disk Partitioning

On the Frontend, create a new XML configuration file that will replace the current auto-partition.xml configuration file:

```
# cd /home/install/profiles/site-nodes/
# cp skeleton.xml replace-auto-partition.xml
```

Inside replace-auto-partition.xml, add the following section:

```
<main>
<part> / --size 4096 --ondisk hda </part>
<part> swap --size 1000 --ondisk hda </part>
<part> /mydata --size 1 --grow --ondisk hda </part>
</main>
```

This sets up a 4 GB root partition, a 1 GB swap partition, and the remainder of the drive is partitioned as /mydata. Additional drives on your compute nodes can be set up in a similar manner by changing the --ondisk parameter.

In the example above, the syntax follows directly from Red Hat's kickstart program (aside from the <part> and </part> tags). For more information on the part and the clearpart keywords, see *Red Hat Linux 7.3: The Official Red Hat Linux Customization Guide* in the *Rx Supplemental Documentation* binder or at http://hardware.redhat.com/doc/manuals/linux/. User-specified partition mount point names (*/mydata* for example) cannot be longer than 15 characters.

Warning: If the user-specified partitioning scheme is not currently configured on an installing compute node, all the partitions on the compute node are removed and the user-specified partitioning scheme will be forced onto the node. If you change the partitioning scheme, all partitions are removed and reformatted. This is because we have been unable to make Red Hat's clearpart --drives= **work as advertised.**

If the desired user-specified partitioning scheme is currently configured on an installing compute node, only the root partition is reformatted, and all other partitions remain intact.

To change the size of an existing partition, you must rename the mount point for the partition. This is because the matching logic writes keys off mount point names only.

2.4.3.3 Removing a Compute Node From the Cluster

This process describes how to replace a node that has already been inserted into the cluster with insert-ethers. This procedure is somewhat non-intuitive, but is effective.

1. On your Frontend, execute:

```
# insert-ethers --replace= your compute node's name
```

For example, if the compute node's name is compute-0-1, you would execute:

```
# insert-ethers --replace= compute-0-1
```

This presents a screen similar to the following shown in Figure 2-18:

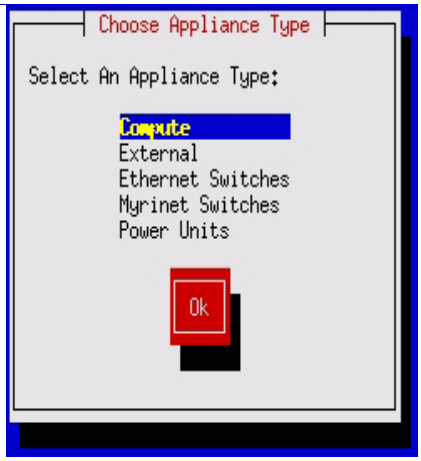

Figure 2-18 Appliance Type Selection Screen

2. Press the [Enter] key. This changes the display to:

Figure 2-19 Compute Node Selected

3. Now press the [F1] key to exit `insert-ethers`.
4. The appropriate configuration files need to be rebuilt (/etc/hosts and /etc/dhcpd.conf for example) and their respective services need to be restarted. This is accomplished by executing:

```
# insert-ethers --update
```

The compute node has now been removed from the cluster, and the compute node entry has been removed from the mySQL database on the Frontend.

In order to insert a node with the same name as the node you just removed:

1. Turn off the compute node.
2. Run the insert-ethers program on the Frontend with the desired compute node name as input:

```
# insert-ethers --cabinet=x --rank=x
```

3. Boot the compute node with your Rx CD 1 (or pxe boot).

For example, if the compute node's name is compute-0-1, you would execute:

```
# insert-ethers --cabinet=0 --rank=1
```

This will re-add the compute node as compute-0-1.

2.4.3.4 Troubleshooting

The following sections provide solutions for problems that might occur during installation.

2.4.3.4.1 Error opening kickstart file /tmp/ks.cfg

When I plug the monitor into a compute node that I'm attempting to boot with an Rx Installation CD, it displays the error message `Error opening kickstart file /tmp/ks.cfg. No such file or directory`. What went wrong?

A compute node kickstart requires the following services to be running on the Frontend node:

- dhcpd
- httpd
- mysqld
- autofs

Check whether or not httpd and mysqld are running with the following commands:

```
# ps auwx | grep httpd
# ps auwx | grep mysqld
```

If either one is not running, restart them with:

```
# /etc/rc.d/init.d/httpd restart
```

and/or:

```
# /etc/rc.d/init.d/mysqld restart
```

The autofs service is called automount. Check to see if it is running:

```
# ps auwx | grep automount
```

If it isn't, restart it by entering:

```
# /etc/rc.d/init.d/autofs restart
```

Test to see whether or not the Rx installation infrastructure is working:

```
# cd /home/install
# ./kickstart.cgi --client=compute-0-0
```

This should return a kickstart file.

Finally, check for any errors associated with kickstart.cgi:

```
# ./kickstart.cgi --client=compute-0-0 > /dev/null
```

2.4.3.4.2 Error Can’t mount /tmp

When I attempt to install a compute node, the error message on the compute node says, "`Can't mount /tmp. Please press OK to restart`." What should I do?

This situation generally arises when the disk drive on the compute node doesn’t have enough storage space, or its `/tmp` partition is too small. The Rx installation procedure formats the disk on the compute node if Rx has never been installed on the compute node before. To avoid this situation, we recommend that the compute node disk drives have a minimum of 6 GB of storage space for the operating system and the default partitioning scheme.

If you have a smaller disk drive, the applicable fix requires changing the way Rx partitions disk drives. See “Modifying Compute Node Disk Partitioning” on page 24 for details.

Hint: There is another possibility for this error. Check to make sure that eth0 and eth1 are properly configured and networked on the Frontend and compute nodes. The compute nodes need to communicate over eth0 to the Frontend for installation.

2.5 Basic Cluster Commands

Before you begin computing, the following section describes some basic cluster management commands you can use to verify that your cluster is functioning correctly. Although the cluster-fork command can't specify the hosts directly on the command line, it uses Secure Shell™ (ssh™) to login to each compute node and run the specified command. This is useful to copy and remove files, and shutdown the cluster.

The basic command is cluster-fork *command*. For example, you can begin by checking processes running on the Rx cluster nodes with the following command:

```
% cluster-fork ps
```

You can copy a file from the Frontend to each node with the following input:

```
% cluster-fork scp user@frontend:/path/to/file /path/to/file/on/compute/node
```

For example, if user jack wants to copy /tmp/stuff on the Frontend, which has the hostname master, to /tmp on every node, on master he would type:

```
% cluster-fork scp jack@master:/tmp/stuff /tmp
```

You can remove a file with the following input:

```
% cluster-fork rm file
```

To install software on the cluster (after you've put the rpm into the distribution with rocks-dist), for example, input:

```
% cluster-fork rpm -Uvh /home/install/rocks-dist/7.3/en/os/i386/RedHat/RPMS/<rpm name>
```

To shut down the compute nodes from the Frontend, input:

```
% cluster-fork shutdown -h now
```

Cluster-kill stops processes owned by a particular user (if run by root or the user), however, one user can't cluster-kill another user's processes.

```
% cluster-kill user
```

Finally, use two commands to reinstall the compute node specified on the command line:

```
% ssh-agent $SHELL
% shoot-node compute-x-y
```

2.6 Running Jobs on the Cluster

2.6.1 Sun Grid Engine (SGE)

The Grid Engine project (http://gridengine.sunsource.net/) is an open source community effort sponsored by Sun™ Microsystems to facilitate the adoption of distributed computing solutions. SGE is the default workload management software for the Rx cluster. For additional information on workload management, see Section A.2.3.1, "Workload Management." The following sections outline the basics of SGE usage. Additional SGE documentation can be found in the *Rx Supplemental Documentation* binder and at the Grid Engine website at http://gridengine.sunsource.net/project/gridengine/documentation.html.

SUN Grid Engine software is a batch system in which jobs (formulated as shell scripts) are put into queues and executed when the resource requirements of the job are fulfilled. Jobs are sorted in FIFO (first–in–first–out) fashion according to their priority. The job priority can only be lowered by an ordinary user. Jobs not eligible for execution will be placed in the pending job pool. The jobs are also sorted by equal-share-scheduling which means that within each priority level jobs are sorted among different users. This prevents a user from "pushing" other users downwards by submitting a series of jobs (from a shell script).

When a job has ended, the console output of the script will be put into files in the user's home directory. The names of the files are composed of the job script file name, an appended dot sign followed by an `o` for stout file and an `e` for the stderr file and finally the unique job ID. These files can be merged and placed in other locations by supplying the right flags, described below. So if a user submits the job `simple.sh` the system will answer: `your job 231 [simple.sh] has been submitted`. When the job has been executed, the output will be called, `simple.sh.o231` and `simple.sh.e231`.

2.6.1.1 Using Grid Engine

The SUN Grid Engine (SGE) is a batch system. Users submit jobs that are placed in queues, and the jobs are then executed. The job execution order depends on the following factors:

- System load
- Opening hour of the queues
- Job priority

2.6.1.1.1 Usage

To use SGE on a Callident Rx cluster, you will need to create a file called /etc/USESGE to turn it on:

```
# touch /etc/USESGE
```

Then, re-source /etc/profile.d/gridengine or re-login to load the SGE environment. SGE should work properly after that.

2.6.1.1.2 Command Summary

The main command line SGE commands are qsub, qstat, qhost, qdel, and qconf.

The qsub command submits jobs to the queuing system. It has the following options:

-cwd	Run the job from the current working directory (Default is $HOME)
-v	Pass the variable VAR (-V passes all variables)
-o	Redirect standard output (Default is $HOME)
-e	Redirect standard error (Default is $HOME)

This example uses some of the options listed above:

```
qsub -cwd -v SOME_VAR -o /dev/null -e /dev/null myjob.sh
```

Started with no arguments qsub accepts input from STDIN (press the [Ctrl] key and the [D] key simultaneously to send input).

Typically, qsub is used for traditional batch job submissions. A batch job is a shell script that can be executed without user intervention and does not require access to a terminal. All qsub options given in a script file require the special comment symbols #$. The following code sample shows a qsub batch script example with its options explained.

```
#!/bin/sh
#
# submit.sh- a qsub batch script example with options explained
#
# qsub options:
#
# name of the job
```

```
# -N
#
# define job output file
# -o
#
# define job error file
# -e
#
# change to the current working directory upon starting of the job
# -cwd
#
# notify me about pending SIG_STOP and SIG_KILL
# -notify
#
# join the error and standard output streams into one file
# -j y
#
# my e-mail address
# -M username@callident.com
#
# don't flood myself with e-mail
# -m n
####################
# All qsub options need to use the leading #$.
# Here's the actual business end of the script:
#
#$ -N myjob
#$ -cwd
#$ -o myjob.out
#$ -e myjob.err
#
# Launch the job
my_job
########
```

To submit this job script to SGE, type the following:

```
# qsub submit.sh
```

2.6.1.1.3 Parallel Jobs

SGE provides support for parallel programs in the form of parallel queue environments. A queue can be defined as a parallel queue containing a number of slots. Users can allocate slots until the slots are all consumed.

