AUTOMATED INSTALLATION SOFTWARE FOR LINUX HIGH-PERFORMANCE COMPUTE CLUSTERS

by

Paul Kreiner

A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Computer Science

Boise State University

April 2007

The thesis presented by *Paul Kreiner* entitled *Automated Installation Software for Linux High-Performance Compute Clusters* is hereby approved.

Amit Jain, Advisor	Date
Tim Andersen, Committee Member	Date
John Griffin, Committee Member	Date
John R. Pelton, Graduate Dean	Date

dedicated to Edward W. Kreiner, my friend, inspiration, role model, and father

ACKNOWLEDGEMENTS

First and foremost, I wish to thank my wife, Mayela, for her unwavering support through all the long hours I spent away from her and our family. I have focused heavily on my education over the past three and a half years, and my wife and three children have sacrificed by not getting my full attention.

I wish to thank my parents, who are shining examples of the achievements toward which I can (and do) strive. My mother gets special thanks, for providing me the education and tools which have enabled me to go this far, and for continuing to nudge, cajole, and encourage me to complete my higher education. To any who think home schooling produces socially and/or academically disadvantaged students, I am a living counterexample, and I thank my mother for that. Thanks to my father for demonstrating that it is possible to support and care for a family and still achieve academic excellence while earning an advanced education. You are my role model.

Thanks to several in academia, particularly my advisor, Dr. Amit Jain, for being available and willing to help. If it were not for his positive attitude and personable demeanor at my initial meeting with him in 2004, I would not have continued to pursue my secondary education. His breadth and depth of knowledge is inspiring, and working with him has been a pleasure. Thanks also to Dr. John Griffin, Dr. Sin Ming Loo, and Dr. Gary Ganske (Northwest Nazarene College) for demonstrating a personal interest in me.

Finally, thanks to my employer, CRI Advantage, for allowing me the flexibility to continue my studies while maintaining my employment full-time. Without this flexibility I would have been unable to complete either my undergraduate or graduate coursework.

AUTOBIOGRAPHICAL SKETCH

Paul Kreiner was born in Texas, and moved to Montana when he was four years old, where he grew up loving the mountains, the great open spaces, and the snow. An early curiosity regarding computers and how they worked, the purchase of a Commodore C-64, and a library book which was an elementary BASIC tutorial were enough to launch Paul on the path which he follows today. The harsh winters of eastern Montana gave him plenty of free time to start tapping away at his C-64, learning and mastering BASIC on that system, then adding 6502 assembly language when the poor performance and inherent limitations of BASIC became too crippling.

Paul's family moved to Meridian, Idaho in 1993, when his father was asked to take over the pulpit of Meridian Assembly of God church. It was here that Paul acquired his first IBM-compatible system (a 12MHz 80286 PC), followed about two years later by a 90MHz Pentium® system running Microsoft® Windows™. It took another year before Paul found his first non-crippled operating system, the venerable FreeBSD, and installed it. The installation went well, using a stack of about forty floppy disks, but he was unable to figure out what to do with the end result. In 1997, Paul was finally able to take the plunge into Linux (Slackware 3.2 at the time) and actually do something useful with the system after he got it installed. The ability to get "under the hood" of the system and learn by interacting with the operating system at a low level was a vast improvement from the relatively stale Windows environment,

and by 1999, he was a convert. He still fondly recalls the thrill of compiling the Linux kernel from source for the first time, then successfully booting it on his hardware.

Since then, Paul has continued to work with Linux and Open Source software, both at home and at work, implementing existing software packages, modifying others, maintaining some personal Linux kernel patch trees, and even starting two entrepreneurial endeavors based on Open Source software and technologies. This latest foray into the world of High-Performance Computing represents another step in the path of increased breadth and depth of exposure to the technologies and solutions provided by Open Source software.

Paul is looking forward to many more years of challenging, rewarding work with software that "just works", isn't based on undocumented or proprietary technology, is widely supported, and doesn't try to get in his way.

ABSTRACT

The process of acquiring and building the software necessary for a complete, fully-functional high-performance computing (HPC) cluster is complex and daunting, even for someone experienced in building non-HPC systems and applications. While many cluster installation utilities exist, most of them simply automate the process of installing software across a homogeneous collection of systems. These utilities still require the system administrator to perform significant amounts of application setup and configuration work before a fully-functional cluster is realized.

The Boise State Automated Cluster Installer (BSACI) provides system administrators a tool which extends the automation process to include installation and configuration of user-space tools, job schedulers, parallel and cluster-specific development and runtime environments, and monitoring. BSACI will deliver a fully-functional Linux-based Beowulf HPC cluster, with minimal knowledge and intervention required from the system administrator. No longer is the system administrator expected to digest tomes of documentation regarding various cluster-specific software, just to get a basic working configuration. Someone with basic knowledge of network configuration and disk layout will be able to install a cluster in literally a matter of minutes.

BSACI provides a robust tool which is usable by students, instructors, and researchers to build a fully-functional Linux-based HPC cluster.

TABLE OF CONTENTS

LIST OF FIGURES . . . . . **xv**

1 INTRODUCTION **1**

1.1 Rationale and Significance 1

1.2 Prior Work 3

1.2.1 Yet Another Cluster Installer (YACI) 4

1.2.2 SystemImager 4

1.2.3 IBM Extreme Cluster Administration Toolkit (xCAT) 5

1.2.4 Fully Automatic Installation (FAI) 6

1.2.5 Open Source Cluster Application Resources (OSCAR) 8

2 AN EASY-TO-USE AUTOMATED CLUSTER INSTALLER . . **10**

2.1 The Ideal Automated Cluster Installer 10

2.1.1 Easy to Use 10

2.1.2 Robust 11

2.1.3 Broad User Base 11

2.1.4 Good Hardware Support 12

2.1.5 Maintainable 12

2.1.6 Automated 13

2.2 Design Goals 13

3 BSACI DESIGN AND IMPLEMENTATION 18

3.1 Project Approach 18

3.1.1 Build on Existing Tools 18

3.1.2 Human-readable files 19

3.1.3 Include useful tools for development and maintenance 20

3.1.4 Plan A: use Amit's add-ons 21

3.1.5 Plan B: make a standalone tool 22

3.2 The YACI Framework 24

3.2.1 The Role of the Master Node 24

3.2.2 What YACI Provides 26

3.2.3 What's Missing 27

3.3 Handling Configuration Files 28

3.3.1 The master parameters file 28

3.3.2 Scripts, scripts, and more scripts 30

3.4 Modularity 31

3.5 Speed and Performance 32

3.5.1 A watched pot never boils 32

3.5.2 Use the cluster when possible 33

3.5.3 Employ parallelism 34

3.6 Graphical User Interface (GUI) 35

3.6.1 Self-contained Apache and PHP 36

3.6.2 Asynchronous Javascript and XML (AJAX) 37

3.6.3 The xajax utility library 38

3.7 Command Line Interface (CLI) 38

3.7.1 Why provide a CLI? 38
3.7.2 Usage Scenario 39
3.8 Core Software 40
3.8.1 Portable Batch System (PBS) 40
3.8.2 Parallel Virtual Machine (PVM) 40
3.8.3 OpenMPI 41
3.8.4 MPICH2 42
3.9 Additional Software 43
3.9.1 Data Abstraction Layers 43
3.9.2 Cluster Filesystems 44
3.9.3 Cluster Management Tools 46

4 USING THE BOISE STATE AUTOMATED CLUSTER INSTALLER 51

4.1 Prerequisites 51
4.1.1 Supported Processors 51
4.1.2 Supported Disks and Cluster I/O 52
4.1.3 The Data Network 54
4.2 Preparing the master node 55
4.2.1 Installing Fedora Core 56
4.3 Installing the cluster 59
4.3.1 The initial RUNME.sh setup script 59
4.3.2 MASTER-SCRIPT-1.sh: preparing the master node 61
4.3.3 Using the web wizard to enter configuration details 62
4.3.4 Detecting compute nodes 64

4.3.5 `MASTER-SCRIPT-2.sh`: building the node tarball 65
4.3.6 `MASTER-SCRIPT-3.sh`: deploying the node tarball 66
4.3.7 Final steps 67
4.3.8 Using the Command-Line Interface 68
4.4 Restarting a cluster installation 72

