BOISE STATE AUTOMATED CLUSTER INSTALLER UPGRADE

by

John D. Prestwich

A project

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Computer Science

Boise State University

May 2009

This project presented by John D. Prestwich entitled *BOISE STATE AUTOMATED CLUSTER INSTALLER UPGRADE* is hereby approved:

Amit Jain Advisor	Date
Teresa Cole Committee Member	Date
Michael Stark Committee Member	Date
John R. Pelton Dean of the Graduate College	Date

dedicated to my wife Cathy, who kept believing even when I doubted.

ACKNOWLEDGMENTS

The author wishes to express gratitude to his family for sacrificing in many ways so that he could focus on his studies. To his committee members Dr. Teresa Cole and Dr. Michael Stark for their willingness to be on his committee and for being very flexible with scheduling. To Dr. Sin Ming Loo for allowing him to explore. And ultimately to Dr. Amit Jain for this project and for being very patient.

ABSTRACT

A high performance computing (HPC) cluster is a group of individual compute nodes linked together, via hardware and software, for the purpose of acting in unison to solve large scale computing problems. Although a relatively young technology, HPC clusters have proven to be highly useful in many applications. Because HPC cluster construction and maintenance span many technologies, they can be rather difficult to create, making it inconvenient for the novice wishing to have small cluster for exploration. Even the most seasoned administrator will find it a daunting task to create a cluster from scratch because of all the labor that is involved deploying it across a network. The Boise State Automated Cluster Installer (BSACI) was created to address these issues. Devised by Dr. Amit Jain and Paul Kreiner it is simple enough that a student with a reasonable understanding of computers and programming can expect to have a working cluster within a couple of hours. For the more experienced user BSACI is a ready made and tested HPC suite with a full set of support libraries. This suite deploys automatically across clusters of both small and large scale with little effort. Like all software, BSACI needs to be updated periodically to keep it abreast of the latest technology. This report considers the task of updating the installer by providing an overview of the various components of BSACI and how they interact. A specific approach to upgrading the installation tool is also given, illustrated with examples from the Fedora Core 5 to Fedora 7 upgrade.

TABLE OF CONTENTS

ABSTRACT vi

LIST OF FIGURES ix

LIST OF ABBREVIATIONS x

1 Introduction 1

1.1 History 1

1.2 The need for BSACI 2

1.3 The Need to Stay Current 4

1.4 Project Deliverables 6

2 The Boise State Cluster 8

2.1 General Description 8

2.2 Master Node 11

2.3 Slave Nodes 13

3 Boise State Automated Cluster Installer 14

3.1 YACI 14

3.1.1 The YACI Package 14

3.1.2 Using YACI to do an Install 17

3.2 BSACI 20

3.2.1 Setting up an Installion 21

3.2.2 Web-Based Application 23

4 Putting It All (Back) Together 25

4.1 Beginning an Upgrade to a New Operating System 25

4.1.1 Agile Test Platform 25

4.1.2 Managing Source Material 28

4.2 Package Upgrading 30

4.2.1 Slave Node 32

4.2.2 Master Node 37

4.3 Testing 47

5 Conclusions 50

REFERENCES 52

A RPM Sorting Scripts 55

A.1 extractPackName.bash 55

A.2 findNewRev.bash 56

B Master Node RPM Install Script 58

LIST OF FIGURES

2.1 Example topology similar to Boise States Beowulf cluster. 9

3.1 BSACI/YACI Relationship 15

LIST OF ABBREVIATIONS

BEOSH – Beowulf Cluster Shell

BSACI – Boise State Automated Cluster Installer

CLI – Command-Line Interface

COTS – Commodity Off The Shelf

GUI – Graphical User Interface

GFarm – Grid Data Farm

HDF(5) – Hierarchical Data Format Library (Version 5)

HPC – High Performance Computing

LAM – Local Area Multicomputer (An older implementation of the MPI standard)

MPI – Message Passing Interface

MPICH2 – Is an open sourced implementation of the MPI standard.

NFS – Network File System

NIC – Network Interface Card

OS – Operating System

PDSH – Parallel Distributed Shell

PRAND – Parallel Random Number Generator

PTK – Parallel Tool Kit

PVFS2 – Parallel Virtual File System, version 2

PVM – Parallel Virtual Machine message passing library

PXE – Preboot eXecution Environment

PBS – Portable Batch System

RPM – Red Hat Package Manager

SVN – Subversion Version Control System

TFTP – Trivial File Transport Protocol

YACI – Yet Another Cluster Installer

CHAPTER 1

INTRODUCTION

1.1 History

The Boise State Automated Cluster Installer [1] (BSACI) was devised to ease the work of installing the necessary software on both master and slave nodes of a high performance computing (HPC) cluster. Its main objectives were, first, to make an installer that approached the ease and convenience offered by other modern software installation tools, i.e., graphical user interface (GUI), easy to select configuration options, help menus, etc. A second objective was to have it based entirely on open source packages. Other cluster installation tools were available at the time of conception of BSACI and several were considered for this project. Many were proprietary solutions, which excluded them from the choice. Others like OSCAR [2] tended to not be as automated as hoped for, in that they required many "pre-installation" steps. Ultimately, the original designer, Paul Kreiner, found a lightweight installation tool called YACI (Yet Another Cluster Installer). YACI was not by any means a simple tool to understand, making it seem an unlikely choice at first. What it did provide, however, was flexibility and extensibility. All setup and configuration for an installation was realized via text files, thus making it easy to perform the setup and configuration automatically. With most of the work being done by scripts (BASH with some Perl), it was possible to see exactly how the installation was accomplished.

It also provided the ability to modify the installation process. More scripts were added that installed YACI and copied the packages required for the cluster to the master node. With the additional enhancement of a Web-based GUI, BSACI was born.

1.2 The need for BSACI

The claim of BSACI is that its eases the burden of creating a cluster for both the inexperienced and experienced user alike. For the novice (i.e., the student or other would be experimenter) BSACI provides a versatile set of preconfigured HPC packages, which allows them to have a working cluster without having to know how to configure each of the many packages needed to build a complete system. For the expert BSACI is a time saving tool, which allows the installation of a large number of nodes with little effort. It also provides a template from which the expert user can start and build his own custom solution.

One of the distinguishing features of BSACI is it ease of use. In many ways it is as easy to install the Boise State Cluster package as it is to do a standard operating system (OS) install. This "out of the box" feature saves the beginner weeks, if not months, of time by not having to learn how to create a cluster from scratch. It also provides a proven configuration that is known to work well so that the student or would be experimenter can concentrate on application development and not on system maintenance.

For the professional cluster administrator BSACI saves time by automating the process of distributing the operating system to the many nodes in a cluster. Anyone who has ever installed any type of operating system on just a single machine knows

that this can be time consuming. A cluster, on the other hand, is by no means a typical installation, generally requiring extra setup beyond that of the typical installation. This custom configuration will be required, not only on the master node, but on the slave nodes as well, meaning that each node in the system will require individual set up. The Boise State beowulf cluster, which has over 64 nodes, is probably a fairly typical example of the type of research cluster this package was created for and also provides a clear example of why automation is desirable.

BSACI does away with the burden of having to manually configure each node by allowing all of the slave nodes to be installed over the network. The concept of installing an OS on multiple computers simultaneously via a local area network is not a new one, in fact it is common enough that it is covered in the *Fedora 7 Installation Guide* [3]. Still, it is a cumbersome process and not necessarily automatic in and of itself. To aid this process BSACI provides a custom slave node image which is preloaded with all of the packages necessary for a cluster to function while eliminating many of the unnecessary ones that come with a typical installation. It also automatically configures the slave nodes at install time, eliminating the need to do this by hand, and also eliminating many of the human induced errors that accompany this process.

Because BSACI is open source it is possible to see how it works and also gives the user the ability to modify it. This provides a great deal of flexibility to anyone who is desirous to customize their installation. Not only is it possible to select which packages to install, but also add new packages that may not already be included. Since the source code for the installer is accessible it is possible to customize the install process as well.

Clearly, BSACI can serve as a time saving aid for both the novice and the pro. Being open source it is very approachable from both a cost and learning perspective. Yet despite its inexpensive price tag it is built upon industrially proven applications and provides a serious tool for the professional researcher as well.

1.3 The Need to Stay Current

As with most modern software, the core packages that make up a "Boise State Cluster" are dependent upon an underlying operating system and libraries. In fact, several of these packages, such as `mpich2`, need to be compiled specifically for the OS they are intended to be used with. In addition to installing packages that are specifally intended for use on a cluster, the Boise State Cluster Installer (BSACI) also has the task of distributing and installing an OS image on each slave node. Initial development was undertaken with Fedora Core 3 as the operating system packaged with BSACI, but by the time initial development was complete it had migrated to Fedora Core 5, demonstrating even in its genesis that updates would always be looming. Fedora has an aggressive release schedule, producing a new release about every 6 months with support for a given release provided only through the next two releases, or about 13 months [4]. Trying to keep pace would be a daunting task. Of course, it is not necessary to provide a new version of BSACI compatible with every new release of Fedora that is produced. Also, other Linux based options have emerged, such as CentOS, which have longer life cycles[1] (new releases every 2 years [5]). Although these last two options lengthen the period between updates, it does not eliminate the need for them.

[1]Future releases of BSACI will be based off of CentOS for the very reasons mentioned above.

One metric of whether a project is successful or not is its ability to stand the test of time. For a project to do this it must remain viable for the long term, or in other words, it must have long term maintainability. Software maintenance is probably considered one of the least glamorous aspects of software engineering, but the care given to this task can, in many ways, determine the overall success of a project. According to *Practices of Software Maintenance* [6], an article where maintainers of large software systems from several industrial sites were interviewed, there were certain *truths* that emerged about the maintenance of these systems. Among these were, first, that maintainers are (must be) experts in the systems they maintain; second, the source code is generally considered the primary source of information; and third, other documentation is useful but not generally considered as reliable as the source code itself. Future maintainers of BSACI will find that these truths ring true for this project as well.