This next script template can be used with the MPI parallel environment (queue).

```
#!/bin/sh
#
# submit_parallel.sh-a qsub batch script example for submitting a
# parallel job to SGE      # with options explained
#
# qsub options:
#
# use the parallel environment mpi with n processors
# -pe mpi n
#
# name of the job
# -N
#
# define job output file
# -o
#
# define job error file
# -e
#
# change to the current working directory upon starting of the job
# -cwd
#
# notify me about pending SIG_STOP and SIG_KILL
# -notify
#
# join the error and standard output streams into one file
# -j y
#
# my e-mail address
#$ -M username@callident.com
#
# don't flood myself with e-mail
#$ -m n
#####################################
# All qsub options need to use the leading #$ in a qsub script.
# Here's the actual business end of the script:
#
```

```
#$ -N myjob
#$ -cwd
#$ -o my_parallel_job.out
#$ -e my_parallel_job.err
#$ -pe mpi 4
#
# Launch the job

mpirun -np 4 my_parallel_job
####################
Usage :qsub submit_parallel.sh
```

To submit this job script to SGE, type the following:

```
# qsub submit_parallel.sh
```

Note: qsub only accepts shell scripts, not executable files. Although one could write a small wrapper script around binaries to submit them, there are two convenient techniques to submit binaries as jobs very simply without involving a separate script.

1. Type the qsub command, along with any desired flags and options, then press [Enter] without specifying a job script. You will then see a secondary shell prompt. At this prompt, you can type in the name of the binary. You can then press [Enter] and continue to enter more binary or shell commands. When you are done specifying your job, press the [Ctrl] + [D] key.

   ```
   % qsub -l arch=solaris64
    sleep 60
    <ctrl-D>
    your job 47427 ("STDIN") has been submitted
   ```

2. Type the qsub command, along with any desired flags and options, then use the STDIN redirect construction EOF. Type in one or more lines containing any combination of binaries and shell commands at the secondary prompt as above. Then, on a line by itself, type EOF and press [Enter].

   ```
   % qsub -N test << EOF
   ? sleep 60
   ? EOF
   your job 47428 ("test") has been submitted
   ```

Both of the techniques described in the steps above take advantage of the fact that qsub uses the STDIN stream as a job script (if you don't specify a script file as an argument).

The qstat command shows the current status of queues, and jobs associated with queues.

When the qstat command is started without arguments, it shows the jobs that are currently running or jobs that are pending. It has the following options:

-f	Show full listing of all queues
-ne	suppresses empty queues in conjunction with -f
-j	Shows detailed information on pending/running job
-U	Shows current jobs by user

The qhost command shows the job or execution host status. When qhost is started without arguments it shows a table of all execution hosts and provides information about their configuration. It has the following options:

-l	attr=val Show only certain hosts
-j	Shows detailed information on pending/running job
-q	Shows detailed information on queues at each host

The qdel command deletes a job from the queuing system. Pending jobs are dequeued, and running jobs are killed. The user must supply a job_id given at submission or by qstat.

The qconf is used to increase/decrease the number of slots for each queue.

The slots correspond to the number of processors for each node, but you can change it. To change it for a queue (compute-0.0.q for example), you would type the following command:

```
# qconf -mq compute-0-0.q
```

This opens up a queue configuration session with your current $EDITOR. Look for the attribute called slots and modify that to the desired number of slots in each queue.

The following section describes the use of MPI without SGE on the Rx cluster.

2.6.2 Using mpirun

Mpirun on a Rx cluster launches parallel jobs that are linked with the Ethernet device for MPICH.

Note: You must run HPL as a regular user (that is, not root).

If you don't have a non-root user account on the cluster create one for yourself with:

```
# useradd username
```

Create a password for the account with:

```
# passwd username
```

For example, if you want to interactively launch the benchmark "High-Performance Linpack" (HPL) on two processors, follow these steps:

1. Create a file in your home directory named machines and put two entries in it, such as:

   ```
   $ compute-0-0
   $ compute-0-1
   ```

2. Download the two-processor HPL configuration file and save it as `HPL.dat` in your home directory.

3. Now launch the job from the Frontend node:

   ```
   $ /opt/mpich/ethernet/gcc/bin/mpirun -nolocal -np 2 -machinefile machines
   /opt/hpl-eth/bin/gcc/xhpl
   ```

2.6.3 Using mpirun.ch_gm

Mpirun.ch_gm on Rx clusters is used to launch parallel jobs that are linked with the Myrinet device for MPICH.

Note: You must run HPL as a regular user (that is, not root).

If you don't have a user account on the cluster, create one for yourself with:

```
# useradd username
```

Create a password for the account with:

```
# passwd username
```

For example, follow these steps to interactively launch the benchmark "High-Performance Linpack" (HPL) on two processors:

1. Create a file in your home directory named `machines`, and put two entries in it, such as:

   ```
   compute-0-0
   compute-0-1
   ```

2. Download the two-processor HPL configuration file and save it as `HPL.dat` in your home directory.

3. Now launch the job from the Frontend:

   ```
   $ /opt/mpich/myrinet/gcc/bin/mpirun.ch_gm -np 2 -machinefile machines
   /opt/hpl-myri/bin/gcc/xhpl
   ```

2.7 Running Linpack

2.7.1 Interactive Mode

This section describes ways to scale up a HPL job on a Rx cluster.

To get started, you can follow the instructions on how to run a two-processor HPL job in the previous sections. Next, add more entries to your `machines` file to scale up the number of processors. For example, follow these steps to run a 4-processor job over compute nodes `compute-`*`0-0`* and `compute-`*`0-1`*.

1. Add the following entries to your `machines` file:

   ```
   compute-0-0
   compute-0-0
   compute-0-1
   compute-0-1
   ```

2. Adjust the number of processors in `HPL.dat` by editing the following entries:

```
1 Ps
2 Qs
```

to:

```
2 Ps
2 Qs
```

Note: The number of total processors HPL uses is computed by multiplying *P* times *Q*. That is, for a 16-processor job, you could specify:

```
4 Ps
4 Qs
```

3. Adjust the `np` argument on the `mpirun` command line:

```
$ /opt/mpich/ethernet/gcc/bin/mpirun -nolocal -np 4 -machinefile \
   machines /opt/hpl-eth/bin/gcc/xhpl
```

If you want to make the job run longer, you need to increase the problem size. This is done by increasing the *Ns* parameter. For example, if you want to quadruple the amount of work each node performs change the following values:

```
1000 Ns
```

to:

```
2000 Ns
```

Hint: Keep in mind that doubling the `Ns` parameter **quadruples** the amount of work. For more information on the parameters in `HPL.dat`, see http://www.netlib.org/benchmark/hpl/tuning.html.

Chapter 3

Cluster Monitoring

This chapter provides information about how to monitor your Rx cluster by accessing the Rx Table of Contents page, Rx mySQL database, and Ganglia monitoring environment.

3.1 Monitoring Your Cluster

An Rx cluster makes monitoring of its activities and configuration available through a set of web pages. The Frontend node of the cluster serves these pages using an Apache webserver. This section describes the web-based monitoring tools immediately available on all Rx clusters. The Frontend must have the X Window System configured to use these web-based monitoring tools. If you do not have X Windows installed on the Frontend, refer to the "Configure the Frontend and Compute Nodes" section of Chapter 2 or Red Hat's *The Official Red Hat Customization Guide* that you can find in the *Rx Supplemental Documentation* binder or at http://www.redhat.com/docs/manuals/linux for installation information. Follow these steps:

1. Ensure that the X Window System is configured on the Frontend.
2. Open the Netscape® browser.
3. Enter the following address to connect to the Rx cluster web server:

   ```
   http://localhost
   ```

3.1.1 Table of Contents Page

If you can successfully connect to the cluster's web server, you will be greeted with the Rx Table of Contents page as shown in Figure 3-1. This simple page has links to the monitoring services available for this cluster. With them, you can edit the mySQL database, view the state of the cluster's resources, and examine the software repository.

Note: The PBS Queue Monitoring selection shown in Figure 3-1 is not supported.

Figure 3-1 Connection to Table of Contents Web Page

3.1.2 Accessing Cluster Website Using SSH Tunneling

For security purposes, web access is restricted to the internal cluster network by default. However, since usually only Frontend and compute nodes (which have no monitors) reside on this network, some extra effort is required to view the monitoring web pages on computers outside the internal cluster network.

The first method of viewing webpages involves sending a web browser screen over a secure, encrypted SSH channel. To do this, follow these steps:

1. Log into the cluster's Frontend.

   ```
   $ ssh frontend-0
   ```

2. Ensure you have an X Server running on your local machine. Start Netscape on the cluster with the following command. The `ssh` process will setup an encrypted channel for the Netscape window to operate through.

   ```
   $ netscape &
   ```

3. Wait until the Netscape window appears on your local machine. Open the URL http://localhost/ and browse as normal.

3.1.3 Enabling Public Web Access with Control Lists

To permanently enable selected web access to the cluster from other machines on the public network, follow the steps below. Apache's access control directives will provide protection for the most sensitive parts of the cluster web site, however some effort will be necessary to make effective use of them.

Warning: HTTP (web access protocol) is a clear-text channel into your cluster. In some sense it is inherently insecure, even though the Apache webserver is mature and well tested. Opening this port by following the instructions below will make your cluster more prone to malicious attacks and breakins.

1. Edit the /etc/sysconfig/iptables file. Uncomment the line as indicated in the file.

```
...
-A INPUT -i eth1 -p tcp -m tcp --dport ssh -j ACCEPT
# Uncomment the line below to activate web access to the cluster.
#-A INPUT -i eth1 -p tcp -m tcp --dport www -j ACCEPT
... other firewall directives ...
```

2. Restart the iptables service. You must execute this command as the root user.

```
# service iptables restart
```

3. Test your changes by pointing a web browser to http://my.cluster.org/, where my.cluster.org is the DNS name of your Frontend.

If you cannot connect to this address, the problem is most likely in your network connectivity between your web browser and the cluster. Check that you can ping the Frontend node from the machine running the web browser, that you can ssh into it, etc.

3.2 Cluster Database

This web application allows you to view and edit the active Rx mySQL database shown in Figure 3-2. Rx uses this database to store data about its configuration, and information about the nodes in this cluster. See Figure 5-1, Relational Schema Diagram on page 56, for a description of this database's structure and semantics.