5 MAINTENANCE 73
5.1 Maintaining the cluster 73
5.1.1 Shared Network Filesystems 73
5.1.2 The Parallel Distributed Shell `pdsh` 74
5.1.3 Portable Batch System (OpenPBS) tools 75
5.1.4 ClusMon 78
5.1.5 Patches and Software Upgrades 80
5.1.6 To upgrade or to re-install? That is the question. 84
5.2 Maintaining the Boise State Automated Cluster Installer 87
5.2.1 Acquire the most up-to-date packages 88
5.2.2 Create a list of required packages 91
5.2.3 Test the new node tarball 94
5.2.4 Test the BSACI configuration scripts 96
5.2.5 Test the graphical and command-line user interfaces 99
5.2.6 Test other custom built packages 101
5.2.7 Test a complete installation 102
5.2.8 Test the final result 106
5.2.9 Updating custom-compiled and embedded code 108

6 CONCLUSIONS 113

6.1 What have we done so far? 113

6.2 Future Directions 114

REFERENCES 116

APPENDIX A MASTER PARAMETERS FILE KEYS 119

APPENDIX B BUILD AND TEST ENVIRONMENTS 122

APPENDIX C INSTALLATION README FILES 126

C.1 README 126

C.2 README.GUI 131

C.3 README.CLI 132

C.4 README.example-code 134

C.5 scripts/NEWORDER 135

C.6 scripts/NEWORDER.notes 137

APPENDIX D DEVELOPMENT README FILES 139

D.1 README.placeholder_files 139

D.2 development-tools/README 139

APPENDIX E PROJECT MANAGEMENT 141

E.1 Source control 141

E.2 Where to get the code 141

LIST OF FIGURES

3.1 Network design with private cluster network and dual-homed master node. 25
3.2 Format of the master parameters file. 29
3.3 Code which loads network parameter keys as local shell variables. . . 30
3.4 How `beosh` distributes tasks. 48

Chapter 1

INTRODUCTION

1.1 Rationale and Significance

In the past several years, High-Performance Computing using Linux-based clusters has become prolific, all the way from the huge 100 teraflops/second clusters at Lawrence Livermore National Laboratory[2] to personal clusters that run on a couple of salvaged personal computers under students' and researchers' desks. As Linux-based clusters have grown in popularity, the number of people wishing to get a taste of this new technology has also increased. However, the tools and processes to install these clusters have remained complex and difficult to master. As a result, many potential new users have been scared away from Linux cluster technology, simply because they did not have the time or the desire to tackle the steep learning curve involved in assembling a Linux cluster.

By mid-2005, several commercial and open-source projects and tools were available, each attempting to make cluster installation and maintenance a little easier. Most of these tools were targeted toward administrators who were already familiar with the concepts and tools necessary for cluster installation, and the tools simply

automated some of the mundane and error-prone parts of the installation process. There was still not an "easy" cluster installation tool which was designed to introduce someone with little to no cluster experience to the technology.

The Boise State Automated Cluster Installer (BSACI) is intended to fill that gap, providing a cluster installation tool which can be given to students or researchers who have little Linux system administration experience, but desire to build a fully-functional Linux-based compute cluster. BSACI provides a turn-key Linux compute cluster based on the Fedora Core distribution, complete with parallel development tools, job scheduling, cluster management and monitoring tools, two cluster filesystems, and additional performance tweaks specific to high-performance clusters. All the research, coding, patching, configuration, and integration necessary to get the various tools to work together has been done.

The driving concept behind BSACI was to provide the user a "one-click installation" experience, or as close to that as possible, including a graphical user interface which would clearly identify and describe the information needed to configure the cluster and would provide context-sensitive help for users. At the same time, the tool also needed to satisfy a second role—for more experienced users, a more fully-automated installation method (one which does not require a graphical user interface or any user intervention) was also desired. This flexibility would allow the tool to remain useful to a broad set of users, from those with little experience to those who are experienced, but need a quick, fully-automated method of bringing an entire cluster

on line in just a few minutes.

For example, a professor can customize the packages provided in BSACI, burn a CD or DVD of the resulting code, and provide this to his or her students, enabling them to easily build their own parallel clusters at home or work, with minimal user intervention required. Similarly, a system administrator experienced with using BSACI can easily bring up an entire departmental research cluster in under one hour, and enjoy a pleasant lunch break during the process. Making Linux clustering technology fast, automatic, and easily accessible gives users a chance to tackle and learn from problems which they would otherwise not be exposed to, and it promotes further development and adoption of high-performance cluster computing as these users move into the work force. It also enables system administrators to focus their time on more rewarding problems, instead of wasting time performing relatively mundane cluster installation tasks.

1.2 Prior Work

Before embarking on a quest to build a cluster installation tool, it would be wise to see what other tools are out there and how they address the problems inherent in building a cluster installer that will work for the masses. Each of the tools that exists does so to fill a particular niche, and none of them fill the niche of a turn-key cluster installer. Following is a brief overview of several popular cluster installation packages.

1.2.1 Yet Another Cluster Installer (YACI)

YACI[1] is a lightweight tool for quickly installing Linux on a set of machines, typically in a cluster environment. It is a relatively simple tool which restricts itself to RPM-based distributions such as Red Hat and SuSE. Because it is lightweight and simple, it is easy to modify, but it also has a primitive feature set compared to other cluster tools. In this sense, it may be better to refer to YACI as a framework for cluster installation—it provides the basics necessary to install software on multiple machines, but there are many bells and whistles which are missing.

Documentation for YACI is relatively sparse and almost always out of date. Often, new builds of YACI introduce new functionality and new bugs. As a result, YACI is definitely not a tool for novices, as it requires a heavy investment of effort to get working. However, it is a fine tool for experienced cluster installers, and those who want a quick way to install many systems in parallel. Because of its open source license, its simplicity, and the fact that it is built using Perl and shell scripts, it is extremely easy to modify.

1.2.2 SystemImager

SystemImager[3] uses the "golden image" concept: it creates a filesystem-level image of an existing system, then deploys this image to a number of computers. SystemImager is unique among the tools reviewed in this chapter, because it provides a filesystem-level 'diff' capability which allows existing systems to be upgraded by send-

ing new and changed files to update their existing installations, instead of re-imaging the systems from scratch. It relies on a companion tool, SystemConfigurator[4] to push out custom configuration files on a per-machine basis.

One of the key advantages of SystemImager is that is is distribution agnostic. Because it uses filesystem-level imaging, instead of pushing out .rpm or .deb packages to cluster nodes, it will work for any distribution. Likewise, SystemConfigurator has "footprints" which describe the location and style of configuration for each distribution, so it is well-adapted to many distributions.

Unfortunately, SystemImager cannot easily handle situations where one system image is to be installed on machines with different hardware, and it requires that the original system from which the "golden image" is generated be configured carefully by hand. While this approach is valuable, it does not meet our objective for ease of installation on systems with differing hardware.

1.2.3 IBM Extreme Cluster Administration Toolkit (xCAT)

xCAT[5] is a collection of tools for the deployment, administration, and maintenance of Linux clusters. Along with basic installation of the operating system across multiple machines, it includes high-performance computing software such as batch job schedulers, parallel programming libraries, and cluster management utilities. xCAT appears to be very well documented and supported, with new versions posted at least once per year. It is designed specifically to support RPM-based distributions, such as

Red Hat and SuSE, and it supports a broad variety of hardware platforms, including Intel's 32-bit 80x86-compatible architecture (IA32), Intel's 64-bit Itanium® architecture (IA64), Advanced Micro Devices' 64-bit extended 80x86-compatible architecture (AMD64), and Motorola's Power PC architecture.

xCAT includes significant added functionality that is specific to IBM hardware, such as low-level hardware monitoring and power management tools, remote console and reboot functionality, and remote BIOS console. Interestingly, it also supports imaging a Windows disk for deployment, although it primarily focuses on Red Hat and SuSE Linux. While all of the non-hardware-specific capabilities of xCAT are available in other cluster management tools, a system administrator running with all IBM hardware would definitely benefit from the low-level tools and functionality which are included with xCAT, especially when maintaining a compute cluster.