Having stated the above, it quickly becomes evident that this document will unavoidably fall into the area of the third truth, meaning that it will be useful as an entry point to this project, but that true expertise will probably only come from delving into the source code itself. Still, it is hoped that as an entry point it can save the next generation of would-be BSACI experts some time on their journey to becoming such. Becoming an expert in BSACI means that a maintainer must have a good working knowledge of the various components that make up a HPC system. This involves not only packages like `mpich2` and `PVFS2`, which make up the end product, but also the packages needed to support them. This means a working knowledge is needed that stretches from system libraries to networking and the underlying operating system, including the kernel itself. To help in this area,

some of the more salient features of a cluster and its related components, and how they relate to BSACI, will be covered in more detail in Chapter 2.

In addition to having a good working understanding of the HPC system itself, a competent maintainer must also have an in-depth understanding of the tools used to do the actual installation. At the center of the installer is YACI. To fully understand BSACI it is essential to understand YACI and its various components. Once YACI is understood then the additional components added by BSACI become much more comprehensible and the larger picture of what BSACI actually provides becomes apparent. To help with this, the YACI installer and its related components will be covered in more detail in Chapter 3.

With an understanding of the Boise State Cluster package and the tools used to install it, one is ready to undertake the task of updating BSACI. As with most projects of any size, this can be a process filled with unseen obstacles. Chapter 4 strives to provide a road map of this process and also relates the lessons learned from the Fedora 5 to Fedora 7 update. The three previously mentioned chapters are meant to serve as a "quick start guide" to BSACI and are to be used in conjunction with the original cluster installer documentation [1].

1.4 Project Deliverables

Ultimately, what all the previous prose leads to, is that BSACI needs an upgrade. This project is concerned in particular with moving BSACI from Fedora Core 5 to Fedora 7. The finished product is a CD or DVD image of BSACI designed to install the suite of Boise State cluster packages on a freshly-installed Fedora 7 master node.

This software image will be provided in ISO9660 format, will be targeted for x86 (32-bit) architectures, and will include the following:

- updated automated cluster installation and configuration scripts,
- updated graphical interface,
- updated cluster specific packages tailored to the Fedora 7 operating system.

CHAPTER 2

THE BOISE STATE CLUSTER

2.1 General Description

The ultimate goal of the BSACI project is to have a properly configured and fully functioning cluster when it completes an installation. In order to know if that goal is met, it is useful to have a definition of what a properly functioning cluster is. The most common definition of a cluster is a group of linked computers, working together closely so that in many respects they form a single computer [28]. This definition holds true for the Boise State Cluster package, which focuses on high performance parallel computing and high volume (high bandwidth) data storage and retrieval. Starting with these applications it is possible to work backward and delineate the other components, both software and hardware, needed to realize a system that meets these needs.

Inherent in any definition is that the various nodes of the cluster must be able to communicate with each other. Network topology is a branch of study in itself, so no attempt will be made in this report to discuss the various strengths and weaknesses of different network layouts. It is instructive, however, to mention some of the typical aspects that are associated with a cluster of the sort we are targeting. Figure 2.1 shows a cluster arrangement similar to those used in the Boise State Beowulf [25] and student labs. This setup is flexible in that it can scale reasonably well from a

few slave nodes to a hundred or more. The master node is the "control center" and typically the only node with a monitor, keyboard, and mouse. It also, in most cases, provides the only network connection to the outside world for the entire cluster.

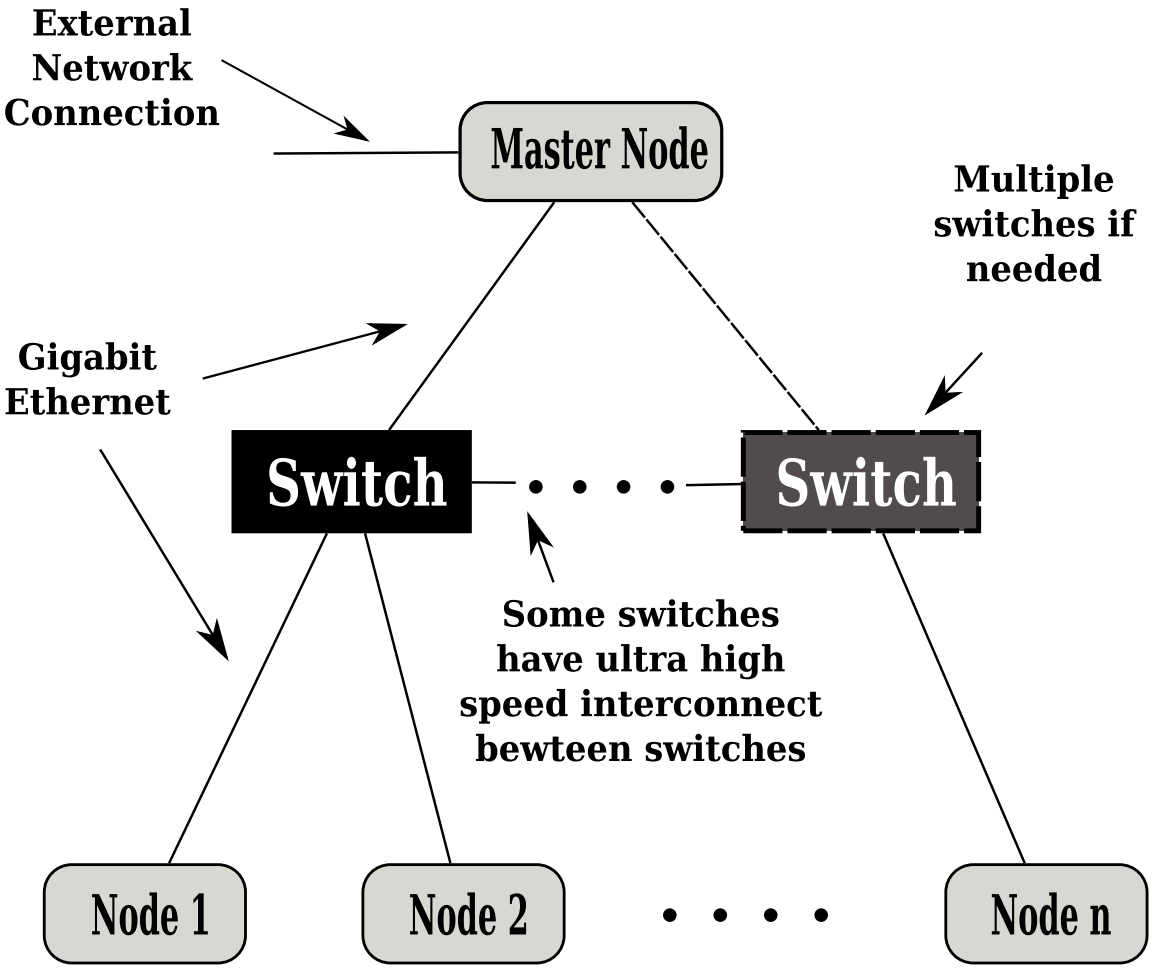

Figure 2.1: Example topology similar to Boise States Beowulf cluster.

User input to the slave nodes, during normal operation, is only provided via the network connection from the master node. Slave nodes do not have direct access to the outside world but are required to access it through the master node. Generally, slave nodes need only communicate between the various nodes of the cluster, so not

having direct access to an external connection is not much of a limitation. Also, since slave nodes generally only need external access for updates, and these updates will be applied in the same manner to each slave node, the bandwidth limitation of all slave nodes going through one node can be mitigated by doing a single download to the master node and then distributing the updates from there to the slaves.

The cluster designer is looking to maximize the bandwidth of the cluster, while keeping its overall cost to a minimum. The choice of what type of network switch(es) used can have dramatic effect in these areas. Switch technology is changing rapidly so any discussion of optimum setup quickly becomes obsolete, and is again beyond the scope of this report.

Much focus has been placed on using commodity off the shelf (COTS) components for the compute nodes [26, 27, 7]. At the time of this report, the popular choice is a dual core x86 processor with gigabit ethernet network interface card (NIC), due to the low price and relatively high performance. However, with the price of x86-64 workstations and 10 gigabit networking equipment decreasing, this is likely to change soon. The master will likely be equipped with two or more NICs: one will be used for external network access as mentioned above; the other(s) will be connected to the cluster through the switch(es). Often it is desired to have more than one NIC going from the master to the cluster to improve bandwidth and to prevent the master node from becoming a bottleneck.

Since software was the major focus of this project, the remaining sections of this chapter will be devoted to quantifying what is needed to make a functioning cluster.

2.2 Master Node

As noted above, the master node is the control center for the entire cluster and thus its software requirements are different from the other nodes in the system. While slave nodes are scaled down to a minimum set of packages, the master node receives a "fuller" set of the OS and other support packages. In many ways, the initial master node installation will resemble that of a typical workstation, i.e., X-server with desktop, standard command line utilities (bash, sed, awk, etc.), programming packages (gcc, g++, Perl, TCL, etc.), data analysis tools, etc. With Fedora 7 this base installation can be realized by using the standard installation DVD and selecting at a minimum, *Office and Productivity* and *Software Development* from the general package selection.

The additional packages, which provide the HPC functionality mentioned above, can be broken into the following subcategories.

- Message Passing Interface Libraries (MPI).
  - MPICH2 [8]
  - Open MPI [9]
  - Local Area Multicomputer (LAM) [10]
  - Parallel Virtual Machine (PVM) [11]
- Support libraries for MPI programming.
  - Parallel Toolkit Library (PTK) [12]
  - Global Arrays (GA) [13]
  - Hierarchical Data Format (HDF5) [14]
  - SZIP [15]
  - PRAND (Parallel Random Number Generator) [16]
- Support and management tools.

  - Oscar Modules [17]
  - Portable Batch System (PBS) [18]
  - The Cluster Monitor (ClusMon) [19]
- Parallel and distributed remote shell clients.
  - Parallel Distributed Shell (PDSH) [20]
  - Distributed Cluster Shell (BEOSH) [21, 22]
- Parallel and/or distributed file systems.
  - Parallel Virtual Filesystem (PVFS2) [23]
  - GFarm (Grid Data Farm) [24]

As mentioned in Chapter 1, the closer to the source code, generally the more accurate the information. With this in mind, it is recommended to go to the individual projects that support these packages for the most up-to-date information, since most of them are supported by very active open source projects and are generally well documented. Another good source of information is the research notes and package descriptions in the original thesis on BSACI [1].