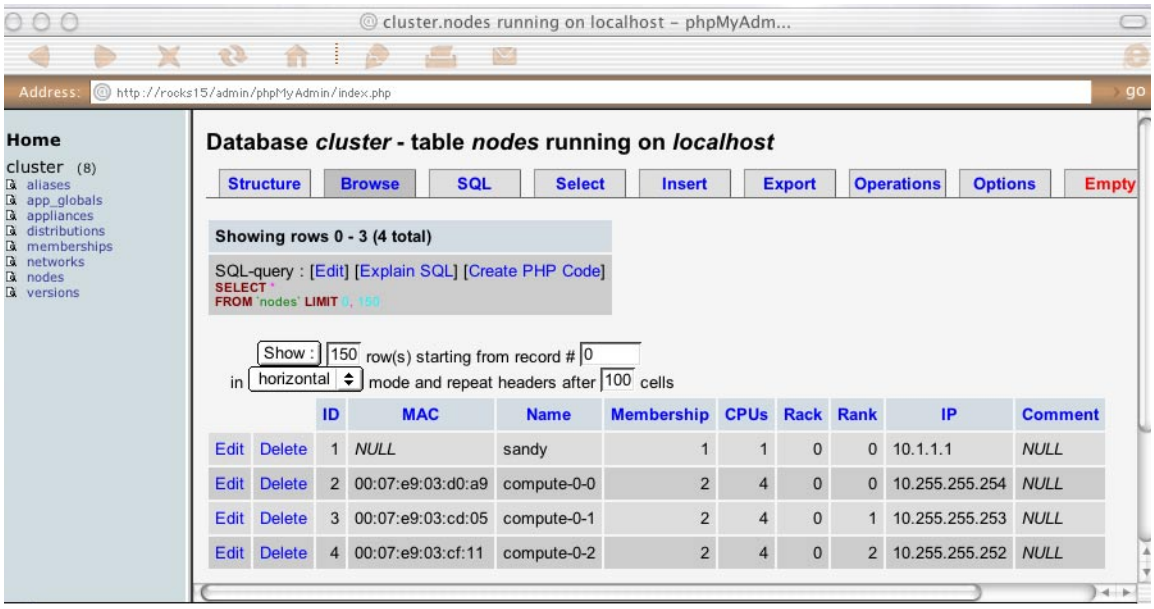

Figure 3-2 Cluster Database

The web database application will allow Queries, Inserts, Updates, and Deletes to the active database. Any changes made via the web application will be immediately visible to any services that consult the database. Because of this design, access to this page is restricted to only hosts on the internal network. To enable extended access to the database web application, edit the /etc/httpd/conf/rocks.conf file as follows.

```
<Directory "/var/www/html/admin/phpMyAdmin">
        Options FollowSymLinks Indexes ExecCGI
        AllowOverride None
        Order deny,allow
        Allow from 127.0.0.1
        Deny from all
</Directory>
```

Add additional Allow directives in this section to specify which additional hosts will be given access to the web database application.

3.3 Cluster Status (Ganglia)

The Ganglia package is an open source toolkit that provides scalable, distributed monitoring for clusters. Ganglia is documented at www.sourceforge.net/projects/ganglia/ and in the *Rx Supplemental Documentation* binder.

Ganglia's monitoring environment consists of the following daemons and tools:

gmetad	Meta daemon that stores states on the Frontend
gmond	Monitoring daemon that runs on each node
gstat	Connects to monitoring daemon to provide node status with a command line interface tool
gmetrics	Command line interface tool that defines the metrics that monitoring daemons track

As shown in Figure 3-3, Ganglia's cluster monitoring webpage provides a cluster status summary and a graphical interface to collect live cluster information reported by the Ganglia monitors running on each node. In the left column on the Cluster Report, Ganglia provides a summary of the cluster's status on the Frontend. The monitors gather values for various metrics such as CPU load, free memory, disk usage, network I/O, and operating system version. These metrics are sent through the private cluster network and are used by the Frontend node to generate the historical graphs shown.

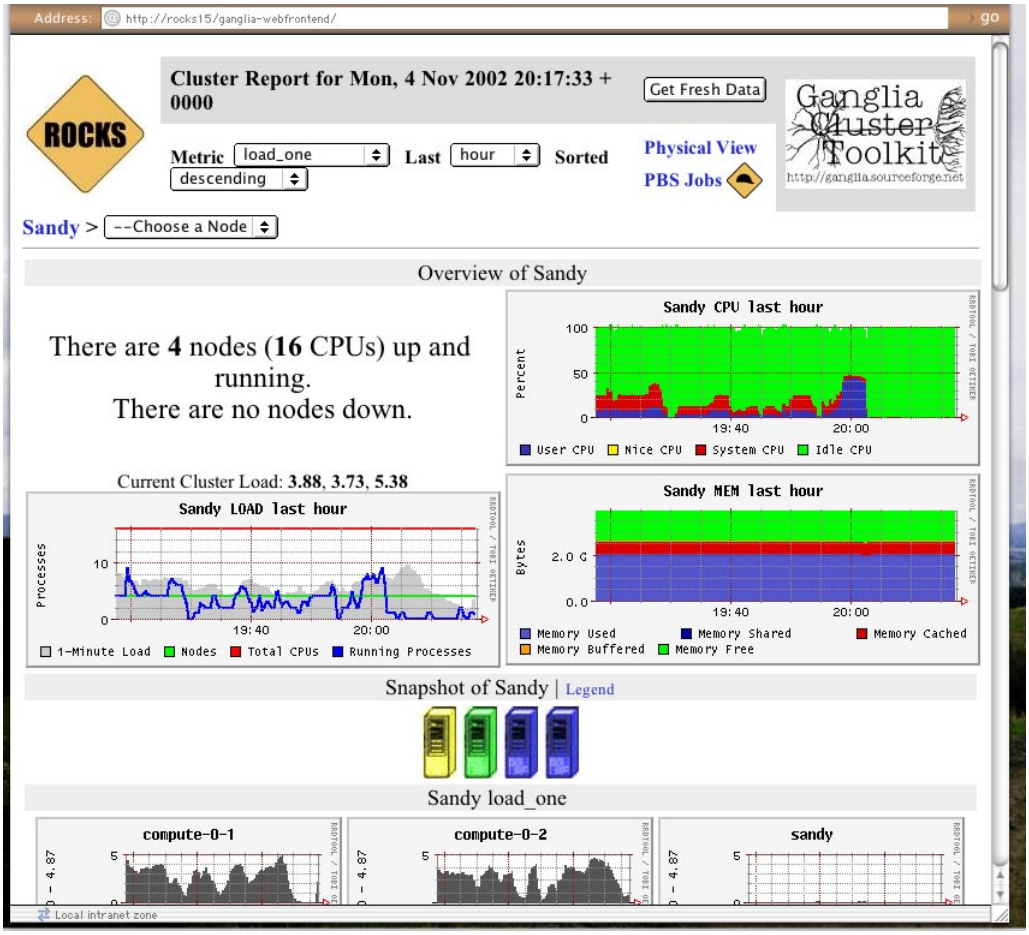

Figure 3-3 Ganglia Cluster Report

3.4 Other Cluster Monitoring Facilities

3.4.1 Proc File System

The next link leads to a standard Apache file system view of the Linux /proc file system. These files and directories are dynamically generated by the Linux kernel upon request. They are used

to convey dynamic information about resource usage and running processes on the machine. The information provided by the /proc files is extremely fresh, and represents the current state of the operating system at the time the file was requested.

However, data contained in these files may reveal information useful to hackers and other malicious parties. In addition to user names and program parameters, this area contains data about local network interfaces and firewalls. Therefore, by default this link is subject to the same "private network only" restriction as the database web interface.

3.4.2 Cluster Distribution

This link displays a file system view of the /home/install/ directory tree on the Frontend node. This area holds the repositories of RPM packages used to construct nodes in the cluster, along with the XML kickstart graph that defines the various node types. The distribution used to build the cluster may be examined here.

Knowledge of the software versions present on the cluster is considered sensitive since it may give hackers information about available security holes. By default, access to this link is restricted to the private network as well.

3.4.3 Rocks User's Guide

The final link on the Table of Contents website page leads to the original NPACI Rocks User's Guide. This is simply a local version of the guide present at www.rocksclusters.org. Since the Rx User's Guide and the Rocks User's Guide differ in many respects, we don't recommend that you refer to this document as the Rx User's Guide. This link will be removed and replaced with an online version of this manual in a future release.

Chapter 4

Customizing Your Rx Installation

This chapter provides information about how to add rpms, Ethernet interfaces, create custom kernels and change default configurations.

4.1 Adding Packages to Compute Nodes

Follow these steps to add rpms.

1. Copy the rpm you want to add into the following directory:

   `/home/install/contrib/7.3/public/arch/RPMS`

   Note: The subdirectory `arch` is the microprocessor architecture ("i386" or "ia64").

2. Build a new Rx distribution. This binds the new package into a Red Hat compatible distribution in the directory `/home/install/rocksdist/....`

   ```
   # cd /home/install
   # rocks-dist dist
   ```

3. Create a new XML configuration file that will extend the current `compute.xml` configuration file:

   ```
   # cd /home/install/profiles/site-nodes/
   # cp skeleton.xml extend-compute.xml
   ```

4. Inside `extend-compute.xml`, add the package name by changing the section from:

   `<package> <insert your package name here> </package>`

   to:

   `<package> your package </package>`

   Note: Enter the base name of the package, not the full name, in `extend-compute.xml`. For example, if the package you are adding is named `XFree86-100dpi-fonts-4.2.0-6.47.i386.rpm`, input *`XFree86-100dpi-fonts`* as the package name in `extend-compute.xml`. For example:

   `<package>XFree86-100dpi-fonts</package>`

5. Reinstall the compute node with shoot-node or by resetting the node.

4.2 Customizing the Configuration of Compute Nodes

Create a new XML configuration file that will extend the current compute.xml configuration file:

```
# cd /home/install/profiles/site-nodes/
# cp skeleton.xml extend-compute.xml
```

Inside extend-compute.xml, add your configuration scripts that will be run in the post configuration step of the Red Hat installer.

Put your bash scripts in between the tags <post> and </post>:

```
<post>
<insert your scripts here>
</post>
```

To apply your customized configuration scripts to compute nodes, reinstall them.

4.3 Exporting Accessible /home Directory

How do I export I export a new directory from the Frontend that is accessible to all the compute nodes under `/home`?

Follow these steps:

1. Add the directory you want to export to the file `/etc/exports`.

   For example, if you want to export the directory `/export/data`, add the following to `/etc/exports`:

   ```
   /export/data 10.0.0.0/255.0.0.0(rw)
   ```

This exports the directory to nodes that are on the internal network only (in the above example, the internal network is configured to be `10.0.0.0`)

2. Restart NFS:

   ```
   # /etc/rc.d/init.d/nfs restart
   ```

3. Add an entry to `/etc/auto.home`.

If you want /export/data on the Frontend node (named frontend-0) to be mounted as /home/data on each compute node, for example, add the following entry to /etc/auto.home:

```
data frontend-0:/export/data
```

4. Finally, rebuild the NIS database:

```
# make -C /var/yp
```

Now when you login to any compute node, and change your directory to /home/data, it will be automounted. Refer to *Managing NFS and NIS* published by O'Reilly and written by Hal Stern, Mike Eisler, and Ricardo Labiaga for additional information.

4.4 Configuring Additional Ethernet Interfaces For Compute Nodes

On compute nodes, Rx uses the first Ethernet interface (eth0) for the following functions:

- Management (reinstallation, for example)
- Monitoring (Ganglia, for example)
- Message passing (MPICH, over Ethernet for example)

Often, compute nodes have more than one Ethernet interface. This procedure describes how to configure them.