As one might expect, all this functionality does have a cost. In the case of xCAT, there are actually two costs: First, xCAT is proprietary software, which must be licensed from IBM. Second, it has a steep learning curve, simply due to the complexity of the package. Unfortunately, these are both show-stoppers for us. Any tool we wish to use must be freely distributable to others, and must be easy to use.

1.2.4 Fully Automatic Installation (FAI)

The Fully Automatic Installation tool[6] will install and update a cluster or network of workstations in an unattended fashion, with minimal user intervention. It relies on

cfengine, Perl, and shell scripts to customize the configuration files on each machine, and is scalable to many hundreds of nodes. It also supports a broad variety of target hardware platforms, including IA32, IA64, AMD64, and Alpha.

Unlike most other cluster installers, FAI is designed around the Debian package management system, and does not support Red Hat packages. It is a generic installer, not just for installing clusters, but for any situation where a large number of similar operating system installations must be performed. Because it is not a cluster-specific installation tool, it operates at a lower level than cluster-specific tools such as xCAT. The system administrator must first define the packages necessary for compute nodes, then write and debug any new scripts and cfengine directives for maintaining the configuration files related to these packages. Setting up FAI also involves establishing local Debian package mirrors, setting up the master node by hand, and building class files for each machine type to be installed.

As one may infer from the preceding description, FAI is a very powerful and configurable tool. It also has the benefit of a complete, professional, and up-to-date set of documentation. In the hands of an experienced system administrator, FAI could be a powerful time-saving tool. However, configuring FAI generically to work on any hardware that could be thrown at it is not an easy task, nor is the task of modifying it to support RPM-based distributions, such as Fedora Core. As its author states, FAI is targeted toward experienced system administrators who wish to save time by automating mundane tasks. It is not a tool for the inexperienced.

1.2.5 Open Source Cluster Application Resources (OSCAR)

OSCAR[7] is a mature cluster installation tool, designed to take the best known tools and methods for building, programming, and using clusters, and combine them into a fully integrated and easy-to-install software bundle intended for high performance computing. It supports RPM-based Linux distributions, including Red Hat, Fedora Core, CentOS, and SuSE, and supports IA32 and AMD64 hardware platforms.

OSCAR was fairly immature when the Boise State Automated Cluster Installer project was started, but it has matured immensely in the meantime, including support for the 2.6 kernel, recent versions of several parallel job schedulers and message-passing libraries, cluster node monitoring software, an installation graphical user interface, etc. It also fully automates node installations, by taking a filesystem-level snapshot of a working node and deploying it, or by building a snapshot in a `chroot`'ed environment and deploying it. Overall, OSCAR supports a broad range of features and functionality. This, plus the project's stated goal of being a fully integrated, easy-to-install software bundle, make it a good choice for users with some experience in Linux system administration.

However, OSCAR still has its rough edges, and is not as user friendly as one might hope when handing it to less experienced users. Perusing the OSCAR installation manual[8] reveals that there are a number of manual pre-configuration steps which the user must perform to ensure that the master node will function properly with OSCAR. Additionally, users are admonished not to do anything unpredictable while

using the graphical user interface, or to try performing installation steps out of order, because the installer will likely crash. Users are reminded to read the installation manual thoroughly.

These restrictions are not ideal for an installer which should be robust in the face of misuse and misunderstanding by inexperienced users. While a 100%-foolproof software package may never be reality, our goal is to provide software that is more forgiving and protects the user better than this. While OSCAR is undoubtedly a good tool (probably the best of all the tools analyzed in this project), even the current release of OSCAR is still lacking in this critical area.

Chapter 2

AN EASY-TO-USE AUTOMATED CLUSTER INSTALLER

2.1 The Ideal Automated Cluster Installer

The ideal cluster installer should meet several design criteria, and it is these criteria which the current design of the Boise State Automated Cluster Installer attempts to fulfill.

2.1.1 Easy to Use

First and foremost is ease of use. The installer should be easy enough to use that a novice user can successfully install it without needing to consult reference guides or read extensive documentation prior to starting the installation process. Current system hardware and software configurations will be detected, missing software packages installed, and configurations updated to allow the cluster software to operate. Automatic configuration should be applied whenever possible, minimizing the number of questions which must be asked of the user. Ideally, the users should need to know no more about their system to install the cluster than they knew when performing the base install of the operating system on the master node.

2.1.2 Robust

Along with ease of use is the concept of "idiot-proofness" or robustness. This is important because the ideal cluster installer should gracefully and predictably handle erroneous input. Novice users often provide input which is blatantly incorrect or not well thought out, or they may skip critical installation and configuration steps completely. The installer must do everything possible to avoid giving novice users the opportunity to shoot themselves in the foot through such actions. User input must be double-checked for validity, and the user prompted when he or she provides obviously incorrect data. Users should be given a restricted, simple set of options which must be completed with valid data before proceeding to the next set of options. Invalid data should result in a meaningful error message which indicates what data should be corrected, and context-sensitive help should be available to help the user understand the effect(s) of each set of options.

2.1.3 Broad User Base

The installer should be a useful tool for a broad collection of users, ranging from novices to experts. Its user interface must be simple and friendly enough that inexperienced users are not scared by complex questions, frustrated by lack of online help, or confused by too many choices. At the same time, the installer must provide an interface that expert users will not find limiting, annoying, or a waste of time. Put simply, the installer must make the complex tasks of cluster installation and configu-

ration accessible to novices, and it must not get in the way of expert users who wish to automate the cluster setup process.

2.1.4 Good Hardware Support

The installer must support a wide range of software and hardware. The ideal installer should also support a broad range of Linux distributions and commodity hardware platforms, such as IA32, IA64, AMD64, Power PC and Alpha. Because students and independent researchers are the primary target demographic, the installer must accommodate clusters which are built on varying (often cheap) hardware. Users operating on a budget will probably build a cluster out of scavenged hardware, and will not have a room full of identical hardware at their disposal.

2.1.5 Maintainable

Because software is continually evolving and being upgraded, the ideal installer should also be maintainable—able to easily track the changes in Linux distributions as new versions are released—without excessive work on the part of the maintainer. Code that is specific to a single distribution or package version should be modularized, so an upgrade that is incompatible with the current module code does not require an audit of the entire codebase to identify what must be changed. Instead, changes will be localized to a single module, making the job of the maintainer easier.

2.1.6 Automated

Finally, the installer is generally worthless if it doesn't save time. It should be fast, and should require minimal user interaction. Many system installation tasks require some amount of babysitting, where the system administrator busy-waits until a prompt informs him or her to press a key or some similar action. This type of busy-waiting is just as wasteful in real life as it is in the programming world, and must be minimized by the installer. Ideally, the installer should interact with the user exactly once, when collecting configuration information for the install. After this,it should proceed automatically through the remainder of the installation process.

2.2 Design Goals

For the BSACI project, the initial design goals stemmed from the concepts defined as part of the ideal cluster installer. Scope for the initial design was limited to allowing students to automatically install a working cluster on their own collection of personal computers at home or at work. Considering the type of hardware these target users have available, and given previous experience with clustering on the Red Hat 9 and Fedora Core 1 platforms, it was decided that the cluster installer should restrict support to the Fedora Core 3 distribution, running on the IA32 hardware platform. Users were expected to perform a clean "full" installation of Fedora Core 3 on the system chosen as the master node, and then insert the BSACI DVD to launch the cluster installer. Because the primary users were students, a full graphical

user interface, presented as a series of web pages, walked them through the steps of configuring their software, building a distribution image, detecting the compute nodes, and finally installing software on them.

As the project matured, these goals were refined, and other goals were added in an effort to move BSACI closer to being the ideal cluster installer. The core requirement for BSACI remains: It must make Linux high-performance computing accessible to novice users. Plenty of installers exist which fulfill the technical requirements of being a good cluster installer, but none of them can offer a turn-key user experience. This is what sets BSACI apart. The following design goals were defined for BSACI:

- The primary audience is students, particularly students who may have minimal knowledge of Linux system administration, but who wish to wet their feet with parallel programming in a message-passing environment.
  - The installer should be robust when given invalid input, sanity-checking as many values as possible, and prompting the user if any sanity checks fail. The interface should provide valuable feedback in the event of an error, helping the user correct their mistake and avoid common pitfalls.
  - The primary mode of interaction with the installer is by means of a modern web-based graphical user interface. This provides a rich and flexible environment for displaying data and prompts to the user, and for acquiring user feedback.
  - Target users are familiar with "wizard-style" user interfaces, so the installer

will adhere to this paradigm. The advantage of this approach is that it is naturally simple, limiting the information displayed to the user and the questions asked of the user to only a few at a time. Users can easily see their progress as they advance through each screen of the wizard. Another advantage of this approach is that it keeps users from performing setup steps in an incorrect order.