For convenience in executing various parallel and distributed applications, the following three NFS (Network File System) mount points are made available on the master node: `/usr/local`, `/opt`, and `/home`. Allowing the slave nodes to mount these three points simplifies the execution of these applications across the cluster by providing one source for all applications and libraries. Having only one source of applications and libraries also aids in maintenance of the system, in that only the master node needs to be updated if newer revisions or patches are to be installed. For this scheme to work, however, the cluster must be made up of relatively homogeneous compute nodes. By "homogeneous" it is meant that the compute nodes all have similar type processors and are using the the same OS (flavor and revision).

With this in mind, the model followed by MPI type applications on the Boise State cluster has all applications developed and executed from a user directory under the `/home` directory. All libraries, scripts, and binaries needed to execute these applications are (or should be) located in either the `/usr/local` or `/opt` directory.

2.3 Slave Nodes

The slave nodes have a minimal install of the same operating system as the master node. Since these nodes will not have normal input and output devices connected to them while in normal operation, many of the GUI packages of the typical installation such as Desktop, etc., have been removed. Only basic X server functionality is preserved for remote graphical debugging. The requirement of the node image is to provide the necessary utilities, libraries, and networking support to allow for efficient cooperation with the master node. While this minimal install involves roughly half the number of packages that are needed (for the master node) it is still a substantial amount of software, totaling over 600 RPM packages with over 86,000 files. Since this minimal install set is unique to the Boise State cluster it is incumbent upon the maintainer of BSACI to recreate this set each time it is desired to move to a new OS. If the move is just from one revision to another of the same OS, many of the packages will continue to carry the same name from one revision to the next, and thus are relatively easy to update. However, even when the move is just from one revision to the next there will most likely be some repackaging needed.

CHAPTER 3

BOISE STATE AUTOMATED CLUSTER INSTALLER

3.1 YACI

At the heart of BSACI is YACI (Yet Another Cluster Installer). YACI is a lightweight system management tool that allows the user to quickly install large-scale (and small-scale) Linux clusters. Included in the tool set are an `initrd` for booting the nodes, `pxelinux` to actually perform the boot, and a collection of scripts to automate the installation process [29]. To put it another way, YACI provides tools to create installation images[1] and the means to transfer and install them to the various nodes. Though YACI is a very capable tool, it is rather lightly documented and somewhat cumbersome to configure, which is part of the reason BSACI was developed [1]. As is illustrated in Figure 3.1, BSACI adds additional components, which eliminates much of the guess work of configuring YACI.

3.1.1 The YACI Package

Since YACI is so central to BSACI it is necessary to understand its nuances to fully grasp the intricacies of BSACI. A good way of doing this is to examine how YACI would be used in a "standalone fashion." The YACI project comes bundled

[1]The term image is used loosly here. The file created by YACI is not a complete sector-by-sector copy of the file system intended for the slave nodes but rather a tarball of this file system.

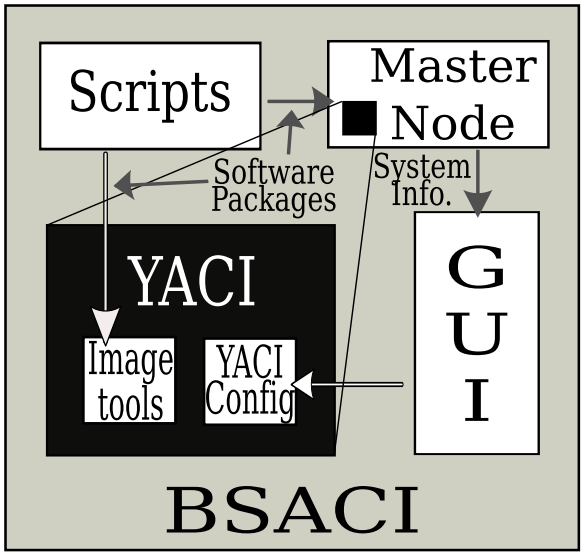

Figure 3.1: BSACI/YACI Relationship

in a RPM [30] (Red Hat Package Manager) package with three different versions (standard, ramdisk, and src). More about what these different versions provide is discussed below. When YACI is first unpackaged, most of the files are installed in the `/tftpboot` directory since it uses TFTP (Trivial File Transport Protocol) to distribute the image to each node. What documentation there is, is copied to the `/usr/share/doc/yaci-<version>` folder. The following is a listing of the `/tftpboot` folder just after the install of the standard RPM (as of the 12-7 version).

```
$ ls tftpboot -l
total 5944
drwxr-xr-x 2 root root    4096 2008-12-30 23:26 elilo
drwxr-xr-x 2 root root    4096 2008-12-30 23:26 etc
drwxr-xr-x 2 root root    4096 2008-08-02 14:08 images
drwxr-xr-x 2 root root    4096 2008-12-30 23:26 local
drwxr-xr-x 2 root root    4096 2008-08-02 14:08 log
-r-xr-xr-x 1 root root   11822 2008-08-02 14:08 pxelinux.0
```

```
drwxr-xr-x 2 root root    4096 2008-12-30 23:26 pxelinux.cfg
drwxr-xr-x 4 root root    4096 2008-12-30 23:26 ramdisk
drwxr-xr-x 2 root root    4096 2008-08-02 14:08 rpms
drwxr-xr-x 3 root root    4096 2008-12-30 23:26 scripts
drwxr-xr-x 2 root root    4096 2008-08-02 14:08 tarfiles
drwxr-xr-x 2 root root    4096 2008-12-30 23:26 tools
-r-xr-xr-x 1 root root 3063384 2008-08-02 14:08 vmlinuz.i686
-r-xr-xr-x 1 root root 2896884 2008-08-02 14:08 vmlinuz.x86_64
```

YACI uses pxelinux to PXE boot each node, which necessitates the use of a custom kernel for each type of node in the cluster. The standard version of the YACI package comes with two kernels: `vmlinuz.i686` and `vmlinuz.x86_64`. The first is targeted to the x86 family of processors and the latter to x86-64 processors. In most cases these pre-compiled images work as long as the kernel version matches the kernel version of the operating system being installed on the nodes. If a version of YACI that matches a particular kernel version cannot be found, there are two options available. The first, and probably the preferred, is to request a new version from the YACI maintainers. The second is to create a customized YACI package with a new kernel using the "src" version of the YACI package. More information on the second option will be given in Section 4.2.2.

As part of the installation process YACI will create an initrd for the kernel, which is PXE booted by each slave node. This ram disk is created from the standard utility programs, modules, and configuration files of the master node. Since an initrd based on the packages on the master node is not always be appropriate, the automatic building of the ram disk can be turned off by setting the `BUILD_RAMDISK` field in the `/tftpboot/scripts/variables` file to "NO." A generic initrd file is provided in the "ramdisk" version of the YACI package, which provides most of the standard functionality.

3.1.2 Using YACI to do an Install

Below is an example of the steps that would be used if YACI were being manually configured and deployed. Note that the following steps presume TFTP and NFS servers have been installed on the master node and that all the slave nodes have PXE (Preboot eXecution Environment) capable network cards.

1. Determine the different types of node images needed. (Although BSACI is currently set up to create and use just one image, YACI does have the capability to create and distribute multiple images. This feature might be useful in the future in that one installer could be set up to create and deploy images to a mixed set of 32-bit and 64-bit processor nodes.)

2. Determine operating system to be installed on the various nodes and obtain necessary RPMs. Place these RPMs into the `/tftpboot/rpms` directory on the master node.

3. Set up a `partition_list.NODE_TYPE` file for every type of image wanted in the `/tftpboot/etc` directory.

4. Determine files from master node to be included in node image(s).

5. Create copies of all system configuration files to be included in the slave node image in the `/tftpboot/local` directory. Alternatively, files to be copied can be listed in the `/tftpboot/local/localize` file. The latter is actually the newer and preferred method of doing this.

6. Create (if desired) a "post create" script to be executed at the end of the `/tftpboot/scripts/create_image` script.

7. Use the `/tftpboot/scripts/create_image` script to create node images and initrd ram disk.

8. Create (if desired) a "post-install" script to be executed at the end of the `node_install` script when it finishes installing an image on a node.

9. Obtain the MAC (Media Access Control) address of each node and create a `MAC.info` file in the `/tftpboot/etc` directory specifying node name MAC address, and image type for each node.

10. Create a subnet listing for the TFTP server in `/etc/dhcpd.conf` and add a host listing for all nodes.

11. Create an entry in the `/etc/exports` file for the `/tftpboot directory`.

12. Start the TFTP and NFS servers.

13. Reboot each slave node in cluster.

When each slave node reboots it will PXE boot the default kernel and initrd as determined by the `/tftpboot/pxelinux.cfg/default` configuration file on the master node. In the case where different kernels are needed by different nodes, multiple configuration files can be created, allowing custom boot configurations to be defined on a node by node basis or by IP address range (see the pxelinux documentation for more details [31]). When the kernel and initrd are loaded the `/etc/init.d/node_install` script will mount the `/tftpboot` directory on the master node and execute the `scripts/node_install` script. Note that there are two scripts called `node_install`. The first is part of the initrd ram disk downloaded to each node. Its main purpose

is to mount the master node via NFS. The second is on the master node under `/tftpboot/scripts` but is executed on each node via NFS file sharing.

The second `node_install` script (i.e., the one under `/tftpboot/scripts` on the master node) does the work of reformatting the hard drive of each node and installing the new image. The output of this script is sent to a file under the `/tftpboot/log` directory on the master node. The filename is `node_name.log` where `node_name` corresponds to the node from which the output was generated. These log files are invaluable when debugging installation problems. The following is an overview of what the `node_install` script performs.

1. Blank and partion hard disk.
2. Format partitions.
3. Transfer image to node and untar onto hard drive.
4. Populate the `/dev` directory.
5. Copy over the `/etc/fstab` and create a new `/etc/modules.conf`.
6. Configure the boot partition.
7. Set date and time.
8. Run post-install script.