Additional Ethernet interfaces are configured from the Frontend with a command line utility called add-extra-nic. It manipulates the networks database table on the Frontend to add information about an extra interface on a node (a description of the networks table can be found in Chapter 5. See "Networks" on page 63. This program simply manipulates the database. It doesn't change any of the existing configuration information about a running node.

Once the information has been entered into the networks table, the additional NIC is configured, each time you reinstall. The structure supports multiple additional interfaces on each node.

For each node that has an additional Ethernet interface on the Frontend, execute:

```
# add-extra-nic --if=<interface> --ip=<ip address> --netmask=<netmask> --name=<host
name> <compute node>
```

Where:

interface	The name of the Ethernet interface (eth1 for example)
ip address	The internet address for the interface (192.168.1.1 for example)
netmask	The network mask for the interface (255.255.255.0 for example)
host name	Host name for the interface (fast-0-0 for example)
compute node	The name of the compute node to apply the configuration to (compute-0-0 for example)

For example, if you want to configure interface eth1 for compute node compute-0-0 with the IP address 192.168.1.1 with a netmask of 255.255.255.0 and you want to name the new interface fast-0-0, the call to add-extra-nic would be:

```
# add-extra-nic --if=eth1 --ip=192.168.1.1 --netmask=255.255.255.0 --name=fast-0-0
compute-0-0
```

Reinstall the nodes that you have defined an additional interface for (use shoot-node).

4.5 Enabling RSH on Compute Nodes

The default Rx configuration does not enable rsh commands or rlogin to compute nodes. Instead, Rx uses ssh as a drop in replacement for rsh. There may be some circumstances where ssh does not have exactly the same semantics of rsh. Further, there may be some users that cannot modify their application to switch from rsh to ssh. If you are one of these users you may wish to enable rsh on your cluster.

Warning: Enabling rsh on your cluster has serious security implications. While it is true rsh is limited to the private-side network, this does not mean it is as secure as ssh. Be sure to ask the site security expert (if applicable) about the security risks involved with enabling rsh.

Enabling rsh is done by modifying the default kickstart graph. Using your favorite text editor open the file /home/install/profiles/2.3/graphs/default.xml and search for the following block of code:

```
<!-- Uncomment to enable RSH on your cluster (this is not very secure!)
<edge from="slave-node" to="xinetd"/>
<edge from="slave-node" to="rsh"/>
-->
```

Next, follow the instructions and uncomment this block. This forces all appliance types that reference the slave-node class (compute nodes, NAS nodes, ...) to enable an `rsh` service that trusts all hosts on the private side network. This uncommented block should look like this.

```
<edge from="slave-node" to="xinetd"/>
<edge from="slave-node" to="rsh"/>
```

The next step is to re-install your compute nodes to add the changes.

4.6 Disabling Reinstallation After A Hard Reboot

When compute nodes experience a hard reboot (when the compute node is reset by pushing the power button or after a power failure, for example), they will reformat the / file system and reinstall their base operating environment.

Follow these steps to disable this feature:

1. Login to the Frontend.
2. Edit the file `/home/install/profiles/nodes/auto-kickstart.xml`.
3. Remove the line:

   ```
   <package>rocks-boot-auto<package>
   ```

4. Reinstall all your compute nodes

   An alternative to reinstalling all your compute nodes is to login to each compute node and execute:

   ```
   # /etc/rc.d/init.d/rocks-grub stop
   # /sbin/chkconfig --del rocks-grub
   ```

   To do all the compute nodes at once from the Frontend, execute:

   ```
   # cluster-fork /etc/rc.d/init.d/rocks-grub stop
   # cluster-fork /sbin/chkconfig --del rocks-grub
   ```

4.7 Creating a Custom Kernel rpm

This procedure involves bringing up a compute node with Rx first, then logging on to the compute node to make the custom kernel. Additional general information about this process is found in the *Rx Supplemental Documentation* binder and at www.tldp.org/HOWTO/Kernel-HOWTO/index.html.

1. Login to a compute node. For example:

   ```
   # ssh compute-0-0
   ```

2. Go to the directory where the Linux kernel source code resides:

   ```
   # cd /usr/src/linux-2.4
   ```

3. Build your .config file.

   Hint: We recommend that you copy the appropriate config file from the configs directory to .config, then edit it to suit your needs. For example, if you want to configure a kernel for a SMP-based i686, input:

   ```
   # cp configs/kernel-2.4.9-i686-smp.config /usr/src/linux-2.4/.config
   ```

   To determine the processor architecture of the node, execute:

   ```
   # uname --machine
   ```

4. Build a kernel rpm based on your modified .config:

   ```
   # make rpm
   ```

5. Copy the resulting rpm back to the Rx distribution on the Frontend. The final lines of the make rpm command indicate the name of the resulting kernel rpm.

   For example:

   ```
   # scp /usr/src/redhat/RPMS/i686/kernel-smp-2.4.9-31.i686.rpm \
   frontend-0:/home/install/rocks-dist/7.3/en/os/i386/force/RPMS/
   ```

6. Rebuild the distribution on the Frontend:

   ```
   # ssh frontend-0
   # cd /home/install
   # rocks-dist dist
   ```

   Your new kernel is now applied to the Rx distribution.

7. Test the new kernel by reinstalling a compute node:

   ```
   # shoot-node compute-0-0
   ```

If the kernel works satisfactorily, reinstall all compute nodes that you want to run the new kernel.

4.8 Making Your Own Cluster Distribution Media

If you want to incorporate any upgrades, additions or other changes made to the Rx packages into a media set, you must execute the following commands before building the media set.

1. Login to the Frontend.
2. Type the following commands:

```
# cd /home/install
# rocks-dist dist
```

3. Build the media set (in this case, build the CD set) by entering the following commands:

```
# cd /home/install
# rm -rf cdrom
# rocks-dist --dist=cdrom cdrom
```

This puts the CD set under /home/install/cdrom/7.3/en/os.

The first three ISOs have the following names, and any additional ISOs follow the same naming convention:

1. /home/install/cdrom/7.3/en/os/rocks-disk1.iso
2. /home/install/cdrom/7.3/en/os/rocks-disk2.iso
3. /home/install/cdrom/7.3/en/os/rocks-disk3.iso

The procedure is complete; you now have a customized cluster distribution ready to burn onto blank CD media.

Chapter 5

Resources

This section describes the SQL Database Schema used by the Rx cluster.

5.1 Rx Cluster Database Schema

The MySQL DBMS server manages the schema in a single database named cluster. This database forms the backbone of the Rx system, coordinating tasks as diverse as kickstart, node typing, configuration file building, and versioning.

5.1.1 Relational Schema

This diagram describes the database relations in standard UML notation. It is shown in Figure 5-1.

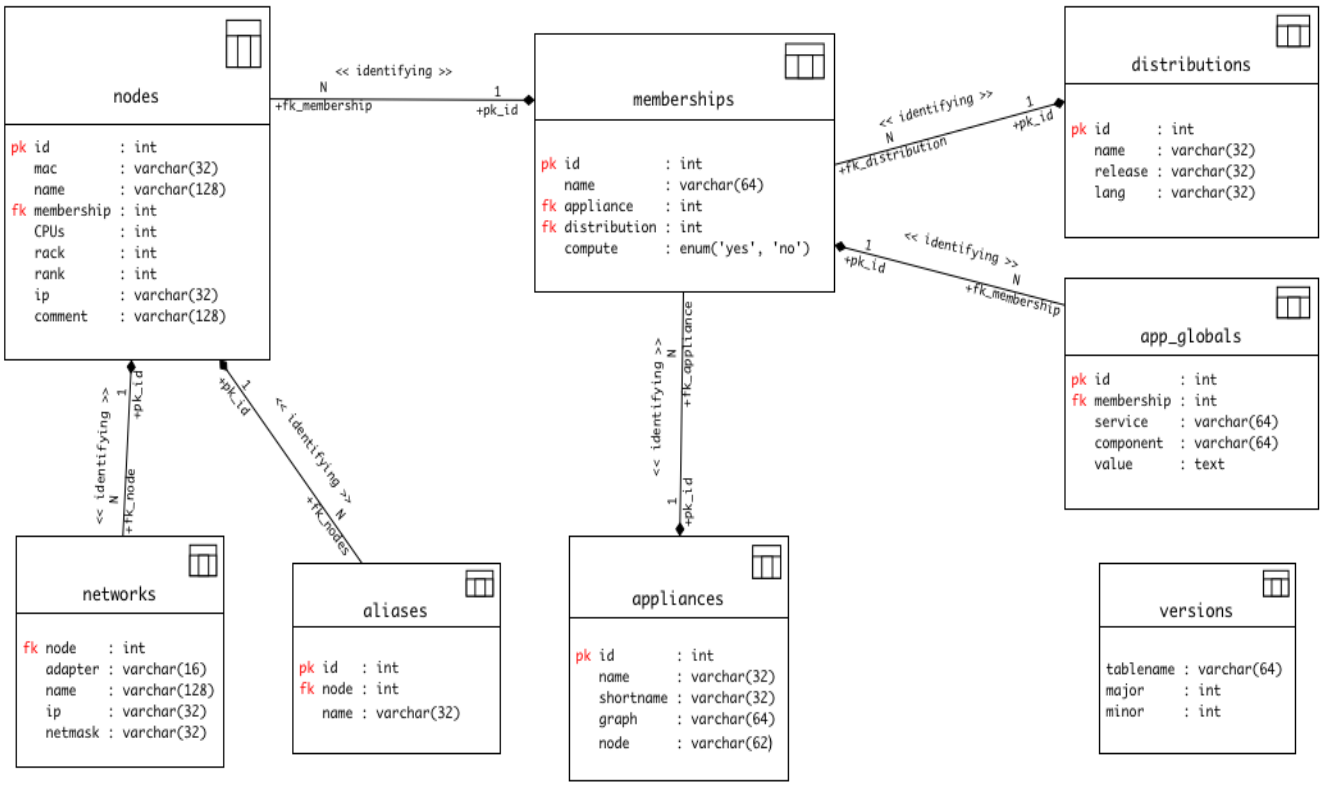

Figure 5-1 Relational Schema Diagram

5.1.2 Cluster Database

These tables are contained in the cluster database, and each has the relationship to the others as shown in Figure 5-1, the Relational Schema Diagram. Each table is described in the following sections of this chapter.

- Aliases
- App_globals
- Appliances
- Distributions
- Memberships
- Networks
- Nodes
- Versions

5.1.3 Tables

A subset of the Information Engineering (IE) notation method is used to describe these tables. Primary keys are marked with an asterisk (`*`), and foreign keys are designated by an at symbol (`@`). Attempts have been made to ensure this schema is in the Second Normal Form (2NF).

5.1.3.1 Aliases

The aliases table shown in Figure 5-1 contains any user-defined aliases for nodes. Table 5-1 shows the Field and Type values for any user-defined aliases.