  - The installer should operate as close to "one-click" as possible. The fewer manual (or preparatory) steps the user must perform prior to starting the installation, the better. It is recognized that a small number of exceptional conditions may still require manual intervention, but these cases must be minimized.

- While the primary audience is students and novice users, the installer must remain a useful tool to advanced and expert users.

  - The installer must not require the user to use the GUI, nor must it require the user to use the physical console of the master node for performing a cluster installation. The system administrator should be able to start and monitor the installation from any point in the network.

  - The installer must not get in the way of a system administrator who wishes to perform a *completely unattended* install. It must support automation, and be easily integrated by other automation tools, such as Perl or shell scripts.

  - The installer must follow standard conventions when reporting errors, and not require system administrators to write complex text parsers to `expect` scripts just to interface with the installer.

- The installer must support ubiquitous IA32 hardware, and must not require all cluster nodes to be identical. While some similarity between systems is necessary, the installer must try to accommodate systems with varying hardware. Other hardware platforms (IA64, AMD64, etc.) are not yet popular and cannot be fully tested, therefore they are not yet supported.

- The installer must be Free Software. This precludes the integration of any proprietary or non-free software into BSACI. Inclusion of non-free software limits the audience to whom this software may be distributed, and may alienate a class of users entirely. While not including such software reduces the utility of BSACI, being able to reach a wider audience is an acceptable tradeoff.

- The installer builds on a recent version of the Fedora Core operating system. This is a widely-used Linux distribution which has a broad set of software available for it, and is actively being maintained. It will appeal to the broadest set of potential users, and will be widely available.

- The installer must be easy to maintain. Documentation should be provided, giving guidelines on how to update and/or customize the available packages and configuration scripts in the future. Each configuration change should be

isolated in its own script, so incompatibilities introduced by newer software remain localized to that one script.

While the project does try to approximate the ideal cluster installer, some of the concepts behind the ideal installer were not part of the project design goals, typically for pragmatic reasons. For example, supporting a broad variety of hardware platforms would be ideal; however, the BSACI author only had IA32 hardware readily available for development and testing. Given this, and the fact that IA32 is the most widely-available commodity hardware at the time of this writing, the decision was made to restrict the project design so BSACI only officially supports IA32.

Chapter 3

BSACI DESIGN AND IMPLEMENTATION

3.1 Project Approach

3.1.1 Build on Existing Tools

There are many cluster installation tools already in existence, but many do not meet the specific goals defined for the BSACI tool. BSACI could be implemented as an all-new cluster installer, or it could be designed to extend and customize an existing cluster installer. It was desirable to reuse as much existing code as possible, which would allow development time to be focused on adding features to BSACI, instead of reimplementing and debugging functionality already present in other cluster installers.

After reviewing the capabilities of other existing tools, YACI was found to be a good candidate to use as a starting platform for BSACI modifications. YACI supports Fedora Core, is designed for IA32 and AMD64 architectures, and is architected to provide a simple, no-frills cluster installation framework. Therefore, very little of the existing YACI code would need to be modified when adding new features and functionality to the YACI codebase.

3.1.2 Human-readable files

Any cluster installer which performs more than basic system configuration must have a means of tracking variables, options, and user-supplied parameters which affect the installation process. For an automated cluster installer, the list of configuration items which must be tracked gets quite large, and changes over time as new software packages replace older versions. It is tempting to use a machine-oriented data format, such as XML, when building a means of tracking these data elements, since such a data format is designed for precisely this purpose.

However, BSACI was designed to make life easier for both novices and experienced users alike. Most human users do not wish to parse through an XML data file when trying to manually set parameters in a configuration file. Forcing users to do that, or to use some sort of translation tool, would simply alienate them, due to the additional steps this requires. Suddenly, the automated tool they were trying to use has become a headache in its own right.

Instead, BSACI uses a human-readable configuration file, modeled after the Postfix[9] configuration file format. This format uses a simple key/value pair mapping, with one entry per line, and allows comment lines to be interspersed between key/value mapping lines. This format lends itself well to in-line documentation of all configuration options, and has proven itself to be popular among system administrators (who are by nature rather picky about these sorts of things).

3.1.3 Include useful tools for development and maintenance

Cluster installation software such as xCAT and OSCAR already provide users with a broad range of features. Although BSACI will provide better automation than either of these cluster installers, it must also provide a compelling feature set which will place it apart from other cluster installers.

The need for a unique and compelling feature set drove the inclusion of several advanced tools which provide cluster monitoring and management, parallel development libraries, and distributed file systems. These tools offer compelling features to users because they go beyond simply providing the framework for clustering and parallel development. Instead, they build upon these framework components, and provide advanced, useful services to users, in the forms of higher-level data abstractions which can be used to tackle tough problems without having to bother with low-level cluster programming mechanics. Some of the tools included with BSACI are:

- The Parallel Virtual Machine (PVM)[10] message-passing library and toolkit
- The OpenMPI[11] message-passing library and toolkit
- The MPICH2[12] message-passing library and toolkit
- The OpenPBS[13] portable batch processing system tools and utilities
- The Global Arrays[14] distributed data abstraction toolkit
- The Hierarchical Data Format (HDF5)[15] distributed data abstraction library

and API

- The beosh[37] distributed shell
- ClusMon[36], a stable, high-performance tool for monitoring, reporting, and alerting based upon cluster hardware health
- The Parallel Virtual Filesystem (PVFS2)[16], a filesystem optimized for high throughput on parallel compute clusters
- GFarm[17], a fault-tolerant distributed grid filesystem

3.1.4 Plan A: use Amit's add-ons

The scope of BSACI is ambitious and relies upon many prerequisite packages, which may be a problem for some users. As a result, one option for deployment of the cluster installer is to bundle it with a collection of prerequisite software, perhaps as an add-on CD or DVD with an installation script. A user would then perform a full installation of the base Fedora Core operating system, followed by installing all software packages on the add-on media. After these prerequisites are installed, the BSACI tool is launched, setting up clustering software and pushing out system images to all the compute nodes.

This approach has the obvious advantage that it makes life easier for the user who is installing the software, once he or she goes through the effort of acquiring *both* the add-on software bundle and the cluster installer (which could be a multi-gigabyte

download). It also means system integrators can quickly and easily build a known-good collection of software to bundle with the cluster installer, and they can easily include customized software packages if necessary.

However, the approach also has a number of disadvantages. It increases the maintenance dependencies of the cluster installer, because it now must track with new builds of the Linux distribution *and* the add-on software bundle, instead of tracking only the new Linux distribution. It also opens the possibility that the cluster installer package could become dependent upon an add-on software package which is non-Free. Such a dependence would be at odds with the "Free Software" design goal of BSACI. Finally, any bugs the system integrator introduces into their add-on collection will also be introduced into the cluster installer software base.

3.1.5 Plan B: make a standalone tool

An alternate approach is to make BSACI a standalone tool, only dependent upon the user to install Fedora Core before starting the cluster installation process. All required software packages would be included in the BSACI software bundle. Instead of trying to merge configurations with "stock" software already on the system, the installer would include isolated copies of needed software (web server, database server, Java runtime environment, etc.) which are specific to the cluster installer software.

The advantage of this approach is that it makes future maintenance of BSACI easier, as only changes to the Linux distribution and any custom software packages

must be tracked and tested. The size of the BSACI download will increase, but the overall amount of data which the user must download will decrease, since only cluster-specific software will be downloaded, instead of a complete collection of many add-on packages. This approach also protects the project from becoming dependent on non-Free software, and it allows the included tools to utilize known-good versions of supporting software, even if the software shipped by the distribution changes in the future.

One disadvantage of this approach is that system integrators will have to do more work on their add-on software package collections, because they may introduce incompatibilities between their add-ons and the packages the cluster installer bundles. It also means that some software and configurations which were part of the add-ons will now have to be re-implemented as part of the cluster installer. The code reuse principles of good software engineering dictate that this should be avoided if possible. Finally, this approach uses additional disk space to store duplicate copies of supporting software packages such as the web server, database server, and Java runtime environment.