When `node_install` is finished with the install it creates a text file under the `/tftpboot/pxelinux.cfg` directory, which is named after the IP address of the node (written as a single hexadecimal number). This file is created to prevent the node from PXE booting from the master node after it is rebooted by the `node_install` script.

3.2 BSACI

Referring back to Figure 3.1, there are three ways BSACI assists YACI: it installs and configures the cluster specific packages on the master node, it provides the packages to build the slave node image, and sets up the configuration files under YACI for the image build process. A typical install has the following flow:

1. The user does a clean install of new OS.

2. The cluster installation is started by executing the `RUNME.sh` script from the root directory of the install CD. It sets up some environment variables and then calls `scripts/MASTER-SCRIPT-1.sh`.

3. The `scripts/MASTER-SCRIPT-1.sh` script installs YACI, the packages for creating the slave node image, and HPC packages mentioned in Section 2.2 onto the master node.

4. After `MASTER-SCRIPT-1.sh` completes, it returns control to the `RUNME.sh`, which then prompts the user concerning whether to do a Web-based (GUI) or CLI (command-line interface) installation of the slave nodes. The difference between these two paths is that the CLI requires the hand editing of the BSACI configuration file (which is placed in the `/tmp` directory on the master node during the installation) to enter network interface and slave node partition information. The Web-based path prompts the user for this information and then enters the information into the configuration file.

5. Following either path, after the network and partition information is entered, a series of scripts are called, which set up the configuration files under YACI so

that it builds the cluster as desired. BSACI will then prompt the user to reboot all slave nodes so that it can capture their MAC addresses. At the same time it will build the slave node image in the background.

6. When the MAC addresses are captured and the slave node image is complete, BSACI will then have the user reboot each slave node. As the slave nodes reboot, they PXE boot the YACI kernel as described in Section 3.1.2. BSACI tracks which nodes have completed their installation and then prompts the user to reboot the master node when all slave nodes are finished.

3.2.1 Setting up an Installion

One of the advanced features YACI offers is the ability to customize the image created for the slave nodes. This allows the user to preconfigure the slave node image with the desired settings before it is installed, thus eliminating the need to individually configure each node after installation. Even with this time saving feature there is still a substantial amount of configuration that must be done. Much of this setup requires specific system information that must be gathered through the use of various system level utilities and requires a reasonably advanced understanding of the underlying OS. Fortunately, BSACI comes to the rescue by automating this task, not only saving the user time, but making it possible for someone with less advanced system level knowledge to do a complete installion. A bonus feature is that it also reduces the probability of human induced error.

The task of gathering information and then using it to set up the various configuration files is delegated to a set of bash scripts found in the `scripts/` directory under the root directory of the installation CD. These scripts can be categorized into

several different groups depending on their functionality and the order in which they run. The main functionality classifications are *probe*, *configure*, and *do* scripts and each one can be easily identified by its name, i.e., `probe-*`, `config*-`, and `do_*` [1]. As the name implies, the `probe-*` scripts have the task of collecting master node system information. These scripts are deployed by the Web-based install application so that it can provide a selection of parameters to choose from in its setup stage. The information gathered by probing is stored in a master configuration file where it is later retrieved by the Web-base application and ultimately by the `config-*` and `do_*` scripts.

The last two classifications of scripts are aimed at either creating or modifying various system level configuration files both on the master node and in the slave node image. The distinction between a `config-*` and `do_*` scripts is that `config-*` scripts are generally focused on setting up a particular file, while the `do_*` scripts can be a little more involved (for example, they may copy or edit multiple files). Also, `config*-` scripts are deployed first, although this ordering does appear to be somewhat arbitrary and probably could be changed if the need arises.

The other broad categorization of these scripts is whether or not they need to have knowledge about the nodes in the system (i.e., the number of nodes, node naming conventions, network address, etc.). Since collecting node information requires manual input, i.e., rebooting each node to collect MAC addresses, it is one of the slower portions of the install. In order to make the most efficient use of time and resources, the gathering of MAC addresses is done in parallel with the creation of the slave node image, since one is I/O and the other is processor bound. Since creating the image requires that the setup to all files that will be encapsulated in the tarball image be completed before building commences, the "needs node knowledge" distinction is

made. To simplify the deployment of these scripts at the various stages, four staging ground scripts were created, namely

- `quickie--all-config.1.sh`
- `quickie--almostall-do.1.sh`
- `quickie--all-config.2.sh`
- `quickie--almostall-do.2.sh`

They are deployed in the order given with the first two deploying before the capturing of MAC addresses and the last two after. At the top level these four scripts are called from a set of `MASTER-SCRIPT-*` scripts located in either the `scripts` directory on the installation CD, or the `htdocs` directory of the Web-based application, depending on the installation path followed.

3.2.2 Web-Based Application

The Web-based installation application was conceived as an easy method to produce a high quality GUI interface for the novice user. The application prompts for the applicable network and slave node partition information, in most cases providing default settings and/or a list of valid choices to choose from. A web server is not typically part of a default OS installion so BSACI provides its own Apache [33] web server for this task. The web server is installed under the `/tmp` directory on the master node with the understanding that it will be removed after the installation is complete. The application is written in a combination of PHP and Javascript. The application (web server and code) is located in the `srcs/` directory of the installation CD portion of the project in a tarball called `apache-overlay.tar`. It is copied from

the CD and untared by the `MASTER-SCRIPT-1.sh` script, which also starts the web server, so the web application can be accessed later.

CHAPTER 4

PUTTING IT ALL (BACK) TOGETHER

4.1 Beginning an Upgrade to a New Operating System

Some of the most daunting aspects of a project such as this have little to do with the development of the actual code that makes it function. With the BSACI upgrade project, most of these daunting tasks can be boiled down to the following two statements: developing an effective test platform, and managing the vast amount of source material in the project. While these are mostly logistical issues, they can greatly impede progress if good solutions are not found. Often these solutions become what is colloquially known in the technical trades as "tribal knowledge." In other words, they are passed on only by word of mouth and lost if an effective transfer from one maintainer to another is not made. Often they are taken for granted, because once they are known they seem *obvious*, hiding the fact that much time was spent in refining them.

4.1.1 Agile Test Platform

One of the greatest challenges to testing the installer is the amount of time that is required to restore the master node to a pristine starting point. If done as most users would be doing this, it would require installing (or reinstalling) the OS of choice from scratch. After the installation was complete, updates to the system would then have

to be downloaded and installed. These two tasks together can take an hour or more depending on hard drive size and download speed. Assuming that most maintainers (at least for the foreseeable future) are going to be students who most likely only have a couple of hours here and there in a day to devote to the project, this obviously makes for slow progress.

Paul Kreiner, in the original thesis on BSACI, recommends using a platform virtualization tool such as VMWare [34], which allows a snapshot of a virtual node to be taken. This snapshot can be used as a restore point to bring the node back to a pristine state in much less time than a full install. Virtualization tools also have the added advantage that multiple nodes can be set up on one computer, thus making a virtual cluster.

The down side of virtualization software is that although there are many open source solutions (including parts of VMWare itself), most of the advanced features like Snapshot tend to be commercial add-ons. Since the market for these tools is mostly focused on enterprise applications, full commercial licenses tend to be rather expensive, and the cost can be prohibitive. Also, one of the stated goals of BSACI is to look for open source solutions. Still, the advantages of virtualization are notable, so it is probably worth revisiting at the onset of a major upgrade to see if more features become open source in the future.

Luckily, virtualization tools are not the only resources for backing up partitions and hard drives. A quick search will reveal that there are a plethora of such utilities ranging from full commercial tools to freeware and open source tools. Many are geared only for Microsoft Windows and NTFS. Several, such as *SystemRescueCd* and *Ghost for Linux (G4L)*, are completely open sourced and support ext2 and ext3 file

systems. They also have the added advantage that they boot from a CD, making it much easier to back up the root file system.

Ultimately what proved to be the simplest solution was to use a live CD[1] [35, 36] and the `dd` command-line utility to make a backup image. The `dd` utility allows low level copying of raw data from files and block devices. Use of the utility is fairly simple with the following command line being all that is needed to initiate a backup of the `sda1` partition to a file called `root.back`.

```
dd if=/dev/sda1 conv=sync,noerror bs=4k of=/backup/root.back
```

Referring back to the original premise for doing the backup, i.e., restoring a pristine state in less time than a full installation requires, anyone who has ever done a full backup image of a hard drive knows that it is not exactly a quick operation. A few observations can help in this regard. Though it is tempting to pipe the output of `dd` through `gzip` or some other compression medium to create a smaller backup image, it is not recommended due to the fact that it greatly increases the time required to back up and restore. Not using compression has the complication of finding a secondary storage medium large enough to hold the entire root partition. Hard drive storage capacity is growing more rapidly than the standard Linux distribution, so this concern is likely to diminish in the near future.

Still, making a backup of the entire root partion, even without compression, can take enough time that the benefits gained over a full installation are minimal. With this in mind, it is useful to minimize the size of the root partition. One method of accomplishing this is to split off portions of the file system that are not greatly affected by an installation, to separate partitions. In the case of the Fedora 7 version

[1] A live CD or live DVD is a CD or DVD containing a bootable computer operating system. Live CDs are unique in that they have the ability to run a complete, modern operating system on a computer lacking mutable secondary storage, such as a hard disk drive [37].

of BSACI it was found that creating separate partitions for the `/tftpboot` (this is where the slave node image is created and takes up several gigabytes of storage) and `/tmp` directories allowed the root file system partition to be sized to 14 gigabytes. This provided enough room for the OS and cluster packages, plus some additional space to use while testing the final installation. With a root partition sized thus, it is possible to boot the live CD and restore the partition in 20 minutes or less, which amounts to at least a factor of 3 reduction in time for a full installation. It should be noted that a significant portion of that time is spent in booting the graphical interface of the live CD, so finding a command-line version of a live CD might produce even greater time savings.