Table 5-1 Aliases Table Values

Field	Type
ID*	int (11)
Node@	int (11)
Name	var (32)

The following list describes the Field values in Table 5-1.

ID A primary key integer identifier. Auto-incremented.

Node A foreign key that references the `ID` column in the Nodes table.

Name The alias name. Usually a shorter version of the hostname.

5.1.3.2 App_Globals

The app_globals table shown in Figure 5-1 contains Key=Value pairs used for essential services such as Kickstart. Examples include the Keyboard layout, Public Gateway, Public Hostname, DNS servers, IP mask, Cluster Owner, Admin email, etc.

Table 5-2 shows the Field and Type values.

Table 5-2 App_Globals Table Values

Field	Type
ID*	int (11)
Membership@	int (11)
Service	varchar (64)
Component	varchar (64)
Value	text

The following list describes the Field values in Table 5-2.

ID A primary key integer identifier. Auto-incremented.

Membership A foreign key that references the `ID` column in the Membership table.

Service The service name that will use this KEY=VALUE pair. Examples are *`Kickstart`* and *`Info`*.

Component The key name for this row. Corresponds to the name attribute of the `<var name="key"/>` XML tag during the Kickstart Pre-Processing (KPP) phase of Rx node configuration.

Value The value of this row. Can be any textual data.

5.1.3.3 Appliances

The appliances table shown Figure 5-1 defines the available appliance types. Each node in the cluster may classify itself as a single appliance type. The Graph and Node attributes define the starting point in the Rx software configuration graph, which wholly specifies the software installed on a node.

Table 5-3 shows the Type values associated with each Field.

Table 5-3 Appliances Table Values

Field	Type
ID*	int (11)
Name	varchar (32)
Shortname	varchar (32)
Graph	varchar (64)
Node	varchar (64)

The following list describes the Field values in Table 5-3.

ID A primary key integer identifier. Auto-incremented.

Name The name of this appliance. Examples are *Frontend* and *compute*.

ShortName A nickname for this appliance.

Graph Specifies which software configuration graph to use when creating the kickstart file for this appliance. The default value of `default` is generally used.

Node Specifies the name of the root node in the configuration graph to use when creating the kickstart file for this appliance. The software packages for this appliance type are fully defined by a traversal of the configuration graph beginning at this root node.

5.1.3.4 Distributions

The distribution table shown in Figure 5-1 connects a membership group to a versioned Rx distribution, and plays an important role in the Rx kickstart file generation process. The Release relates to the Red Hat distribution version (7.3 for example), while the Name specifies where to find both the Rx configuration tree and RPM packages. The location of these resources is in the /home/install/<Name>/<Release] directory.

Table 5-4 shows the Type values associated with each Field.

Table 5-4 Distributions Table Values

Field	Type
ID*	int(11)
Name	varchar(32)
Release	varchar(32)
Lang	varchar(32)

The following list describes the Field values in Table 5-4.

ID A primary key integer identifier. Auto-incremented.

Name Specifies where to find the Rx configuration tree graph files. The Name field of the configuration graph located in the /home/install/<Name>/<Release>/ directory.

Release Gives the Red Hat distribution version this configuration tree is based on, 7.3 for example. The Release field in the graph location 7.3 /home/install/<Name>/<Release>/ directory.

Lang The language of this distribution. A two-letter language abbreviation like *en* for English or *fr* for French.

5.1.3.5 Memberships

The memberships table shown in Figure 5-1 specifies the distribution version and appliance type for a set of nodes. An alternative name for this table would be *groups*, however that is a reserved word in SQL. The memberships table names a group of nodes in the cluster and allows multiple memberships to tie into one appliance type.

Table 5-5 shows the Type values associated with each Field.

Table 5-5 Memberships Table Values

Field	Type
ID*	int(11)
Name	varchar(32)
Appliance@	int(11)
Distribution@	int(11)
Compute	enum(*yes*, or *no*)

The following list describes the Field values in Table 5-5.

ID A primary key integer identifier. Auto-incremented.

Node The name of this membership. A type of node in the cluster, like *Frontend*, *Compute*, *Power Unit* or similar. The software installed on nodes in a given membership is defined by an appliance ID.

Appliance A foreign key that references the `ID` column in the Appliances table. Helps define the software installed on nodes in this membership, and therefore their behavior.

Distribution A foreign key that references the `ID` column in the Distributions table. The second key used to define the software for nodes in this membership.

Compute Either *yes* or *no*. Specifies whether this type of node will be used to run jobs.

5.1.3.6 Networks

The networks table is shown in Figure 5-1. It describes and configures network interfaces on networks other than the standard private cluster network. For example, if a node has five network cards, it would have four entries in this table: the fifth is the "standard" interface with the IP address as listed in the nodes table.

Table 5-6 shows the Type values associated with each Field.

Table 5-6 Networks Table Values

Field	Type
Node@	int(11)
Adapter	varchar(16)
Name	varchar(128)
IP	varchar(32)
Netmask	varchar(32)

The following list describes the Field values in Table 5-6.

Node A foreign key that references the `ID` column in the Nodes table.

Adapter The name of a network interface, such as <*eth0*>, or <*ppp0*>.

Name A descriptive name for this type of network interface, like <*Myrinet*> or <*Gigabit-Ethernet*>.

IP The IP address assigned to this interface.

Netmask The bit mask used to define the network ID and broadcast portion of the IP address for this interface. Can be specified as <*255.255.255.0*> or <*/24*>.

5.1.3.7 Nodes

The nodes table is shown in Figure 5-1. It is a central table in the schema. The nodes table holds one row for each node in the cluster, including the Frontend, compute, and other appliance types. The node's location in the physical cluster (which rack it lies in on the floor, for example) is specified in this table as well.

Table 5-7 shows the Type values associated with each Field.

Table 5-7 Nodes Table Values

Field	Type
ID*	int(11)
MAC	varchar(32)
Name	varchar(128)
Membership@	int(11)
CPUs	int(11)
Rack	int(11)
Rank	int(11)
IP	varchar(32)
Comment	varchar(128)

The following list describes the Field values in Table 5-7.

ID A primary key integer identifier. Auto-incremented.

MAC The 6-byte Media-Access-Layer address (Layer 2) of the node's first (eth0) Ethernet adapter.

Name The hostname for this node. Rx convention is `compute-<Rack>-<Rank>` for compute nodes, and `frontend-[id]` for Frontend nodes, where *id* is an integer identifier.

Membership A foreign key that references the `ID` column in the Memberships table. Specifies what type of node this is.

CPUs The number of Processors in this node. Defaults to 1. Although this column violates the second normal form, it is more useful here than in a separate table.

Rack The X-axis coordinate of this node in euclidean space. Zero is the leftmost rack on the floor by convention. Note that we only use a 2D matrix to locate nodes, the plane (Z-axis) is currently always zero.

Rank The Y-axis of this node in euclidean space. Zero is closest to the floor (the bottom-most node) by convention.

IP The IPv4 Internet Protocol address of this node in decimal-dot notation.

Comment A textual comment about this node.

5.1.3.8 Versions

The versions table shown in Figure 5-1 is intended to provide database schema versioning. It is currently not widely used.

Table 5-8 shows the Type values associated with each Field.

Table 5-8 Versions Table Values

Field	Type
TableName	varchar(64)
Major	int(11)
Minor	int (11)

The following list describes the Field values in Table 5-8.

TableName The name of a table in this database schema.

Major The major version number of this table. Usually the first integer in the version string.

Minor The minor version number of this table. The second integer in the version string.

Appendix A

Basic Cluster Concepts and Terminology

A.1 Cluster Concepts

A.1.1 Introduction

The concept of using more than one computer to solve a problem is not new. What has changed during the last decade is the manner in which computer clusters are built and their applications. Two of the most popular uses for computer clusters are:

- Hosting website services
- Supercomputing, also known as high performance computing (HPC)

Website services are their own speciality, the focus in this documentation is on supercomputing. The term “supercomputer” needs definition. The goal isn’t to build and use a traditional, monolithic mainframe supercomputer like the Cray T3E. The goal instead is a supercomputer made from cutting edge, modular clusters of personal computers with the capacity for parallel processing similar to a Cray or other traditional supercomputers.

The modular units that the supercomputer will be built of are ordinary, commodity, personal computers (PCs). The same technological and economic trends responsible for increasing desktop PC performance, affordability, and availability have also had a profound affect on the lofty realm of supercomputing. These trends are summarized with Moore’s Law: the PC performance you can obtain for a given dollar amount essentially doubles about every 18 months. The ubiquitous PC is now powerful enough and cheap enough to have become the building block of choice when constructing parallel processing computers. To achieve a certain level of processing power nowadays, surprisingly, it is currently more affordable to network many ordinary PCs together with Ethernet into a loosely coupled parallel processing cluster than it is to purchase a tightly integrated, traditional supercomputer built with proprietary, expensive components.

These new types of high performance computational clusters built from commodity PCs and free operating system software, like Linux, are known as a “Beowulf” or a “Beowulf cluster.” Although Beowulf clusters in excess of 1000 CPUs have been built, the Beowulf cluster you build can be as large or as small as your space and budget allow. Your first Beowulf can be built with

two PCs or two hundred. When the opportunity presents itself to expand the computational power of your Beowulf, it's simply a matter of adding more PCs to the cluster.

A.1.2 Generic Clusters

A computer cluster, or just cluster, is a broad term that simply means a number of computers networked together to collectively provide some sort of service. The computers in a cluster are networked together and programmed to pool their resources in sharing a common workload, like serving web pages.

For example, consider a website that has excessive hits during peak hours. Instead of replacing the web server with a faster computer, you can place another computer (faster or slower) on the network and install some software on both computers, allowing them to share the web serving workload. In addition, adding a second computer to assume the whole workload if the other computer fails adds what is called "failover capacity." The web pages are now served by a two-node, loadsharing cluster with failover capability. Adding a third computer to your cluster can provide additional redundancy. Two computers can be programmed to share the web serving workload, while the third can be programmed to sit idle until one of the working computers fails. If one computer crashes, there is another computer available to assume the workload of the failed computer and the workload is still divided evenly between the two computers. In the event a second computer should fail, the remaining computer can assume the whole web serving workload.

Clusters built around this theme of spreading required services over multiple computers on a network are often used to provide high availability, load balancing, failover, fault tolerance, and redundancy to the Internet's web sites, search engines, mail servers, file servers, and databases. When it comes to hardware and software requirements, these computer clusters have no restrictions on cluster membership, and can largely be constructed with any kind of computer hardware and operating system software including Linux. Members of this kind of cluster also don't have to reside on the same local network, but can be distributed over a local area network (LAN) or wide area network (WAN). More information on building this type of cluster can be found at http://www.linux-ha.org and http://www.linuxvirtualserver.org.