Another potential pitfall of this approach has to do with the licensing of certain software components and subsystems which are dependencies for some of the included tools. We must be careful not to distribute precompiled and linked software in a manner which may violate the terms of its license agreement. (As it turns out, there are only two software packages which have the potential of causing problems, but both

of these packages have licensing workarounds which we may employ. Sun Java™ was recently licensed in such a way as to make it easy to distribute to end users as part of a package distribution. SZip, a compression library, allows for license-free use, although some functionality is disabled. An SZip license is required for commercial and non-research users to legally access high-performance SZip functionality.)

3.2 The YACI Framework

Because of its straightforward code and no-frills implementation approach, the YACI framework was chosen as a starting point for code enhancements as the BSACI software was implemented.

3.2.1 The Role of the Master Node

YACI splits all systems up into two groups: a single *master node*, and one or more *compute nodes*. Compute nodes can specialize in particular areas, such as cluster I/O, raw CPU power for computations, etc. However, for any cluster, there exists only a single master node. This master node is the system which the automated cluster installer deals with almost exclusively.

Typical cluster architecture is such that all compute nodes reside on a high-speed, low-latency private network that is isolated from the rest of the world. The master node also resides on this network, but it is unique because it also has a second network interface that is connected to "the rest of the world". Therefore, all traffic to and

from any system in the cluster must traverse the master node, which acts as a router for this traffic, and may optionally act as a firewall and network address translation (NAT) device. Compute nodes are typically high-density, headless systems without a monitor or keyboard. Only the master node has a full console—it is the system which users interface with when submitting jobs to the cluster. Figure 3.1 shows a typical network layout using this architecture. Note that the master node acts as a firewall and a network address translating router, allowing the private nodes to masquerade to the outside world as though they are connecting from the master node. Connections from the outside world to the cluster nodes are not allowed.

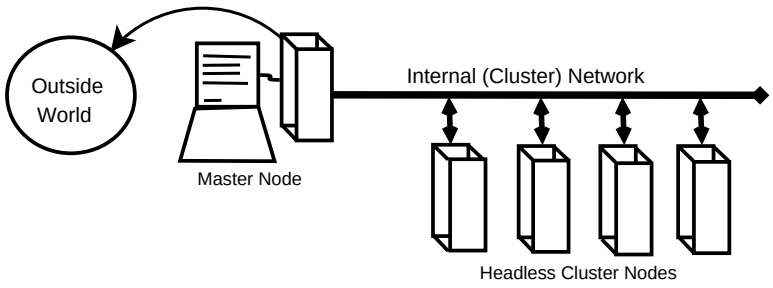

Figure 3.1.: Network design with private cluster network and dual-homed master node.

Essentially, this architecture presents the master node as "the cluster", and all the compute nodes exist simply as semi-transparent resources which may be accessed via message passing code submitted in batch jobs to the master node. The master node supports the cluster nodes with specialized services. Some of these services have already been mentioned: firewalling, NAT, and routing. Other services include DHCP

and TFTP services, batch scheduling, monitoring and reporting, remote management, user management, cluster-wide shared network filesystems, etc.

Because of the many services and associated configurations which reside on the master node, it is the focal point of the cluster installer. Each compute node must communicate with the master node at boot time in order to receive its boot configuration. Therefore, the master node knows about the presence and network configuration of each computer in the cluster. Most customized configuration files are bound to services which the master node provides, so these files are modified and stored on the master node. A few configuration files are specific to each compute node, but even these files usually incorporate data from the master node. Because the master node is central to the configuration of all cluster network services and the cluster compute nodes themselves, it is a natural choice to use it to perform as many cluster installation tasks as possible.

3.2.2 What YACI Provides

YACI provides a framework for building a Linux system image and deploying it to a number of machines, with some minor customizations. It is built specifically for use with the SuSE, Red Hat, and Fedora Core distributions, although it could work with any distribution which uses RPM as its package management system. The master node utilizes standard tools and scripts to build a complete Linux system image in a `chroot` environment. This environment is then archived into a single tarball file

prior to deployment.

YACI provides a PXE boot environment which allows compute nodes to boot, acquire a network address, format and partition hard drives, and begin downloading and unpacking the contents of the system image tarball to the local disks. When this process completes, a simple script will execute, customizing certain configuration files on the newly-imaged system. The script included with YACI is rudimentary, basically only configuring basic network settings—further functionality is left up to the system administrator to implement and test.

3.2.3 What's Missing

Other than basic network settings being customized after each node is installed, YACI does not directly provide functionality for altering the software or configuration of each node. Custom scripts must be written, tested, and maintained. These scripts are intended to manipulate any other configuration details required for a functional cluster, such as RSH/SSH security files and keys, a list of PBS nodes, setup of naming via `/etc/hosts`, NFS exports, PVM/MPI configuration, `inetd`/`ntpd` configuration, and much more.

During the deployment of the system image, YACI provides no easily accessible feedback to the user as to what is happening while the nodes install. A user at the compute node console has no way of seeing progress as YACI installs until the installation is complete and the system has rebooted. A user monitoring the installation

from the master node may have one or two options to view raw status output, but even this is primitive.

Essentially, YACI provides the basics for a cluster installer: a starting point for scripts which build, customize, and deploy the code. There isn't much else included, so it must be implemented as part of the Boise State Cluster Automated Installer.

3.3 Handling Configuration Files

The installation process must create and modify a number of data files, configuration files, and operating parameters on the master node, and several configuration files on the compute nodes. Building an automated cluster installer involves several challenges related to creating and modifying configuration files, especially when the changes to these files are inter-dependent. Maintaining a central repository for these configuration items, and keeping this repository easy to read and modify for both humans and computers was a critical early design decision.

3.3.1 The master parameters file

Since the automated nature of the cluster installer means that a computer must be able to read changes and parse/modify the affected settings, and since there are a number of files and configuration parameters which must be changed during the course of the cluster software installation, it makes sense to keep all this data in one central location. Because of the design goal stating that the cluster installer must be

```
KEY value {value ...}
# comment (entire line ignored)
```

Figure 3.2. Format of the master parameters file.

able to be fully automated, the decision was made to centrally store the configuration and settings changes in a single text file.

The file is flat text, not using any encoded formats such as XML. Entries are restricted to one per line, with each line using the format shown in Figure 3.2. Lines beginning with a '#' are ignored as comments. All other lines are parsed as key/value pairs, with whitespace separating tokens. The first token present on the line is a key identifier, and the remaining token(s) are values to be assigned to the given key. Values which must include whitespace are enclosed by double quotes. Whitespace in key values and lines beginning with whitespace are not allowed, and the behavior of the BSACI software in such situations is undefined.

This file format is easily parsed by both humans and machines (including native Linux text handling tools and shells, including `sed`, `awk`, and `bash`). It is fairly flexible in its usage, and acts as a simple and efficient means of ensuring that all configuration parameters are captured in one central location.

When using this format, parsing this file from within a shell script and extracting the keys as local shell variables becomes easy. Figure 3.3 shows some sample code which reads and accesses the data stored in the master parameters file. This ease of access from within a shell script turned out to be a critical design feature which was

```
# Read in my configuration data:
for NAME in INTIF INTIP INTNET INTMASK EXTIF EXTIP EXTNET EXTMASK; do
  eval $NAME=‘awk -vname=$NAME ’{ if ($1 == name) print $2; }’ < $CONFFILE‘
done
```

Figure 3.3.: Code which loads network parameter keys as local shell variables.

used broadly throughout the cluster installer code.

3.3.2 Scripts, scripts, and more scripts

Another key point in the design process was deciding how to actually perform the act of changing configuration files and parameters. The master parameters file was a good means of centrally storing configuration parameters as they were defined, but a means of merging those parameters with existing configuration files and creating new configuration files was still necessary.

An early approach was to use Perl[35] to parse and modify configuration entries, because of Perl's strong feature set—it was practically designed for this type of job. However, it soon became clear that the strengths of Perl were also its weakness. Code written in Perl is often brief, difficult to read, and even harder to maintain. The idea of an entire installation program written as a monolithic Perl script became something of a nightmare when looked at from a future maintainability perspective.