4.1.2 Managing Source Material

All projects have some sort of management philosophy, which over time must be revisited to ensure that it meets the continually changing needs of the developers. Though it is often unspecified, the philosophy is apparent by the very layout of the project itself. When the Fedora 7 upgrade began it was obvious that one of the overarching concerns of the original project was the considerable amount of storage space required for all the binary packages needed to build the slave node. To work around this concern, only the installation scripts for BSACI had been placed under revision control, and not any of the binary packages. While there are a variety of reasons why this made sense[2], it ultimately lead to problems when reassembling the project at the beginning of this upgrade. As noted in the original BSACI writeup [1], one of the most time consuming aspects of an upgrade is collecting, filtering and

[2]For example, most software revision control systems are designed around the concept of archiving text source code and therefore are not all that efficient when it comes to storing binary files.

verifying these packages. It would be a devastating setback if once this task is complete these packages where lost due to hardware malfunction or human error.

Weighing all of these considerations, it was decided that with hard disk space becoming more abundant and the arguments for archiving the binary packages reasonable ones, it was time to include the binaries in the Subversion [38] (SVN) archive. The first attempt at doing this was to just replace all of the placeholder files with the actual files and then check them into the repository. This created the problem that most of the packages were stored in one large tar file (over 0.5 gigbytes with the Fedora 7 release), which meant that any time one package was changed, a new 0.5 gigabyte tar file would have to be checked in, consuming a great deal more space than is needed. To work around this issue a compromise solution was devised in which the large tar file was removed from the repository and replaced again with its corresponding placeholder file. A directory was added to the repository, outside of the CD portion of the project where the tar file could be untarred, thereby allowing easier access to all of the individual packages. Since building the tar file was a time consuming process in and of itself, a build script was created, which automated the process, not only saving time but also helping to reduce errors as well.

The above mentioned method of archiving the binaries files seems to work well. Later research has indicated that SVN, unlike many other revision control systems, actually does make some attempt to compress binary files through an algorithm that compares the difference between what is currently checked in and what is being checked in, so this new philosophy may not have been as necessary as originally thought. Checking in or deleting a single package is much faster than checking in the entire 0.5 gigabyte tarball so that alone justifies this tactic. In many ways, having all of the packages in an untarred format makes them easier to work with as well. Also,

the new tarball `make` file has proven to be a time saving device in that it allows the tarball image to be quickly rebuilt during the refining process of creating a new slave node image.

Another useful observation in regard to source maintenance is that ultimately BSACI is destined for a portable medium such as CD or DVD and as such can only be tested in a similar manner. With 2 gigabyte (and greater) USB thumb drives being reasonably priced these days, it is useful to download the SVN project directly to one of these, so that testing and updating can be done from the same source (it also cuts down on the number of CDs that will have to be burned). Depending on the size of the thumb drive some care may be needed in choosing which parts of the project to download. Luckily, SVN is very flexible in that each subdirectory in a project can be checked out on its own. Because SVN creates a backup copy of all files when a project is downloaded, the thumb drive needs to be at least twice the size of the actual source files to accommodate the checkout.

4.2 Package Upgrading

The majority of the software packages installed by BSACI are OS distribution RPMs intended for the slave node image. The remaining few are made up of the cluster specific packages mentioned in Section 2.2. Most of the software packages are in the RPM format, the rest are stored in compressed BZIP2 archives. During the final process of making an installation CD, all these packages are tarballed into one archive, called `tftpboot-distro-overlay.tar`, which replaces the placeholder file of the same name in the `srcs/` directory under the CD portion of the project. The main reason for single archive is that it transfers from the CD to the master node in less

time than copying all packages separately. No compression is used on the tarball for a couple of reasons. First, since all packages are either RPM or BZIP2 archives and are already in a compressed form, not much size reduction is gained by compressing the complete archive. Second, unpacking the archive takes considerably more time if compression is involved.

The untarred packages can be found in the

`development-tools/sources/Fedora_7/tftpboot-distro-overlay`

directory of the BSACI SVN repository. The contents of the `tftpboot-distro-overlay` directory is as follows:

```
drwxr-xr-x 7 root root 4096 2009-02-05 15:26 dist
-rw-r--r-- 1 root root  660 2009-02-05 15:26 Makefile
drwxr-xr-x 3 root root 4096 2009-02-05 15:17 rpms
drwxr-xr-x 3 root root 4096 2009-02-05 15:17 site-specific-local
-rwxr-xr-x 1 root root 1157 2009-02-05 15:26 updateRPMList_repos.sh
```

The `Makefile` and `updateRPMList_repos.sh` script are used to automate the tarball building process. The tarball will contain the three directories and all of their contents. The `dist` directory contains all of the packages mentioned above. The `rpms` directory contains a symbolic link to all of the packages that are to be included in the slave node image. The `site-specific-local` directory only contains one file, which is a list of all the RPMs to be included in the slave node image.

During installation, the tarball is untarred in the `/tftpboot` directory on the master node after YACI has been installed. The `rpms/` directory and the RPM list are required by YACI. The `dist/` directory was conceived at the inception of BSACI as a method for organizing the packages in some fashion. This is due to YACI not

being able to tolerate subdirectories in the `rpms/` directory. Rather than placing all packages for the slave node image in the `rpms/` directory with no organization to them, they were organized under the `dist/` directory with symbolic links to each of them being placed in the `rpms/` directory.

The Fedora 7 version of BSACI uses the same directory structure as in the Fedora Core 5 release, which essentially followed the naming conventions of the Fedora Core 5 repositories. The exception to the repository naming convention is the `others/` directory, which is intended for packages specific to a cluster installation. The `others/` directory has a subdirectory called `master/`, which holds packages only intended for the master node.

4.2.1 Slave Node

As mentioned in Section 2.3, the packages for the creation of the slave node image are a subset of what a normal workstation installation would be. Having the slave node built from a subset of a full distribution actually complicates the upgrade process in that the equivalent subset of packages must be extracted from each new distribution. There are two likely scenarios that need to be dealt with when doing an upgrade. In the first, and most likely scenario, BSACI is being moved from one revision of a particular distribution to a different revision of the same distribution. The second involves migrating BSACI to a completely different distribution. The Fedora 7 upgrade falls under the first scenario, so most of the observations made here regard revision changes, although some ideas will also be presented concerning the latter scenario.

As mentioned in Section 4.2, the naming convention for the directories containing the RPMs for the slave node image follow the naming conventions of the Fedora

Core 5 repositories: namely `base`, `updates`, and `extras`, with all non-Fedora RPMs being put in the `others` subdirectory. All RPMs from the various repositories go into the directory named after their repository of origin. Revisions after Fedora Core 5 actually have a slightly different repository naming convention in that the `extras` repository has been replaced by an `Everything` repository. Since the same basic structure still remains (i.e., `base` and `updates` repositories), it was therefore decided to use a convention similar to the previous revision of BSACI. The new convention is that all packages from the `base` and `updates` repositories are still placed in the directories corresponding to the repository of origin, and all other packages, which originated from Fedora, are placed in the `extras` directory.

Even though well over 90% of these RPMs had direct replacements from Fedora Core 5 to Fedora 7, with over 600 RPMs in the project it was still somewhat challenging to find the proper replacement for each RPM. The reason for this was that many packages had very similar file names like

```
hal-0.5.9-8.fc7.i386.rpm
hal-libs-0.5.9-8.fc7.i386.rpm
hal-cups-utils-0.6.9-1.fc7.i386.rpm
```

which made it somewhat difficult to do an automated sort strictly on file name. Still, with over 600 packages to find out of thousands, a manual solution was not feasible either. Luckily the `rpm` utility itself provides a nice feature called "queryformat" to help uniquely identify packages. The listings in Appendix A are for two short scripts developed during the course of this project. Both `extractPackName.bash` and `findNewRev.bash` scripts use the *queryformat* function to help pinpoint the exact name of a package. The `extractPackName.bash` script recursively searches through a directory and its subdirectories and extracts the name of all the RPM packages therein. The `findNewRev.bash` script can then use the list generated by the former

script to search a downloaded repository to find matching updates. Currently these scripts use the package name as the key but both could easily be modified to add other distinguishing features, such as architecture, revision, etc.

The procedure for finding the updates goes something like this:

1. Use `extractPackName.bash` and redirect its output to a file.

2. Since packages found in the `updates` repository have similarly named packages in other repositories, it is best to start with the `updates` repository when using `findNewRev.bash` to find matching updates. The reason is that packages in the `updates` repository should be the most up to date, and therefore if an update exists then that revision of the package is the one to include.

   The output of `findNewRev.bash` is formated

   <package name>,<upgrade file name>

   with each package on a separate line. If a direct replacement is not found for the package then the "upgrade file name" is replaced with "###" making it easy to filter the output (using grep, for example).

3. After performing the previous step and creating a list of packages not yet found, repeat the same process on the remaining list, sorting through the `base` repository and then any remaining repositories.

There is a reasonable chance that when the above process is complete, a few remaining packages will still not have a direct upgrade. The most likely reasons for this are that the package is not part of the OS release, or that the OS has discontinued the use of the package. For the first case, most packages produced for an OS release are

fairly clearly labeled. In the case of Fedora 7 a "`.fc7.`" is found in most file names. Unfortunately, this cannot be a sole determiner of whether a package was created for distribution with a particular OS. This is because many packages will use similar naming conventions to indicate that a package was built specific for a particular OS, even though the package is not directly maintained by the OS distributors. Another resource for determining the origin of a package is to use the query information option, i.e.,

```
rpm -qpi <package-file-name>
```

to see if the packager is listed. If it is determined to be a non-OS package, most likely it is cluster specific and its replacement, if one exists, will most likely be found at the Web site of the actual utility itself.

An example of the second case, which actually arose during the Fedora 7 upgrade, is the `portmap` package being replaced by `rpcbind`. If it is suspected that the package is no longer part of a distribution, it is always a good idea to first verify that the scripts did not make a mistake or that all packages in the repository were downloaded before sorting began. After verifying that the upgrade package truly does not exist, the challenge is to determine if and how to replace it. The Fedora release notes and Google are good resources for this task. The RPM package itself can also provide useful information through the use of the RPM query option (`rpm -q`).