A.1.2.1 Beowulf Clusters

Beowulf cluster, on the other hand, is a very different type of cluster that is used strictly for high performance computing, also known as supercomputing. While a web serving cluster is comprised of multiple computers that serve more web pages than any one of its members could, a Beowulf is comprised of multiple computers that cooperate to do more computations than any of its computers could do alone. Similar to classic "big iron" supercomputers, Beowulf clusters are built

to harness the power of multiple processors in solving computationally intensive problems. When the term "cluster" is used without modification in this documentation, the reference is to a Beowulf cluster running Linux. Unlike the hardware and software heterogeneity and flexible configurations commonly found with other clusters, a Beowulf has strict requirements with regard to its makeup and architecture. A Beowulf is comprised of commodity hardware, commodity operating system software, and commodity communication software.

A.1.2.2 Where do Beowulf clusters come from and where are they used?

While other distributed computing cluster projects like UC Berkeley's network of workstations (NOW) project predate Beowulf, there were no dedicated high performance commodity computing cluster options at the time Thomas Sterling and Donald Becker initiated the Beowulf project (out of necessity no less) at NASA CESDIS in 1993. All the high performance computing at the time was being conducted on very expensive and exclusive supercomputers built with both proprietary hardware and software. It's the deviation from this model that made the first Beowulf unique. While solving problems with free software in general wasn't a novel concept, utilizing free software and standardized, commodity hardware for high performance computing was.

Beowulf clusters are in use at the San Diego, Pittsburgh, Maui, and Ohio Supercomputer Centers; Los Alamos, Lawrence Livermore, Sandia, Pacific Northwest, Argonne, Fermi, and Oak Ridge National Laboratories; the National Center for Supercomputing Applications (NCSA); and many more universities and businesses. Similar to the traditional supercomputers still in use at many of these locations, the Beowulfs are being used to perform fluid dynamics analyses, aeronautical flight simulation, automotive crash modeling, nuclear reaction simulations, astronomical measurements and predictions, climate modeling, and image rendering, to name just a few applications. Many of these clusters are also assisting scientists with research in various fields of bioinformatics, computational biology, and chemistry.

A.1.3 Commodity Hardware for Clusters

Commodity computer hardware refers to the ubiquitous, modular, interchangeable, replaceable components based on industry standards found inside a typical PC. The economics of ordinary PC disk drives, RAM, power supplies, floppy drives, CD-ROM drives, Ethernet network interface cards (NIC), video cards, and CPUs, drive the affordability and performance of computer clusters. While Beowulf-like clusters can be built using more specialized hardware, such as that found in sophisticated engineering and graphics design workstations, this deviates from the definition of a Beowulf cluster based on PC components.

Beowulf clusters can be built with nearly any PC make and model. The first Beowulf was built with 16 100 MHz DX4 processor-containing PCs, with 16 MB of RAM, either 540 MB or 1GB

of hard disk space per node, and 10 Mb/s Ethernet networking infrastructure. Since then, Beowulfs have been built using both old and cutting edge PC hardware. Although a common stigma surrounding Beowulf clusters is that they are made from old, recycled PCs that were destined for donation, the fact that more and more Beowulf clusters built with PC components have made their way onto the list of the top 500 fastest supercomputers in the world (http://www.top500.org) dispels this notion.

A.1.4 Free Software For Clusters

If the reality that the future of high performance computing lies with the price/performance ratios associated with lowly PC hardware comes as a surprise, then it will be a pleasant shock that this new breed of supercomputers is run with commodity software—free software that no one owns.

The commodity operating system software on a Beowulf consists of an open source Unix-like operating system in the form of Linux or BSD. We'll be concentrating solely on Linux Beowulf clusters in this documentation, but the Beowulf principles apply just the same to supercomputing clusters built with open source flavors of BSD or other open source Unix-like operating systems.

Linux was designed by Linus Torvalds to be a Unix work-alike operating system for a PC. Linus built the heart of the operating system—the Linux kernel—from scratch, bundled it with some freely available Unix-compatible tools to round Linux out as an operating system, and released it to the world in 1991. The Linux global mindshare has been extending, modifying, and improving Linux ever since, under the unwavering hand of Linus himself. Today Linux is recognized as a very robust and flexible operating system that runs on many processor architectures.

Linux's growth and maturity into commodity software is due to the fact that it is comprised of freely distributable software components, including many freely available GNU (GNU Not Unix) software tools and the Linux kernel, that are protected under the General Public License (GPL). The GPL, sometimes referred to as the "copyleft," is the license created by Richard Stallman of the Free Software Foundation (FSF) that covers all GNU software (http://www.gnu.org) and the Linux kernel. Linus' decision to release the Linux kernel under the GPL and bundle it with several widely used and GPL licensed GNU software development tools represents the crux of Linux' success. The bundling of the Linux kernel with GNU software is also why some groups such as the FSF, refer to Linux as GNU/Linux.

The defining characteristic of the GPL is that all software distributed under the GPL is free in terms of source code availability and modifications. A more exacting definition of free software and the freedoms associated with it, as mandated by the FSF, can be found at http://www.gnu.org/philosophy/freesw.html. But basically, software distributed under the GPL must contain the program source code. When you obtain a copy of GPL licensed software, like the Linux kernel

from http://www.kernel.org or a full Linux distribution, the program's source code is included so that you may modify, extend, and debug any program to your heart's content. Further, if a GPL licensed program is modified and then redistributed--actions allowed under the GPL--all the modifications made to the program's source code must be included in the redistribution. In this manner, the GPL fosters community, collaboration, and improvement of free software while protecting the software's original authors' desire to keep their software free.

The commodity communication software installed on Beowulf clusters consists of portable implementations of the same software development tools, batch schedulers, and parallel processing libraries used by the rest of the high performance computing (HPC) community. Parallel processing software libraries enable intra-application and inter-node communication, and are absolutely essential to coupling the individual computers in a Beowulf, commonly referred to as "compute nodes" or just "nodes", into a parallel processing machine. The ability to use the same software development tools, libraries, and queuing systems typically found on traditional supercomputers fostered the porting of other HPC tools to Linux, the rapid evolution of Linux as the de facto Beowulf operating system, and the formation of a Beowulf community.

A.1.5 Parallel Programming on Clusters

Beowulf clusters were originally designed to run parallel programs. Parallel programs are compute intensive programs specifically written to take advantage of multiple processors. The goal of parallel programs, and parallel processing in general, is to shorten a program's runtime by dividing computational work over multiple processors. Parallel programs get the name "parallel" because the processors working on a parallel program run concurrently (in parallel), as opposed to sequentially, and cooperate to speed up program execution relative to the same program running on a single processor. For a program to run in parallel and divide work over multiple processors, additional communication software must be incorporated into the program that makes it possible for different parts (processes) of a single parallel application talk to each other (intra-application communication) even though the individual processes of that application are running on different processors (inter-node communication). Obviously, many compute intensive applications are written from scratch to run in parallel. This is no small achievement. Efficient parallel programming is a complicated pursuit and outside the scope of this document.

To take advantage of a cluster's multiple processors without learning copious parallel programming skills, many people have modified already existing serial programs (programs that were originally designed to run on a single processor) into parallel programs that utilize intra-application and inter-node communication software and can take advantage of multiple processors. There are many challenges in parallelizing a program from its serial state (single-processor mode) to running in parallel, and some program types are easier to parallelize than others.

Serial programs that are relatively easy to parallelize are often referred to as "embarrassingly parallel" or "pleasantly parallel." Sometimes these programs are retrofitted with the communication software, but a more common technique used in actualizing the parallel nature of a serial program is not touching the application's code at all, but splitting up the data to be crunched into smaller portions and feeding those smaller datasets to individual instances of the program running on different processors. After the individual programs are finished running, the results are assembled into a single results file, so as to make the final readout indistinguishable from the readout had one instance of the program run through the whole dataset on a single processor. This method, often referred to as "data parallelization" or "domain decomposition," is a simple and powerful way of taking advantage of multiple processors without modifying a serial program's code base to utilize communication software.

While running multi-processor parallel programs on a Beowulf is its raison de être, and running serial applications in an embarrassingly parallel fashion using data parallelization over multiple processors is a close second, running many independent, single-processor, serial jobs without data parallelization is also a very popular mode of Beowulf usage. This is necessary when parallelizing a program either with communication software or data parallelization techniques isn't much more efficient and so not worth the effort, or for whatever reason, simply not an option. All one can do in these situations is attempt to execute the single-processor jobs on the compute nodes with the fastest CPU, most memory, or other resource that will speed up a particular program's execution. Fortunately, the process of submitting either truly parallel, "data parallel" serial applications, or unmodified, single-processor serial applications for execution on a Beowulf is the same; the workload management software layer handles them all.

A.1.5.1 Commodity Ethernet Networking

Just as affordable PC motherboards, CPUs, hard drives, and RAM were key to the overall processing power of the first Beowulf, the availability of a commodity networking technology was critical to interconnecting the individual computers so that they could operate as a single system. The first Beowulf was networked together with 10 Mbps (megabits per second, a megabit is one million bits) Ethernet, and Ethernet in all its flavors remains the most popular interconnect for Beowulf today.

Ethernet is a commodity networking technology—standardized, ubiquitous, inexpensive, robust, and the most widely installed networking technology for computer LANs. Although nowadays, Fast Ethernet technology and Gigabit Ethernet that operate at 100 Mbps and 1000 Mbps is more popular and replacing the older 10 Mbps Ethernet as the default network interconnect. Easing this transition in Ethernet networking upgrades is the fact that Fast and Gigabit Ethernet equipment is backward compatible with regular Ethernet and the hardware often supports auto-negotiation of networking speeds. This allows Beowulf users to seamlessly upgrade Ethernet based Beowulf networks.

The most obvious difference between Ethernet and Fast Ethernet networks is their peak bandwidth, 10 Mbps verses 100 Mbps. Bandwidth refers to the amount of information or bits that can be moved over a medium per unit time and is often used as a measure of network speed. However, network bandwidth isn't the only performance metric you need to consider when deciding on the optimal network for your Beowulf. Network latency is also important. Latency refers to the time elapsed from the time a program requests a signal or data transfer to the time it actually gets completed. Ethernet and Fast Ethernet latencies happen to be roughly the same, approximately 60 microseconds. So, while there is a similar system delay in sending a message over either type of Ethernet network, a 100 Mbps Ethernet network can move 10 times as many bits per second as a 10 Mbps network, and therefore has the higher capacity and perceived speed.

While several hundred microseconds doesn't sound like a long time, these delays do add up and can become a serious hindrance to application performance if your compute nodes are sitting idle while waiting for messages to be sent and received. While serial programs do not generate much or any inter-node communication, and therefore are not really affected by network latency, these message passing delays pose a serious scalability obstacle for truly parallel programs.