Other ideas were considered and rejected for various reasons: a Java application, possibly with a GUI front end, a C/C++ application, linked with the GTK+[34]

or Qt®[33] graphical toolkits, a collection of small Perl scripts operating under one master program.

Eventually, the age-old Unix philosophy of "do one thing and do it well" won out. It was decided that simple `bash` shell scripts, augmented by native text-processing tools, would do all the work. These scripts would be loosely coupled, either by sharing a common configuration file (the master parameters file), or by passing data using standard input/output pipes. Thanks to easy parsing of the master parameters file from within a shell script, implementing this approach was natural and easy to implement.

3.4 Modularity

Although early designs considered the cluster installer to be a single monolithic application, it soon became clear that any monolithic application would be subject to rapidly increasing complexity, leading to code which was difficult to maintain. Instead, a two-tiered approach was decided upon. In this approach, a few high-level applications coordinate the actions of many independent and mostly self-contained lower-level applications.

The approach works well, as each task which is to be performed by the cluster installer is generally split out into simple steps, and each step is implemented in a separate shell script. These shell scripts are independent, only relying upon the presence of the master parameters file for some of their configuration data. As a result,

a change to the way any configuration task is processed (or the addition/deletion of a configuration task) only means rewriting a single, short shell script, and modifying how it interacts with the higher-level script and the master parameters file. Such a change can be implemented and fully tested in under one hour, touching at most only one or two dozen lines of code for all but the most extensive edits.

Keeping these tasks modular is a huge advantage for ongoing maintenance, because each new release of the Fedora Core distribution typically changes how several BSACI configuration processes must operate. While a monolithic application may require extensive rewriting and testing to incorporate these changes and ensure they don't affect any other parts of the code, the loosely-coupled design BSACI employs requires much less effort, and naturally keeps changes in one task from affecting other tasks.

3.5 Speed and Performance

Although a system designed using a collection of shell scripts will not be inherently speedy, there were several design decisions which drastically altered the speed and performance of the cluster installer as perceived by the end user.

3.5.1 A watched pot never boils

It might be better to say "no babysitting required." As anyone who has waited for a software install to complete knows, the seconds a user must spend waiting for the computer to complete a non-interactive task (a.k.a "babysitting" the computer) seem

to stretch into infinity. Even if the wait is only a few seconds, to the user the time spent waiting seems interminable. Therefore, it was an early design decision that user waiting should be minimized, and all user-interactive tasks should be grouped together at one point in time.

In this approach, the user would be asked up-front to perform many interactive tasks, then the computer would batch all the non-interactive tasks, running them after the interactive tasks completed. This eliminates any waiting the user might experience between tasks, if the non-interactive tasks ran interspersed with the interactive tasks. While the batch tasks are running, the user is free to focus on something else (such as getting a cup of coffee, or catching up on news), and may return at his or her leisure to complete any final interactive tasks that come after the completion of the batch processing.

3.5.2 Use the cluster when possible

A key axiom throughout development was to recall that the software was running on a cluster. Therefore, if possible, commands which affected each node should be run on those nodes, and not on the master node. This was important, even if it meant that the associated installer code would be more difficult to implement—having the master node perform the same task 1,000 times for a 1,000-node cluster does not scale well, especially when the installer could instruct the master node and each compute node to perform the task once time.

This approach improved deployment speed significantly when generating SSH host keys for each node, for instance. The process of generating SSH host keys is processor intensive, and takes several CPU-seconds even on a fast processor. The master node must know the public keys of all nodes prior to completing the installation, but it would take hours of CPU time if the master node had to compute the keys of each node for a large cluster. Instead, the code was rewritten so each node computes its own SSH key when it first boots, and transmits that information to the master node. The master node waits until all keys have been received, then proceeds with the installation. Coding this method was more complex than simply having the master node compute all SSH keys, but it is infinitely more scalable.

3.5.3 Employ parallelism

One of the most exciting (and most time-consuming) design decisions was choosing to employ parallelism during the process of gathering user-provided configuration data. Initially, the cluster installer walked the user through a series of configuration questions, asked the user to reboot all compute nodes, then finished its configuration and built a tarball of the entire compute node's filesystem.

While this was a reasonable approach, there were inefficiencies to it. For example, the process of building the node tarball is processor and disk intensive, and might take ten minutes on a modern machine. However, before the tarball is built, the user must spend several minutes answering questions and rebooting cluster nodes. During

this time, the master node is essentially idle. Was it possible to employ this idle time to pre-build the tarball, so that it was ready to deploy as soon as the user finished answering the configuration questions?

After an intensive rewrite of the code and the creation of several patches to the YACI framework (which wasn't designed for any sort of parallel tasks), the answer was "yes." As soon as the user answered a few basic questions, the node tarball would begin being built in the background. The entire time the user was rebooting cluster nodes and answering final configuration questions, the tarball was silently being built, so that intervening time period was not wasted. If the detection of all compute nodes took a long time, when that step completed the node tarball would already be built. Instead of sitting back for another ten minutes waiting for the tarball build to complete, the user could immediately proceed to the installation of the tarball on the compute nodes.

This design decision easily cuts the amount of time the user must wait in half, and eliminates one more "babysitting" step, making the process smoother and more fully automated.

3.6 Graphical User Interface (GUI)

A critical component of the cluster installer is its graphical user interface. One of the design goals of BSACI is that it be easy for novices to use. Providing a GUI for less-experienced users gives them a sense of security, and reduces the apparent

complexity of the installation process.

3.6.1 Self-contained Apache and PHP

One of the first decisions in the design of the cluster installer was the choice of how to present the GUI to the user. Several options were considered, including native GTK+ or Qt® graphical applications, a Java/Swing application, a web-based Java applet, or an HTML-based web application. In the interest of keeping the installer flexible, it was decided that the installer should not require anything more than a text-based console on the master node. Therefore, native graphical applications were ruled out. Allowing users to use a GUI to configure the master node from the comfort of their own desks was a desired feature, and requiring users to install a Flash, Java, or similar third-party module in their browser was not desired. The decision was made to use an HTML-based web application for GUI configuration, since this is the most broadly available GUI platform.

Because the process would be dynamic and interactive, the PHP[32] scripting language was selected for implementation of back-end logic. PHP is naturally coupled with the Apache[31] web server, so Apache was used as the platform on which the PHP code was deployed. One significant problem was encountered because early versions of the cluster installer relied on the packaged versions of Apache and PHP included with the Linux distribution. When the distribution changed, so did the versions of Apache and PHP which were bundled, and the resulting changes required

significant rewrites to portions of the cluster installer code.

To avoid this scenario in the future, the cluster installer was built using a self-contained installation of Apache and PHP, separate from the bundled version provided by the Linux distribution. This self-contained Apache instance is compiled statically with a fixed version of PHP, executes within a subdirectory of `/tmp`, and exists only during the installation process. As soon as the cluster installation completes, it is removed.

3.6.2 Asynchronous Javascript and XML (AJAX)

To provide a rich environment that is similar to a native desktop application, the web interface uses a group of technologies collectively dubbed "AJAX." AJAX relies on background Javascript code to update elements on the web page in real time, providing a highly-interactive web application for the user.

The cluster installer takes advantage of this interactivity by instantly sanity-checking many user-supplied values as they are typed in, providing real-time feedback to the user when values do not pass the sanity checks, and displaying real-time status updates to the user as background processes complete.

AJAX allows the user to get away from the use of static web forms which require the clicking on a "submit" button, then waiting for the server to accept or reject the input data. Instead, data is sent to the server and checked in real time, error messages are displayed in real time, and the "submit" option is not even available until the

input data is valid. On a smaller scale, this is another way of saving the user time by not having to submit a form and then wait for feedback from the computer.

3.6.3 The xajax utility library

One gem which was discovered during the development of the AJAX web interface is the `xajax`[30] utility library. This library is a collection of PHP code that provides a simple, robust API for modifying portions of a web page using AJAX technology. `xajax` handles the low-level manipulation of Javascript code and callbacks, so the developer only needs to know a handful of PHP function calls in order to begin using AJAX technologies. It is under active development, and its performance and available features continue to improve.