Once an upgrade is found for each of the old packages, or it is determined that its functionality is no longer required, then comes the true test of whether the new set of RPMs will produce a functional image: build an actual image and install it on a node. An image is created by the `create_image` script found in the YACI `scripts/` directory. Since this subject is well covered in the original BSACI documentation [1] it will not be repeated here. One difference worth noting, however, is the use of the

`test_rpmlist` script rather than the `create_image` in the early stages of verifying whether all package dependencies are met, since this is not mentioned in the original documentation. The `test_rpmlist` script is a shortened version of `creat_image`, which does a mock build of a node file system, only verifying that the dependencies are met but not actually installing any packages. It is useful in that when first verifying the package set it is likely that several iterations will be needed before all dependencies are met. The `test_rpmlist` script is much quicker than `create_image` at this task.

Moving to another distribution of Linux will require an altered approach, since it will essentially require building a slave node image from scratch. This is due to the lack of a one-to-one relationship between the packages of the original and new distribution. An approach for changing distributions would be to do a normal install of the new distribution of choice and compare it to the installation of a node done with an existing BSACI project. To save some time, packages that are already known to be unnecessary (like desktops, desktop support, office, etc.) should not be installed.

Even though there will not be a direct one-to-one correspondence between packages from one distribution to another, the `rpm` utility can still provide some help. Most of the distributions derive their base package set from the same source code (e.g., nobody creates their own version of `awk`, they all use `gawk`) so even if the top-level package names do not line up, many of the underlying ones will. The following RPM command-line will provide a listing of all packages installed on a system, not just the top level names.

```
rpm -qa --provides | sort > RPMS.txt
```

This process still will likely require a good deal of manual labor but hopefully can serve as a starting point from which to begin. Other RPM querying options (i.e., `-qa` options) that might prove useful are `--whatprovides` and `--whatrequires`.

4.2.2 Master Node

Packages for the master node fall into two categories: those required for building the cluster (see Section 2.2), and those that are for temporary use during the installation process (e.g., YACI and the Web-based installion application).

Cluster Specific Packages

Packages specific to building a cluster are, as mentioned in Section 4.2, a part of the `tftpboot-distro-overlay.tar` archive and most are stored in the `others/master/` directory of the development section of the project. The packages are a mixture of RPM and BZIP2 archives. All packages that are in BZIP2 archives have been specifically compiled for this project either because there was not an up-to-date RPM version of it or because there was special configuration required. Some judgment will be needed to decide how best to upgrade these packages, or whether to update them at all. Below is a brief summary of the updates done for the Fedora 7 upgrade.

- **Message Passing Interface (MPI) Libraries**

  MPICH2 Updated to the 1.0.6 release of the code. Recompiled for Fedora 7.
  OPEN MPI Updated to 1.2.4 release of the code. Recompiled for Fedora 7.
  LAM Obtained the Fedora 7 release of the RPM packages.
  PVM Recompiled for Fedora 7.

- **Support libraries for MPI programming.**

  PTK Recompiled for Fedora 7 and new MPI libraries.

  Global Arrays Recompiled for Fedora 7 and new MPI libraries.

  HDF5 Recompiled for Fedora 7 and new MPI libraries.

  SZIP Recompiled for Fedora 7 and new MPI libraries.

  PRAND New addition to BASCI

- **Support and management tools.**

  Oscar modules No updates. Fixed paths in scripts to point to new MPI libraries.

  PBS Recompiled for Fedora 7.

  ClusMon Recompiled for Fedora 7.

- **Parallel and distributed remote shell clients.**

  PDSH Obtained the Fedora 7 source RPM package and recreated packages. (Needed to rebuild source because `pdsh-mod-machines-*.rpm`, which is needed by BSACI, is not part of the standard Fedora release).

  BEOSH Recompiled for Fedora 7.

- **Parallel and/or distributed file systems.**

  PVFS2 Updated to the 2.7.0 release of code. Compiled for Fedora 7 and MPICH2.

  GFarm Grid Filesystem No update.

With the exception of GFarm, there were no significant updates to the above mentioned packages. In the case of GFarm, it was decided not to update because the new release had just recently been introduced and appeared not to be in general use yet. Most updates to the above packages were done mainly for compatibility with Fedora 7. All packages that were based on the original source code were recompiled under Fedora to help ensure compatibility.

There are a few details concerning building the BZIP2 packages that are worth noting. As mentioned above, one of the reasons for compiling these packages from

the original source code is to perform special configuration. Fortunately, most of this special configuration has been captured in a set of configurations scripts in the `build-scripts` directory under the `development-tools` section of the project. Mostly this special setup is aimed either at relocating where the software ultimately is installed or the need to link various packages to libraries of other packages being installed on the cluster. Since there is an interdependence between some of these packages, the order in which they are built is important. In general, the MPI libraries should be built first, followed by other libraries (such as HDF5), which offer support to MPI applications. For more information on what packages are relocated and the various interdependencies between packages, it is best the study the actual build scripts.

In order for these packages to be archived into BZIP2 packages, they must be correctly installed after compiling. All of the code sets currently included have a `make install` option that allows this to be easily done. It is worth capturing the output of the `make install` step to both ensure that everything installed correctly and also verify where the various components are installed. Another resource of determining what files are needed by each package is to look at the BZIP2 packages of the previous revision. The archiving is done from the perspective of the root directory and can be done using the `tar` archiving utility.

The process used to install RPM packages was changed for the Fedora 7 release of BSACI. It was found that the method used in the Fedora 5 version of BSACI would erroneously not install some packages if certain packages in the base OS installation on the master node had changed. The problem occurred because the Fedora 5 version of BSACI did not provide any means for the user to deal with package conflicts. The most common conflict was that a newer version of a package was already installed.

Having a package not installed because a newer version was already installed was not a problem, but it would cause the entire group of packages (including the packages not having the conflict) to not install.

In an attempt to provide a more robust solution to the above mentioned problem, the `installMasterNodeRpms.pl` script was devised (See listing in Appendix B). It is designed to detect if the reason for package installation failure is because the package (or a newer version of it) is already installed, and it will simply skip the installation of the package while providing a message noting that the package was not installed. If `rpm` is not able to install a package for another reason, then the user is given the error message with the options to either retry the installation of the package with the current setting, force the installation of the package using one of, or a combination of, the `--force` or `--nodeps` options, or to skip the installation altogether. These options give the user several methods to work around potential installation problems without halting the installation itself. For instance, if the problem was due to a missing or out-of-date dependency, then the package meeting the dependency could be located, installed using a separate terminal window, and the "retry installation option" can be used to resume the installation.

The `installMasterNodeRpms.pl` script also allows the maintainer of BSACI to customize the installation options of each package. This feature was added because of the need to force `tcl-8.4` and `tcl-8.5` to coexist in the current version of BSACI, but it may also be useful in solving future problems.

The disadvantage of the `installMasterNodeRpms.pl` script is that currently all RPM packages that need to be installed on the master node have to be listed in a file called `packagesToInstallMasterNode.txt`. The list has to be ordered according to dependency needs, i.e., packages that require other packages in the list must be listed

after their dependency. The need to list the packages in order of dependencies could probably be worked around, but this is left for future work.

YACI

Although YACI has more to do with the installation of the slave node than that of the master node, the actual application itself resides on the master node and thus will be considered by this report a part of the master node installation.

After being installed by the `MASTER-SCRIPT-1.sh` script, YACI is patched to help customize it to the needs of BSACI. To be more specific, these patches fix some bugs, change the image from being a compressed tar file to a regular tar file[3], and add modifications to provide additional debug output during the installation process. Bug patches need to be evaluated from update to update to verify whether they are still needed or not. The other patches need to be checked to verify that they still apply cleanly to new revisions of the scripts.

The `tftpboot-script-overlay.tar` is then, as the name implies, overlaid, or added to, the YACI files in the `/tftboot/` directory. It contains a set of files for inclusion in the new slave node image when it is built. To help encode the path of where the file is to be installed in the slave node image, a "%" is used to denote a "/", e.g., the `bashrc` file located in the `/etc` directory is stored as `%etc%bashrc` in the `local` directory. These files will later be modified by a set of configuration scripts to provide system specific information obtained from the master node. These configuration scripts are described in more detail in Section 3.2.1.

[3]An uncompressed archive is used with BSACI to increase installation speed. With gigabit ethernet cards being standard on most cluster nodes these days, it is faster to upload an uncompressed file than to upload a compressed file and uncompress it.

An update to YACI will need to be considered whenever there is a significant change in kernel revisions during an upgrade. This is necessary to ensure that the PXE kernel included in the package stays synchronized with the kernel of the new distribution. The `ChangeLog` under the `usr/share/doc/yaci-<version>` directory contains the revision of the kernel. The kernel revision in YACI does not have to match exactly the revision being installed, but the closer it is, the less likely unexpected problems are. The maintainers of YACI are very willing to help with kernel updates and will usually have a new version of YACI out on their web site [29] within a day or two of requesting one. Contact information for the maintainers of YACI can also be found on the web site.

Customized PXE Kernels

Sometimes the standard kernel of YACI, even when it is fully up to date, will be configured differently enough from the kernel to be installed that a custom PXE boot kernel will have to be created. One such situation arose during the course of this project, and although it is unlikely to happen often, it is probably worth sharing a few details concerning the problem and how it was solved. The problem encountered with the Fedora 7 upgrade was that its kernel used a newer module to interface with SATA and PATA hard drives than the YACI kernel did. The newer module represented these types of hard drives as `/dev/sd[a,b...]` instead of the way they have been traditionally represented, i.e., `/dev/hd[a,b...]`. This difference caused the boot loader portion of the node image installation to mis-configure `grub`, so that the new image did not boot properly. After considering possible solutions, the most reasonable one seemed to be to add the new module that used the new `libsata` to

the kernel included with YACI, since most major distributions of Linux were moving to the newer module.

There is ample literature available concerning the intricacies of compiling the Linux kernel, and since the steps for doing this are likely to change from kernel to kernel, only the details that are applicable to YACI are covered here.

1. Obtain a copy of the kernel source that is of the same revision as the kernel to be installed. There are several ways of doing this. In the case of Fedora, all RPMs on the installation DVD have a counterpart source RPM. The kernel, being in its own RPM package, therefore has its own source RPM. This package should be installed on a system that is similar to the type of compute node being targeted for a slave node (i.e., same processor type and OS version).