To give you an idea of the scalability issues, let's say that there are 1000 messages that need to be passed to all of the nodes during a program's execution. A priori, we know that there is five seconds of CPU work in this application. So, if we divide the chosen workload over five nodes (1 CPU/node) then each of the five CPUs will spend one second performing computations. Since 5 processors are each doing one-fifth of the work in parallel, the actual perceived run time of the application (also known as the wall clock time) will be 1 second. However, besides the one second spent computing by all the processors, each CPU needs to send and receive 1000 messages to all the other CPUs, including itself, to coordinate execution of the computation in parallel. After requesting a message be sent, a node's CPU sits idle through the network delay until an acknowledgement that the sent message has been received is returned. If the message latency is 50 microseconds (and all message latencies are the same), then the total latency due to message passing is 2000 messages (1000 sends + 1000 receives) x 5 nodes x 50 microseconds = 500,000 microseconds, or one half second. The individual latencies are additive because no two sends or receives can occur simultaneously. In this example, the total latency is 10% of the total CPU work (.5s/5s) and adds an additional 50% to the wall clock time (.5s/1s)! If the number of compute nodes is increased to 50, then the total message passing latency is 5 seconds, which is equivalent to the total CPU work and increases the wall clock time to 400% of the wall clock time when using 5 processors (6s/1.5s)! Even if you were to experience a ten-fold linear speed up in CPU time with 10 times as many processors working on the problem (somewhat unlikely, but not impossible), so that the compute time needed by all the CPUs was reduced from 1s second to.1 second, the wall clock time is still 5.1 seconds, which is greater than triple the wall clock time when run with only 5 processors (5.1s/1.5s). This is definitely a situation where more processors is not better.

While message latencies depend on network and system speeds, and message passing varies per program, the barriers to scalability I've described are an inescapable reality of parallel programming. When running parallel programs, the idea is to avoid the point of diminishing or negative returns that I've described above. One way to do this is to utilize a low latency communication network like Myrinet (see below).

The optimal networking technology for your Beowulf is dictated by whether your application's performance is bound by networking latency or bandwidth. If you are only running serial programs on your cluster and no parallel programs, then you have no need for a low latency network, and a single Fast or Gigabit Ethernet network may provide all the bandwidth you need. If you are using massive datasets and Fast Ethernet doesn't provide the desired performance, then maybe 1 Gbps (gigabits per second, a gigabit is one billion bits) Ethernet (Gigabit Ethernet) or 10 Gbps Ethernet (10 Gigabit Ethernet), which are rapidly approaching affordability and hence gaining popularity, will suffice. While latency improvements may be small with the up and coming Gigabit Ethernet offerings, the bandwidth increases by orders of magnitude. Fortunately, like the backward compatibility from Fast Ethernet to Ethernet, Gigabit Ethernet networking has also been designed to be backward compatible with all previous Ethernet speeds. Almost without exception, a Beowulf contains an Ethernet network. It's for this reason that we are going to refer to a Beowulf's Ethernet network as the primary Beowulf network.

A.1.6 Beowulf Primary Network Design

At its core, the single Beowulf network segment is not much more involved than a simple star topology network. Each node has a network connection to an Ethernet switch (or hub) that allows all the nodes to communicate with each other. Although looking very similar, hubs and switches operate very differently. A hub will broadcast an incoming signal to all the nodes connected to that hub. Incoming signals get sent to every node connected to a particular hub whether intended for it or not. Since all signals coming in to a hub are perpetuated to all the nodes, outgoing signals often collide with other incoming signals and then both signals need to be resent. Since parallel programs rely on inter-node communication, signal collision is naturally a major hindrance to Beowulf performance and to be avoided as much as possible, which is why we recommend switches instead of hubs when networking a Beowulf. A network switch will deliver an incoming signal only to the node it is intended for. This smarter signal routing ability greatly reduces signal collision, increases signal throughput and Beowulf message passing performance. Ethernet switches were too expensive to be incorporated into the first Beowulf, but are much more affordable nowadays.

Not surprisingly, being directly tied to Beowulf network performance, switches are the most expensive component of Beowulf networks. Since the intelligence and network bandwidth of the switch dictates the speed and reliability with which signals are propagated between nodes, switch

selection should be given serious consideration if you're trying to build a very high performance Beowulf. When it comes to switches, like so much computer hardware, you get what you pay for.

The complexity of networking a Beowulf arises when more than one switch is involved. While it's certainly possible to stack full switches on top of each other and connect them with crossover cables, this is not a very efficient or scalable method of networking a Beowulf. It is also rife with single points of failure, and these should be avoided if at all possible by building in alternative signaling pathways. Logistically, providing this type of redundant bandwidth on a large scale becomes rather challenging and expensive, since in the best case scenario it will only double your networking costs. Correctly networking a large Beowulf with acceptable network redundancy is a nontrivial task and beyond the scope of this document.

A.1.6.1 Myrinet

Currently, Myrinet from Myricom (http://www.myri.com) is the most popular low latency network technology, with a network bandwidth of nearly 2Gbps and latency of roughly 8 microseconds. But Myrinet isn't exactly commodity networking hardware like Ethernet, and so you pay a premium for the network speed increase. Myrinet is a standardized, but proprietary protocol, and in order to use Myrinet you need to purchase their very expensive (relative to Ethernet) cables, network cards, and switches. Myrinet networks utilize a message passing system for inter-node communication called GM (general messaging) that is not based on TCP/IP. So in order to use their hardware, you have to download the freely available open source software drivers for the Myrinet network cards and the Myrinet versions of MPI (MPICH-GM), PVM (PVM over GM), and sockets (Sockets-GM) software from their website.

If you install a Myrinet network alongside an Ethernet network on your Beowulf, you can partition network traffic and increase parallel program performance by running communication-intensive applications over the Myrinet network and other applications, as well as administrative tasks, over the Ethernet network.

A.2 Cluster Software

Despite the fact that no two Beowulf clusters are identical, all Beowulfs have plenty of software functionality in common such as an open source OS, software communication libraries, and a workload management system. While Beowulfs may differ in the actual program used to provide a particular function like workload management, this merely represents a variation on a core software component. The following sections investigate each software layer of the core cluster software with a brief description of each.

A.2.1 Linux

Linux is the freely available, open source operating system used as a foundation for the first Beowulf. It is a fully developed, multi-user, and multitasking operating system that employs virtual memory, shared libraries, TCP/IP networking, and everything else you'd expect from a mature Unix operating system. While there are many great things to be said about Linux, Linux was chosen for the Beowulf project because it was a free, standards-compliant, extensible, Unix-like operating system that supported all the common GNU software development tools popular with the HPC community, and it ran on commodity PC processors. It's for these same reasons that Linux is the commodity operating system of choice for building Beowulf clusters today.

A.2.2 Cluster File Systems

One of the more difficult things to manage in a cluster is its file systems. Running applications on a Linux workstation that holds both the necessary programs and data is straightforward. In a default Beowulf configuration the Frontend and all the compute nodes each have a / file system, but all the data and applications are permanently stored on the Frontend. This makes job execution on the Frontend the same as a single Linux workstation. But the purpose of having a cluster is to be able to run applications (a lot of them!) on the compute nodes, while reserving the Frontend's resources for data I/O, cluster services, and administration. So how does one get their applications and data on a compute node for execution? The obvious answer is to copy the programs and data to a compute node. If you want to run jobs on every compute node, then you need to copy your application and data to all the compute nodes. Once your applications and data are on the compute nodes, you can execute a job with a remote command like ssh. The problem with this approach is that every time a file is changed or a new file is needed by the application, it has to be copied to all the compute nodes. Despite the gross inefficiency and tediousness of this approach, if users are allowed to copy applications and data out to any compute node at will, compute node hard drives and file systems would quickly become heterogeneous beyond recognition and the cluster would be unmanageable. Central management of critical file systems like /home, /usr, and /usr/local is needed to prevent a cluster from becoming a graveyard cluttered with outdated files and other application flotsam. On a cluster, the obvious choice for a single centralized location of data and application software is the Frontend. But how can data and applications be perpetuated throughout the cluster in a consistent and controlled manner?

The answer is the network file system (NFS). NFS makes it possible to centrally manage file systems in a distributed computing environment like a cluster. A single copy of a file system like /home on a NFS server can be made available (exported in NFS terms) over the network to as many machines as desired. Computers running an NFS client and proper permission can then mount the exported /home file system over the network, just like it was a file system on a local

hard drive. Filesystems exported by an NFS server and mounted locally provide transparent access to remote file systems by having the same look and feel as if they were located on the local machine. File modifications made to an NFS mounted file system, regardless of where the user is accessing the file system, are physically recorded in one place (the NFS server) and automatically perpetuated everywhere they need to be. NFS enables seamless access to remote file systems and a way to maintain their consistency, while sharing them with all machines on the network.

In a default Beowulf configuration, a NFS server on the Frontend exports all user home directories to all the compute nodes. Upon booting up, each compute node will run an NFS client to mount /home and all its subdirectories over the network, thereby providing user accounts on each node and making any data and applications in the /home directory tree appear to be local and available on every node. With /home NFS-mounted over the whole cluster, Beowulf users can perform all their work by logging into their home directory on the Frontend and all their files will appear to be local to the compute nodes as well. As mentioned before, this is important if users are to be able to launch jobs on compute nodes. Additionally, by using NFS to mount /home over the cluster, a single copy of that file system can be centrally managed by a Beowulf administrator, greatly simplifying many administrative chores like file system backups.

Mounting of /home over a whole cluster via NFS is a common practice that can easily become a performance bottleneck if not managed properly. This is because, despite the apparent locality of NFS-mounted file systems, transferring files via NFS still boils down to copying files over the network. It's just that the copying is done behind the scenes, invisible to the user. Repeated requests for large file transfers over a Beowulf private LAN is a brute force data perpetuation method that does not scale very well and can readily swamp an NFS server. An overwhelmed NFS server typically manifests itself as a sluggish or even unresponsive network, in other words 'nobody's getting any work done'.

One file serving performance enhancement that we will mention here is the use of a RAID configuration on the Frontend. RAID stands for Redundant Array of Independent Disks and is a method by which multiple hard drives can be combined to provide higher disk read/write performance as well as data redundancy. Since the Frontend permanently stores and serves all the important data for the cluster, the speed of the disk subsystem and the integrity of the file systems on the Frontend are far more important than that of the compute nodes, making a RAID configuration desirable.

Compute node hard disks, being lesser citizens in the cluster hard drive community, are partitioned and formatted with one of the default Linux file systems (ext2 or ext3). One advantage of using ext3 is that it is a journaling file system that greatly improves data integrity and reduces the time spent recovering from a node crash. As you can imagine, these are important features for a cluster because its data and file systems need to be coherent and available all of the time. While no important data should be permanently stored on compute node hard drives in a basic Beowulf

configuration, it's still to your advantage to use the ext3 file system on the compute nodes for quicker file system recovery and coherency.

A.2.2.1 Cluster User Account Management

In order for a user to be able to run a job on any Linux workstation, he or she must have an account on the computer with proper access and execution permissions to applications and data. User accounts and file permissions are a simple matter on the Beowulf Frontend because all users log into their home directories that are subdirectories of /home. But what about the compute nodes? Users need accounts and privileges on the compute nodes as well in order for their applications to run. Most problems with user account management typically revolve around making sure that users' accounts and passwords work on all the compute nodes. This information which is stored in the /etc/passwd/ and /etc/group files, is often copied to all the compute nodes (every time there is a change) using a remote execution method like ssh or rsync. This approach suffers from a tedious labor–intensive approach that requires the Beowulf administrator's intervention each time there is a change. Further, if one uses an insecure, unencrypted method, like rsh or remote copy (rcp) to copy the /etc/passwd file across compute nodes, users' passwords can be intercepted and used by the wrong people to do nefarious acts on your cluster.