3.7 Command Line Interface (CLI)

3.7.1 Why provide a CLI?

Since BSACI may be used by advanced users and system administrators who have no need of the graphical user interface, a no-nonsense, no-frills means of interacting with the cluster installer was required. This interface was intended for use by experts, so it did not require the same level of error checking or user friendliness present in the GUI. The cluster installer design included a simple command-line interface which requires nearly zero interaction from the user. This allows advanced users to enter their user parameters ahead of time, prepare their master node, then fire off the installation

script as a batch job. It also allows users who do not have access to a web browser, or for whom the web-based installation doesn't work, a second means of installation.

3.7.2 Usage Scenario

Typically, an advanced user is already aware of the parameters documented in the master parameters file, and is familiar with the documentation of how the cluster installer works. Instead of wasting time setting up a web browser and navigating the "wizard" interface, this class of user wishes to install with minimal hassle. The user edits the master parameters file by hand, inserting the correct values in it according to the installer documentation and file comments, then executes the `RUNME.sh` script that starts the installer. After going through the steps of copying data from the CD/DVD, the user enters the CLI mode by executing a script which is normally processed by the system as part of the graphical installation.

When this script completes, the user is prompted to reboot all nodes so they are detected, then a second script is launched, which completes the install process. The expert user can further automate this process by pre-populating the `MAC.info` file with the physical Ethernet addresses of each detected compute node, then writing a small `expect` script which automatically responds to the installer's prompts. Performing these final few steps would deliver a completely automated cluster installation, since all configuration data is known prior to launching the actual installation script.

3.8 Core Software

The Boise State Automated Cluster Installer provides low-level message-passing functionality and batch job management tools as part of the core set of software which is installed. These packages are considered "core" because a message-passing cluster would not properly function (or it would function poorly) without them.

3.8.1 Portable Batch System (PBS)

At the core of any multi-user compute cluster is the job scheduling system. Since most compute clusters involve long-running batch jobs, as opposed to short-lived interactive jobs, the job scheduling system typically is a batch processing job scheduler. Users submit batch jobs to the scheduler, which prioritizes the job in a queue, protects its resources and output from other users, waits for free resources and runs the job, then delivers the results back to the requesting user. OpenPBS defines batch jobs as shell scripts that include special job-control attributes, and it supports multiple defined queues with differing resource limitations. For instance, job queues can allow jobs to run only at certain times of day, or only on certain machines, or only for a limited duration (as measured by wall clock or processor time).

3.8.2 Parallel Virtual Machine (PVM)

Any cluster must have some means of spreading computation between nodes in a cluster. Parallel Virtual Machine uses message passing to enable a network of ma-

chines to be used as a distributed parallel processor. A single executable is spawned on multiple processors by PVM software, and each process tackles a smaller subtask that is part of the original problem (the "single-program, multiple-data" a.k.a. SPMD approach). Message-passing functions allow these processes to send data structures, notifications, and control messages to one another as they converge upon a solution to the original problem.

PVM supports heterogeneous networks of compute nodes, and was one of the earliest message passing toolkits to gain widespread use [18]. It is a mature toolkit with an active user base. Although newer message-passing toolkits have recently become more popular than PVM, it is still widely used.

3.8.3 OpenMPI

OpenMPI is an open source implementation of version 2.0 of the Message Passing Interface (MPI) standard. The MPI standard was the result of the joint effort of over 40 organizations meeting in an effort to define a set of library interface standards for message passing [19]. The MPI standard initially built upon the capabilities of PVM, providing extensions that allowed greater flexibility and an interface that could be implemented on many vendor's platforms without significantly changing the underlying communication layers or platform-specific supporting software.

MPI version 2.0, formalized as a standard in 1997, went well beyond the capabilities of other message passing toolkits such as PVM, adding many new features which

members of the Message Passing Interface Forum had requested. MPI 2.0 extended the MPI standard to include process creation and management, one-sided remote memory communications, extended collective operations, external interfaces, a rich collection of parallel I/O operations, additional language bindings, and support for the bulk synchronous programming (BSP) model [20, 21].

Although OpenMPI is a relatively new implementation of the MPI 2.0 standard, it is rapidly gaining popularity due to advantages it provides over other MPI implementations. OpenMPI is open source software, using a BSD-style license, so there are no concerns over proprietary code or licensing that sometimes beleaguer other implementations. OpenMPI also combines the technologies, resources, and experience from several other MPI implementation projects, in order to built "the best MPI library available" [11]. Since it is MPI 2.0 compliant, this implementation is already feature-complete and being used by many organizations for MPI development.

3.8.4 MPICH2

Similar to OpenMPI, MPICH2 is an implementation of version 2.0 of the MPI standard. MPICH2 is also open source software, it is mature, and it is widely used. In many cases, MPICH2 is the *de facto* standard implementation of MPI 2.0. In addition to its maturity and popularity, one of the greatest advantages of MPICH2 is its well-developed parallel I/O implementation (ROMIO). Developers looking for a feature-complete, high performance parallel I/O toolkit for use on a message passing

cluster will find that ROMIO is currently the best in its class. Due to the widespread use of MPICH2, BSACI would not be complete if it did not provide this MPI implementation.

3.9 Additional Software

The Boise State Automated Cluster Installer sets itself apart by including a rich set of higher-level tools and utilities which rely on the message passing and batch job management software included as part of the cluster's core software collection. These tools make development of distributed and parallel code easier, unlock some of the advantages of massively parallel and distributed I/O, and make maintenance of a compute cluster less time consuming.

3.9.1 Data Abstraction Layers

Global Arrays This toolkit provides an efficient shared-memory programming interface for clusters of individual machines, allowing an array of data elements to be accessed by any machine in the cluster, even though the actual array elements may be stored on any cluster node. Global Arrays complement the message passing programming model, allowing the programmer to use the message passing (MPI) approach and the Global Array shared memory approach in the same program. The Global Arrays toolkit is public domain code, and is actively maintained.

Hierarchical Data Format (`HDF5`) This library provides efficient storage and I/O operations for storing scientific data in the form of data sets (multidimensional arrays) and groups (similar to C structs). HDF5 was created to address the needs of scientists and engineers working on projects which require massive amounts of data processing. It has been tuned to perform at high speed using parallel I/O and threading. HDF5 relies on a solid MPI implementation, and recommends using MPICH2 with ROMIO for the best parallel I/O performance. HDF5 also supports using `gzip` or `szip` for data compression. The former is free software, but the latter is proprietary, and released under a restrictive license. As a result, the version of HDF5 included with BSACI does not include `szip` compression support. (However, eligible users may activate an alternative version of the `szip` library which includes full compression support. Run the command `switcher szip --list` on a fully-installed BSACI cluster for more information.)

3.9.2 Cluster Filesystems

Parallel Virtual Filesystem (`PVFS2`) Users looking for a high performance filesystem which utilizes the collective storage capacity of each compute node, maximizes filesystem bandwidth, and can be accessed via message passing (MPI-IO) or native Unix filesystem semantics need look no further than PVFS version 2. PVFS2 is intended to be built and deployed with the MPICH2 message passing library and the ROMIO parallel I/O implementation that is part of MPICH2, so it introduces

a dependency on these lower-level tools. PVFS2 is open source, mature, and in widespread use by many institutions and researchers. Partly due to its widespread use and maturity, it also has relatively good documentation.

By designing PVFS2 to operate in a manner similar to NFS (as a stateless filesystem and without the need for explicit file-level locks), file contention and node synchronization bottlenecks are removed, allowing for highly scalable parallel performance. PVFS2 further increases performance by striping data across all nodes in a cluster, opting for maximum parallel saturation of available network bandwidth in lieu of adding fault tolerance to the filesystem. As a result, PVFS2 is extremely fast, its speed continues to scale as the cluster size increases, and it can present very large storage capacities to users. However, the loss of a single participating storage node will render the data stored on that node unavailable until it is brought back into service. Therefore, PVFS2 may be classified as a high performance, scalable, non-fault-tolerant network filesystem.

GFarm Grid Filesystem The author discovered the GFarm Grid Filesystem while researching methods of adding reliability and fault tolerance to PVFS2. GFarm is a shared, distributed cluster filesystem that attempts to realize scalable I/O bandwidth, parallel processing, and NFS-style operation, similar to PVFS2. However, the comparisons end at this point. GFarm can only be accessed using Unix filesystem semantics, although the current implementation offers several means of presenting the filesystem to users.