2. Obtain the latest version of the YACI source RPM and install it on the same machine as the kernel source code. Source RPMs are designated by `yaci-<version>.src.rpm`. Installing the YACI source package places `yaci-12-7.4.tar.bz2` and the `yaci.spec` in the `/usr/src/redhat/SOURCES/` directory. At least this is the case for the Fedora OS; other operating systems may have different locations for installing source files. Uncompress the tar file.

   The YACI source RPM is needed primarily so that the YACI RPM can be rebuilt with the new kernel inside. Also included in YACI are the configuration files used to build previous versions of YACI kernels. These configuration files are useful starting points for the decision of how to configure the new kernel.

3. Prepare kernel code for compilation. With the Fedora 7 kernel release, this involved performing a `make mrproper` and then copying the desired configuration file to `.config` in the top-level directory of the kernel project. Using one of the

more recent YACI configuration files is probably the best choice for a starting point. The other option is to start from one of the default configuration files provided with the kernel.

4. After the configuration file is in place and before making additional modifications to it, it is probably best to do a test compilation. If a compilation error does occur, then most likely there is a problem with the configuration file (although it is a good idea to verify that all relevant patches have been applied). Sorting out configuration problems can be rather tedious. Generally, the best approach is to try different configuration files (either from YACI or one that came with the kernel code itself) until one that compiles completely is found. Then compare the working configuration file (i.e., that one that produces a fully compiled kernel) with the one considered to be closest to what YACI needs. The `xconfig` utility, invoked by doing a `make xconfig`, provides some limited search capabilities, which can be useful in this endeavor. The `xconfig` utility formats the config file, which is a text file, into a more readable listing by organizing related options together so that their connections can be more easily discerned. It also provides additional information about what each option controls. Since the configuration file is a text file it can also be searched with command line utilities, such as `grep`, and compared with other configuration files that are similar in nature.

5. After the kernel compiles, it is good practice to test it by trying to use it in an install. This verifies that it at least provides the same functionality as the

kernel that is to be replaced. To do this, a bzimage[4] needs to be made and placed in the `/tftpboot` directory, replacing the old image.

6. When the kernel from step 5 meets expectations, then changes can be made to the configuration file to add the new desired functionality. Recompile and test the kernel in the same manner as step 5.

7. Once the kernel is performing as desired it needs to be packaged with the YACI RPM. The following instructions assume that the `rpmbuild` package has been installed. Note, there is a `Makefile` associated with the YACI source code but it will not work outside of Lawrence Livermore National Laboratory since it is designed to update the project from the CVS repository before creating the RPM. The instructions below work without access to the repository.

   (a) Replace the kernel image in the

   `/usr/src/redhat/SOURCES/yaci-<version>/`

   directory with the newly compiled image. The easiest way of doing this is to rename the new image the same name as the old. If a new name is desired then changes need to be made to the `pxelinux.cfg/default` file in the project.

   (b) Inspect the project documentation and update appropriately.

   (c) Create a BZIP2 tar file of the YACI project directory. The tar file should also be located in `/usr/src/redhat/SOURCES/`.

   (d) In the `/usr/src/redhat/SPECS` directory edit the `yaci.spec` and update the project version information and documentation. Also, verify that the

[4] A bzimage is a compressed version of the Linux image, which was developed to help work around memory limitations, most notably the i386 Real mode limit [32]

spec file references the correct BZIP2 file and that the "package include file list" is still correct.

(e) Create the RPM by issuing the `rpmbuild -ba yaci.spec` command from the shell prompt while in the `/usr/src/redhat/SPECS` directory.

A new copy of YACI will be in `/usr/src/redhat/RPMS/noarch/`. A new copy of the source RPM can be found in `/usr/src/redhat/SRPMS/`.

Web-Based Application

The Web-based installation application ultimately ported quite cleanly from Fedora Core 5 to Fedora 7. At first it was thought that the Apache web server would need to be recompiled to make it compatible with Fedora 7, but when tested, the current server packaged in the `apache-overlay.tar` file from the Fedora Core 5 release of BSACI functioned correctly. Minor adjustments were needed, but for the most part it just worked. Although the Web-based application itself did not require many changes, it was necessary to learn a certain number of debugging techniques to use in conjunction with it, since it is the control center for the final portion of the installation. Debugging the web application portion of the installation can be rather tedious since data is passed back and forth between PHP, Javascript, and BASH shell scripts, making it rather hard to follow. The fact that there are no easy to install debugging tools for this type of application only adds to this difficultly. Future maintainers of BSACI may have more web experience and therefore this may be less of an issue to them. Even so, a few techniques that proved useful during the Fedora 5 to Fedora 7 upgrade may be useful to future maintainers; these are shared below.

Even without a debugger, a reasonable amount of debugging can be accomplished by printing variable information on the web page itself. Since all code is in script

form and does not require compiling, debugging can sometimes be done in real time by changing scripts and then refreshing the browser. This is useful because BSACI sets a lock when the web application is open, which makes it tricky to reopen the application after the web browser is closed. Some caution should be used with this technique, however, since refreshing the page may have some undesired consequences, such as restarting a lengthy build of the node image.

As mentioned above, once started, the web application puts a lock in place, making it impossible to restart if the web browser has been closed. The lock is to keep more than one web browser from opening the web page at the same time. It is created by placing a file named `DELETE_THIS_LOCKFILE` in the `htdocs/` directory of the web application. Also, along this line, the `CDROOT` and `CONFFILE` environment variables, which are created during the `RUNME.sh` script, are necessary for the web application to work properly. If for some reason the master node is rebooted, or for whatever reason these variables are lost, then these environment variables will need to be restored before restarting the application.

4.3 Testing

Exhaustive testing of all the components in BSACI would take an inordinate amount of time and probably is not necessary since most of its packages have received a reasonable amount of testing before they are released for general consumption. With this in mind, testing focused more on whether the package was correctly installed than on verifying all of its functionality. To this end, a reasonable subset of the more typical use of each package was chosen as the testing criteria.

A brief summary of the testing is given below.

- Slave Node
  - verified that slave node boots correctly
  - verified file system is populated correctly
  - verified that slave node correctly mounts shared folders on Master Node
  - verified that users can do keyless `ssh` into slave nodes from other nodes in the system
  - verified slave node can run a X Window client application
  - verified that node names are correctly generated
- Master Node
  - verified that the `/home`, `/opt`, and `/usr/local` are shared correctly via NFS
  - verified that ssh and MPI keys are correctly generated for each user
  - verified external network connection
  - MPI Packages (MPICH2, Open MPI, LAM, and PVM)
    - verified MPI libraries (A set of example programs is provided with the installation of BSACI, that exercise some of the more commonly used features of these libraries. Although there was not a specific code set for LAM, the Open MPI code set was similar enough to be used to test it. Testing of packages such as the PBS and OSCAR modules, which provide support for programs that use the MPI libraries, was done in conjunction with the MPI testing.)
  - Parallel Shells (PDSH and BEOSH)

  - verified that PDSH and BEOSH shells can be invoked from any nodes in the system, not just the master node
  - verified that it is possible to run shells across all nodes or just a subset of nodes
- Parallel File Systems (PVFS2 and GFarm)
  - verified that both the PVFS2 and GFarm file systems are set up on each node and can be accessed
- ClusMon
  - verified that the web application was functional and able to interface with database

CHAPTER 5

CONCLUSIONS

In the end, the success of a project comes down to one thing: did it work? Fortunately, it can be reported that the Boise State Automated Cluster Installer was successfully upgraded to Fedora Release 7. All packages were successfully ported and a good sampling of the functionality of the cluster was tested and verified to be working. It was decided toward the end of this project that the Boise State Beowulf cluster would be better served by being ported to CentOS for the reasons described in Section 1.3, so the Fedora 7 release was never installed on it. Fortunately, the current release of CentOS (5.2) shared enough in common with Fedora 7 that most of the updated cluster packages could be used "as is" in the CentOS conversion, which was carried out by Dr. Amit Jain. With the use of the Fedora 7 version of BSACI it was possible for the CentOS conversion to be completed in about two weeks, a relatively short amount of time for this project.

One of the purposes of BSACI is to provide a tool that will allow the student to easily explore HPC clusters. Since Fedora is more targeted for the home user than CentOS, the Fedora 7 version of BSACI may be a better choice for the novice, since they may already have some familiarity with it. With this in mind the Fedora 7 version of BSACI can still play an important role.

As with any software project having a maintenance process is important. By following the process outlined and using the scripts provided for updating from one revision of a given distribution to the next should save time and some frustration the next time this has to be done. Having all the binary packages in the repository now guarantees that the ISO image can correctly be rebuilt when needed. Lastly, it is hoped that some of the lessons learned and shared in this report about testing and management of this project will be of benefit to future maintainers.

REFERENCES

[1] Paul Kreiner. *Automated Installation of Linux High-Performance Computing Clusters.* Masters in Computer Science Thesis, Boise State University, 2007.

[2] *OSCAR.* Open Cluster Group. 2000–2008. Web. Last viewed 20 Mar 2009. <http://svn.oscar.openclustergroup.org/>

[3] *Fedora 7 Installation Guide* Red Hat, Inc. July 2007. Web. Last viewed 20 Mar 2009. http://docs.fedoraproject.org/install-guide/f7/en_US/

[4] "Fedora Life Cycle" *The Fedora Project.* Red Hat, Inc. 2009. Web. Last viewed 20 Mar 2009. <http://fedoraproject.org/wiki/LifeCycle>

[5] Ralph Angenendt. "What is CentOS?" *CentOS Wiki.* Creative Commons. 16 Feb 2009. Web. Last viewed 20 Mar 2009. <http://wiki.centos.org/>

[6] Singer, J. "Practices of Software Maintenance." *Software Maintenance, 1998. Proceedings. International Conference on* 16–20 Nov 1998. Pages:139–145

[7] "What makes a cluster a Beowulf?" *Beowulf Project.* Beowulf.org. 2004–2007. Web. Last viewed 20 Mar 2009. <http://www.beowulf.org/overview/index.html>

[8] *MPICH2* Argonne National Laboratory. Web. Last viewed 20 Mar 2009. <http://www.mcs.anl.gov/research/projects/mpich2/>

[9] *The Open MPI Project.* Trustees of Indiana University. Mar 2009. Web. Last viewed 20 Mar 2009. <http://www.open-mpi.org/>

[10] *LAM/MPI Project.* Trustees of Indiana University. 14 Feb 2007. Web. Last viewed 20 Mar 2009. <http://www.lam-mpi.org/>

[11] *PVM (Parallel Virtual Machine).* Computer Science and Mathematics Division of Oak Ridge National Laboratory. 3 Apr 2007. Web. Last viewed 20 Mar 2009. <http://www.csm.ornl.gov/pvm/>

[12] Kirsten Allison. *PTK: A Parallel Toolkit Library.* Masters in Computer Science Thesis, Boise State University, 2007.