A.2.2.2 NIS

Formerly known as the Sun Yellow Pages (YP), the Network Information Service (NIS) provides a simple lookup service for information that has to be known by all machines throughout a network, like usernames, passwords, and group associations. For example, if your username and password are recorded in the NIS database on a Beowulf Frontend, you can login to and run applications on all the compute nodes that have the NIS client and access to the NIS database. NIS works well in dynamic cluster environments that have users added (and sometimes deleted) often. The use of a NIS database and lookup service facilitates user addition to a cluster environment by eliminating the need to manually copy files like /etc/passwd, /etc/shadow, and /etc/group out to all the nodes every time one of these files is modified.

A.2.2.3 OpenSSH

OpenSSH (http://www.openssh.org) is a free, open source implementation of the SSH (Secure SHell) protocols that replaces telnet, ftp, rlogin, rsh, and rcp with secure, encrypted network connectivity tools. By using OpenSSH, you are enhancing the security of your machine. All communications using OpenSSH, including passwords, are encrypted. Telnet and ftp use plaintext passwords and transmit all information unencrypted. Unencrypted information can be intercepted, passwords can be retrieved, and then an unauthorized person logging in to your system using one of the intercepted passwords can compromise your system. The OpenSSH suite of utilities should

be used whenever possible to avoid these security problems. SSH is the primary mechanism by which users on the master node access resources on the compute nodes.

A.2.3 Resource and Usage Monitoring

Beowulf cluster system administrators and users often have the following questions:

- Are my jobs running?
- What is the status of my jobs?
- Which nodes are working the hardest?
- Are any nodes unavailable?

Cluster users always want to know whether their jobs are running and why they aren't given the highest priority, administrators want to know how resources are being utilized, managers want to know whether they are getting their money's worth, and everyone needs to justify Beowulf computing expenses. How is the information needed to answer these questions gathered, organized, mined, and reported?

There are several freely available software tools for monitoring system, network, and compute resource availability, collecting resource utilization data, and constructing usage reports. Some of the more popular tools like Mon (http://www.kernel.org/software/mon), Big Brother (http://bb4.com), and Big Sister (http://bigsister.graeff.com) are a collection of shell or Perl scripts. The Ganglia Cluster Toolkit (http://ganglia.sourceforge.net) is an open source, real-time monitoring tool project based in Berkeley that has become the standard for cluster monitoring. More than just a monitoring tool, Ganglia is a very powerful and robust real-time monitoring and remote execution environment that has its roots in the Millennium and NOW projects from UC Berkeley's Computer Science department. Ganglia has been shown to scale and perform well on clusters consisting of 500+ nodes and can also be used to collectively manage multiple individual clusters or clusters integrated into a computational grid.

A basic feature of many resource monitoring tools is the ability to report node usage metrics like CPU load and percentage of memory usage. Ganglia does this exceptionally well by monitoring 24 different metrics by default and providing a very thorough web interface for viewing the statistics in graphical form. Tools are provided that allow you to monitor cluster metrics from the command line and customize the metrics that are monitored and reported by Ganglia. Where Ganglia surpasses other cluster monitoring tools is in its ability to create a remote execution environment for users. Ganglia can authenticate users via their public/private key pairs (cryptographic signatures), and launch applications on cluster nodes on their behalf.

Despite the reams of critical cluster information that Ganglia provides, it does not need to share any cluster files via NFS or utilize any database. The Ganglia monitoring module consists of a multi-threaded daemon that stores all the information it collects into a hash table that resides completely in memory. There's a Ganglia monitoring daemon (gmond) running on every cluster node collecting data, storing it in the node's memory, and then communicating the data to all other nodes via a multicast channel. All nodes receive updated statistics from all other nodes at predetermined intervals. Ganglia operates as a peer-to-peer, lightweight, distributed, in–memory database that can monitor and report all of the cluster's resource usage statistics.

For the most part, individual job monitoring and status reporting is handled by the batch queue managers in the workload management layer. Workload management is discussed below.

A.2.3.1 Workload Management

In a process that is similar to running a program on a single Linux workstation, you can launch Perl or shell scripts from the command line interface (CLI) on a Beowulf Frontend and wait for the output to return. If you were the only user of a Beowulf cluster and you only had one or two applications you needed to run occasionally, this low throughput method of job submission and execution via the CLI might suffice. But this is rare. Typically, Beowulf clusters like most other HPC resources, have multiple users each with their own self-important agendas and resource-sucking applications that keep the cluster very busy. The workload management software layer provides the software for scheduling, launching, controlling, and monitoring multiple jobs submitted by multiple users on a Beowulf cluster.

A very common and convenient HPC workload management strategy is using a distributed resource manager (DRM). The DRM software is responsible for accepting job submissions, placing submitted jobs into a queue, managing the job queue, prioritizing and scheduling the execution of jobs, coordinating the availability of resources needed by the jobs, launching the jobs, monitoring the jobs as they run, and cleaning up after job completion. Workload management is one of the most challenging and frustrating aspects of managing a cluster, and in many instances can be very difficult to do equitably. Further, a DRM is burdened with efficiently weighing and mapping all job requirements and priorities to the available compute resources so that jobs are executed in a fair and timely fashion. So in theory, DRM's manage all submitted jobs, don't lose any of them (bad!) and they are fair about who gets access to cluster resources. Since they essentially function as the gateway between a job waiting in a scheduling queue and that job running on the cluster, DRMs are often referred to as "middleware." DRMs can be broken down into several parts.

A DRM's queue manager holds submitted jobs in a queue, until the compute resources requested to execute the job become available. While jobs sit in the queue, the job scheduler assesses job priorities, requests compute resources, resource availability, and usage policies. The scheduler has

the responsibility of deciding which job to run and when to run it in order to optimize resource utilization, job throughput, and local policy enforcement.

A DRM's resource manager continually reports the availability of compute resources to the job scheduler so that the scheduler can make job execution decisions. Once a job is selected for execution by the job scheduler, the resource manager copies the chosen application and any required data to the nodes and starts the application. If necessary, the resource manager can suspend, resume, and terminate a running job.

DRMs possess extensive monitoring capabilities that provide a means to account for users' jobs, resource utilization, account allocation, and the ability to build usage reports, often for billing purposes. The usage reports are also typically used to calculate overall system utilization and the efficacy of the DRM's scheduling policies.

Of course, it's the administrators who set the policies that dictate the DRM's actions and some DRMs are easier to use than others. Currently, the most popular workload management system in use in the HPC world and on Linux Beowulf clusters is the Portable Batch System (PBS).

A.2.3.1.1 Portable Batch System (PBS)

PBS was developed by Veridian under a NASA contract and initially released as OpenPBS, an open source project that lives at http://www.openpbs.org. After continual development, Veridian released a commercial version of PBS called PBSPro. Altair Engineering recently bought the rights to sell and support PBSPro from Veridian. Businesses are required to purchase a PBSPro license to receive the source code for internal use. PBSPro information can be found at http://www.pbspro.com. Despite its popularity, PBS is difficult to setup and use.

A.2.3.1.2 The Maui Scheduler

Many HPC sites using PBS have discovered that although it is relatively simple to realize satisfactory system resource usage, maximizing resource utilization has proven difficult. PBS and other resource managers sometimes have difficulty allocating resources, particularly in scenarios where the dynamic range in the resources requested vary widely. This does not tend to happen on systems running a single application, like render farms or simulation clusters, but does happen on a frequent basis in multi-user, multiple application production systems, such as those seen in research organizations. These system usage shortcomings have been traced to the inability of the PBS job scheduler to properly translate complex job scheduling policies into practice. The result is that average system utilization often drops dramatically as resource usage stalls in anticipation of running a specific resource-hungry job.

The Maui scheduler, created by folks at the Maui High Performance Computing Center, was designed to extend available DRM scheduling functionality like that in PBS. Maui is an advanced job scheduler that doesn't contain resource manager functionality, and therefore must be used in conjunction with a DRM such as OpenPBS. Maui was developed to seamlessly extend PBS functionality by directly interacting with the PBS job scheduler, and as a result, its presence is typically transparent to the end user. The Maui Scheduler was developed with extensive feedback from users, administrators, and managers. At its core, it is a tool designed to truly manage resources and provide meaningful information about what is actually happening on the system. It was created to satisfy real-world needs of a batch system administrator as he or she tries to balance the needs of users, staff, and managers while trying to maintain his sanity. The open source Maui project lives at http://www.supercluster.org.

In a nutshell, Maui is an advanced batch scheduler with a large feature set well suited for high performance computing (HPC) platforms. It uses aggressive scheduling policies to optimize resource utilization and minimize job response time. It simultaneously provides extensive administrative control over resources and workload allowing a high degree of configuration in the areas of job prioritization, scheduling, allocation, fairness, fair-share, and reservation policies. Maui also provides statistics collection and profiling tools that allow administrators to test various configurations by simulation, that is, without actually having to implement them on a cluster. Maui's quality of service (QOS) mechanism allows directed delivery of resources and services, policy exemption, and controlled access to special features. Maui also possesses a very advanced reservation infrastructure allowing sites to control exactly when, how, and by whom resources are used.

A.2.3.1.3 Sun Grid Engine

Sun Microsystems acquired Gridware and released the Sun Grid Engine (SGE) as a free downloadable binary for Solaris™ and Linux in 2000. The Grid Engine project (http://gridengine.sunsource.net/) is an open source community effort sponsored by Sun Microsystems to facilitate the adoption of distributed computing solutions. SGE is the Distributed Resource Manager (DRM) of choice for Callident Rx.

A.3 Parallel Communication Methods

A.3.1 Message Passing Interface (MPI)

While other message passing libraries exist, MPI and PVM are by far the most frequently used when parallel programming for Beowulf clusters.

MPI isn't a computer language, but instead a specification for a library of subroutines that can be called by programs written in C, C++, or Fortran. The communication subroutines and their arguments have been specified by the MPI Forum, a consortium comprised mainly of parallel computer vendors and computer scientists. MPICH is one of the most popular implementations of that standard created by computer scientists at Argonne National Laboratory. More information on MPI, MPICH, and the MPI Forum can be found at http://www- unix.mcs.anl.gov/mpi/ and http://www.mpi-forum.org. *Using MPI* by Gropp, Lusk, and Skjellum and *Parallel Programming with MPI* by Peter Pacheco are both excellent MPI references.

A.3.1.1 Parallel Virtual Machine (PVM)

PVM was created at Oak Ridge National Laboratory, and PVM preceded MPI historically. As a distributed programming project that focused on computing in a heterogeneous environment, PVM was pivotal in the success of the Beowulf project. Currently, PVM development is supported by collaboration between Oak Ridge National Laboratory, the University of Tennessee, and Emory University. The PVM homepage is located at http://www.csm.ornl.gov/pvm/pvm_home.html.

Index

D

E

F

G

H

I