[13] *Global Arrays Toolkit.* Computational Sciences and Mathematics, Pacific Northwest National Laboratory. Dec. 2008. Web. Last viewed 20 Mar 2009. <http://www.emsl.pnl.gov/docs/global/>

[14] *HDF5.* The HDF Group. Mar 2009. Web. Last viewed 20 Mar 2009. <http://www.hdfgroup.org/HDF5/>

[15] Michael Schindler *SZIP.* Compression Consulting. 1999–2002. Web. Last viewed 20 Mar 2009. <http://www.compressconsult.com/szip/>

[16] Jason Main & Amit Jain. "PRAND: A Parallel Random Number Generator" *Dr. Amit Jain Research.* Boise State University. 2 Oct 2008 Web. Last viewed 20 Mar 2009. <http://cs.boisestate.edu/∼amit/research/prand/>

[17] "OSCAR Administration Guide" *OSCAR* Open Cluster Group. 2000–2008. Web. Last viewed 20 Mar 2009. <http://svn.oscar.openclustergroup.org/trac/oscar/wiki/AdminGuide/Packages#Switcher>

[18] *PBS Gridworks.* Altair Engineering, Inc. 2009. Web. Last viewed 20 Mar 2009. <http://www.pbsgridworks.com/>

[19] Conrad Kennington. *A Beowulf Cluster Monitor.* Masters in Computer Science Thesis, Boise State University, 2006.

[20] *Parallel Distributed Shell.* Source Forge. 1999–2009. Web. Last viewed 20 Mar 2009. <http://sourceforge.net/projects/pdsh/>

[21] Mason Vail. *beosh: The Beowulf Cluster Shell.* Masters in Computer Science Thesis, Boise State University, 2006.

[22] Mason Vail & Amit Jain. "BEOSH: The Beowulf Cluster Shell" *Dr. Amit Jain Research.* Boise State University. Web. Last viewed 20 Mar 2009. <http://cs.boisestate.edu/∼amit/research/beosh/>

[23] *PVFS.* PVFS Project. Web. Last viewed 20 Mar 2009. <http://www.pvfs.org/>

[24] *Gfarm file system.* Asia Pacific Grid. 31 Mar 2007. Web. Last viewed 20 Mar 2009. <http://datafarm.apgrid.org/>

[25] Amit Jain. "Beowulf Cluster Design and Setup" *Dr. Amit Jain Research.* Boise State University. 26 Apr 2006. Web. Last viewed 20 Mar 2009. <http://cs.boisestate.edu/∼amit/research/beowulf/beowulf-setup.pdf>

[26] Douglas Eadline, Ph.D. “Achieving High Performance at Low Cost: The Dual Core Commodity Cluster Advantage” *High-Performance and Enterprise Computing.* Appro. June 2006. Web. Last viewed 20 Mar 2009. <http://www.appro.com/whitepaper/PentiumD-WP5-final.pdf>

[27] Joel Adams & David Vos. “Small-College Supercomputing: Building A Beowulf Cluster At A Comprehensive College” *Publications (Joel Adams).* Calvin College. Mar 2002. Web. Last viewed 20 Mar 2009. <http://www.calvin.edu/∼adams/professional/publications/SmallCollegeSupercomputing.pdf>

[28] “Cluster (Computing)” *Wikipedia.* Wikimedia Foundation, Inc. Mar 2009. Web. Last viewed 20 Mar 2009. <http://en.wikipedia.org/wiki/Computer_cluster>

[29] Makia Minich & Trent D’Hooge *YACI.* Lawrence Livermore National Laboratory. Web. Last viewed 20 Mar 2009. <http://www.yaci.org/>

[30] *RPM.* Open Source Lab. Web. Last viewed 20 Mar 2009. <http://www.rpm.org/>

[31] *PXELinux.* The SysLinux Project. 11 Jun 2008. Web. Last viewed 20 Mar 2009. <http://syslinux.zytor.com/wiki/index.php/PXELINUX>

[32] “vmlinux” *Wikipedia.* Wikimedia Foundation, Inc. 4 Feb 2009. Web. Last viewed 20 Mar 2009. <http://en.wikipedia.org/wiki/Vmlinux>

[33] *Apache HTTP Server Project.* The Apache Software Foundation. 2009. Web. Last viewed 20 Mar 2009. <http://httpd.apache.org/>

[34] *VMware.* VMware Inc. 2009. Web. Last viewed 20 Mar 2009. <http://www.VMware.com>

[35] *Knoppix.* Knoppix.Net. Web. Last viewed 20 Mar 2009. <http://www.knoppix.com/>

[36] *FedoraLiveCD. The Fedora Project.* Red Hat, Inc. 13 Feb 2009. Web. Last viewed 20 Mar 2009. <http://fedoraproject.org/wiki/FedoraLiveCD>

[37] “Live CD” *Wikipedia.* Wikimedia Foundation, Inc. Mar 2009. Web. Last viewed 20 Mar 2009. <http://en.wikipedia.org/wiki/Live_CD>

[38] *Subversion.* Tigris.org. 2001–2008. Web. Last viewed 20 Mar 2009. <http://subversion.tigris.org/>

APPENDIX A

RPM SORTING SCRIPTS

A.1 extractPackName.bash

```
#!/bin/bash

if [ "$1" = "-h" ]  || [ "$1" = "-help" ] || [ "$1" = "--help" ]
then
 more << EOF
  $0 <Directory to Search> [Search Depth]

  Searches directory and all its subdirectoies for rpm packages
  and extracts the their names.  A optional search depth
  can be added to limit the depth of the search.

 EOF
 exit 0;
fi

if [ ! -d "${1}" ]
then
  echo Need valid directory to search
  exit -1
fi

pushd "${1}" 2>&1 >/dev/null

if [ -n "$2" ]
then
  RPMS=‘find . -maxdepth ${2} -name ’*.rpm’‘
```

```
else
  RPMS=‘find . -name ’*.rpm’‘
fi

for i in $RPMS
do
  name=‘rpm -qp --queryformat "%{NAME}" $i‘
  echo -e "$name,${i##*/}"
done

popd  2>&1 >/dev/null
```

A.2 findNewRev.bash

```
#!/bin/bash

if [ "$1" = "-h" ]  || [ "$1" = "-help" ] || [ "$1" = "--help" ]
then
 more << EOF
  $0 <Input File> <Directory to Search>

  Searches through a directory of rpm packages and
  tries to match those packages to the list in the
  input file.

 EOF
 fi


if [ ! -f "$1" ]
then
  echo Need valid input file
  exit -1
fi

if [ ! -d "${2}" ]
then
  echo Need valid directory to search
```

```
  exit -1
fi

for i in `cat $1`
do
  i=${i%%,*}
  potential=`ls "${2}"$i*.rpm 2>/dev/null | sed 's/ /\\ /g'`
  if [ -n "$potential" ]
  then
    (
    IFS=$'\n'
    found=0
    for j in $potential
    do
      temp=$(rpm -qp --queryformat "%{NAME}" "${j}")

      if [ "$i" = "$temp" ]
      then
        echo $i,${j##*/}
        found=1
      fi
    done
    if [ $found -eq 0 ]
    then
      echo $i,###
    fi
    )
  else
    echo $i,###
  fi
 done
```

APPENDIX B

MASTER NODE RPM INSTALL SCRIPT

```
#!/usr/bin/perl

$|=1;

if (@ARGV==0)
{
 print "Useage: $0 <rpm list>\n";
}

open (RPMFILE, $ARGV[0]) or die "Cannot open file";

@rpmList= <RPMFILE>;

close (RPMFILE);

chomp @rpmList;

foreach $i (@rpmList) {
 $i=~s/#.*//;
}

for ($i=0; $i<@rpmList; $i++) {
 if ($rpmList[$i]=~m/^[ \t]*$/) {
   splice(@rpmList, $i, 1);
   $i--;
 }
}

foreach $i (@rpmList) {
```

```
 $file=$i;
 $options=$i;
 $file=~s/(.*),/$1/;
 $options=~s/.*,[ \t]*(.*)/$1/;
 if ($options=~m/^$/) {
     $options="--upgrade --nosignature";
 }

 while (1) {

  print "Installing $file\n";
  $results=`rpm --test $options $file 2>&1`;
  if ($?!=0) {
   if ($results=~m/already installed/) {
       print "   Skipping installation of $file \
... package already installed.\n";
       last;
   }
   else {
     print "\nThe following conflict ocurred while\
 testing the installion of $file\n\n";

     print "--$results--\n";
     print "How do you want to proceed?\n\n";
     print "If you can correct the problem in\
 another shell\n";
     print "then type \"t\" to test again and \
 install.\n\n";
     print "To force installation by using the \
--force option type \"f\"\n";
     print "To do a --nodeps installation type \
\"d\"\n\n";
     print "Options f and d can be combined.\n\n";
     print "Or \"Enter\" without typing anything \
to skip\n";

     $response=<STDIN>;

     if ($response=~m/[t]/) {
         next;
     }
```

```
      elsif ($response=~m/[fFdD]/) {
        $response=~s/[^fFdD]//;
        $response=~s/[fF]/ --force/;
        $response=~s/[dD]/ --nodeps/;
        $options=$options . $response;
        system "rpm $options $file";
        last;
      }
      else {
        last;
      }
    }
   }
   else  {
     system "rpm $options $file";
     last;
   }
  }
}
```