The key advantage of the GFarm filesystem is its use of multiple replicas to address the need for performance and reliability. By storing multiple data replicas on several compute nodes, the loss of a single node can be tolerated without loss of data. Similarly, highly parallel use of available network bandwidth can be attained by requesting different pieces of a single file or data block from multiple replica servers in parallel. Distributed bandwidth usage can be optimized by ensuring data is replicated to the storage nodes which are closest to the compute nodes and/or users which are using this data.

The cost of adding reliability and fault tolerance is a reduction in the amount of disk space available for use across the cluster. Other disadvantages of GFarm are its relatively sparse documentation, and the fact that it is not currently in widespread use.

3.9.3 Cluster Management Tools

Parallel Distributed Shell (`pdsh`) The `pdsh` distributed shell is a must-have tool for anyone looking to manage a number of similar or identical Linux systems, whether they are in a compute cluster, a web server farm, or a computer lab. Written at Lawrence Livermore National Laboratory as a high performance, multithreaded parallel shell, `pdsh` lets a system administrator enter a shell command which is then executed in parallel on a number of machines. A new thread is spawned for handling the connection and command execution of each destination system, and standard

remote-command protocols `rsh` or `ssh` are used for remotely executing the shell command. The output and error streams from each machine are captured and reported to the user, enabling a quick review of the results from executing the command on all systems in the cluster.

Once a computing cluster is fully configured and operational, there are very few reasons to perform individualized updates on each machine—most updates and changes apply uniformly across all systems. `pdsh` allows cluster administrators to save time by issuing a change or update command only once, and having the command execute in parallel across all systems (or a selected subset of systems) in the cluster.

Distributed Cluster Shell (`beosh`) `beosh` takes the parallel approach of `pdsh` one step further, and also provides another unique approach to utilizing cluster resources: it can operate as an explicitly parallel shell, much like `pdsh`, or it can operate as a distributed shell.

In parallel mode, `beosh` goes one step further than `pdsh`. It allows users to execute commands in parallel across a set of systems in the cluster, but instead of issuing them one at a time, `beosh` provides a fully-interactive shell environment that executes every command in parallel across all selected systems. Feedback from all systems is collected and presented to the user in a fashion similar to `pdsh`.

However, `beosh` has a distributed mode which takes a novel approach to the task of fully utilizing cluster resources. In this mode, the user is presented with a fully-interactive shell environment, each shell job is broken up into individual processes, and

each process is assigned to a different cluster node to execute. In this way, a series of processor or I/O intensive tasks that would normally execute on one system is distributed, taking advantage of the additional resources available on other machines in the cluster. For example, `beosh` in distributed mode would break up the command `cat FILE | grep xyz | sort |uniq | wc` into five separate tasks, distributing the tasks as shown in Figure 3.4.

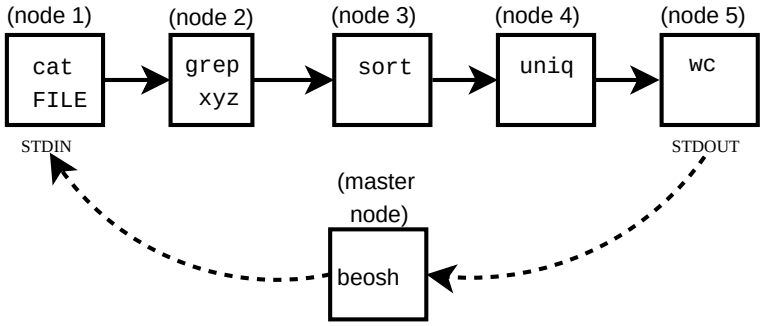

Figure 3.4. How `beosh` distributes tasks.

This distributed mode of operation is useful in cases where the user or system administrator needs to perform a series of processor or I/O intensive tasks which sequentially depend upon one another (typically in a pipeline). Although a near-linear speedup is possible if all tasks in the pipeline are processor-bound, in practice users should see a very modest speedup, since usually only one or two processes in a pipeline are processor-bound.

The Cluster Monitor (ClusMon) No computing cluster would be complete without including provisions for ongoing monitoring and status reporting. At some point, monitoring the cluster's health and utilization becomes a necessary and important task, as more users rely on the uninterrupted availability and performance of the cluster. A cluster system administrator may wish to proactively be made aware of impending problems before they result in a loss of service, thus providing a more robust computing environment for the cluster's users.

For example, the system administrator may wish to be informed of a node which is experiencing abnormally high operating temperatures. If it went undetected, this condition could lead to the sudden failure of the compute node, and the failure would disrupt any users relying on that node—currently running jobs would be lost, and any data stored on that node would be unavailable. Another scenario is a cluster which is beginning to see increasing utilization over time. This may be indicative of heavy cluster usage, meaning additional compute nodes should be added, or it may mean that there are some runaway processes which are taking up CPU time and should be killed. In either of these situations, if the system administrator has a tool which monitors and reports on system health and utilization, keeps track of historical trends, and sends alerts when the values go out of normal operating thresholds, then he or she may proactively research and fix problems before they trigger a failure.

ClusMon is just such a tool. It interfaces with the Linux kernel and the `lm_sensors` hardware monitoring toolkit to collect information about hardware health, system

utilization, and resource availability. A small subset of the data ClusMon collects includes: system temperature and fan speeds, processor, disk, and memory utilization, system uptime, and network utilization. ClusMon presents real-time updates of this data on a web-based graphical console, and stores historical data in a local database for future reference and reporting purposes. If current sensor readings fall outside predefined limits, ClusMon will take immediate action, paging or emailing the cluster system administrator. For less severe situations or trends, the system administrator can review historical data and charts, looking for trends which should be watched or researched. In this way, the cluster system administrator can stay ahead of most failures, being proactive about correcting issues before they cause a node failure, instead of reacting to the emergency situation when a node fails.

ClusMon is architected so it can monitor any group of Linux machines, not just a high performance computing cluster. Its author states that it has been designed to be robust and efficient, in terms of processor utilization, memory and disk consumption, and network bandwidth.

Chapter 4

USING THE BOISE STATE AUTOMATED CLUSTER INSTALLER

4.1 Prerequisites

4.1.1 Supported Processors

Prior to installing software on the cluster nodes, the cluster hardware must be installed and data interconnections must be established. BSACI is designed to accommodate commodity computing hardware, which means the compute nodes must be commodity systems utilizing Intel architecture 32-bit processors and typical on-board or off-the-shelf hardware for interfacing with network and disk storage devices. In other words, the cluster nodes should be a collection of personal computers or PC-class servers, without any exceptionally fancy hardware. Although 64-bit processors are becoming more common, the most commonly-available processors remain the IA32 variety, and is it these processors which are most likely to be at the heart of any cluster built of commodity components.

Although any collection of systems with IA32 processors may be used to build the cluster, it is generally a best practice to have some similarity between the master

node and the compute nodes. If possible, all compute nodes should have the same number and speed of processor—this helps the job control system ensure a uniform distribution of work between all compute nodes. The master node should have at least as much processing power as the compute nodes, and if possible, more processing power, since the master node has more concurrent processes to attend to: it is the primary point of user interaction, and ultimately all running parallel jobs will report their results back to the master node.

4.1.2 Supported Disks and Cluster I/O

Although current clusters focus largely on maximizing available processing power, cluster I/O is beginning to come into focus as a key bottleneck in current designs. As parallel problems become larger and more complex than local compute node memories can handle, cluster I/O becomes the limiting factor in cluster performance. Although a commodity cluster does not have the specialized components available to provide maximal I/O throughput, recent cluster filesystems such as PVFS2 can take advantage of parallelism to provide excellent throughput using commodity hardware.[22]

Local Storage Most commodity computers will include local data storage, in the form of one or more hard drives mounted as part of the system. These hard drives typically use the Intelligent Device Electronics (IDE) interface, although some drives use the Small Computer Systems Interface (SCSI). Because most high-performance compute clusters are processor-bound, and not disk I/O bound, the performance of

these devices is not critical. However, some cluster applications use large amounts of disk storage as a substitute for physical memory, so the disks used must not be too slow. Typical modern IDE disks have a throughput of 30-50 megabytes per second, which is adequate for general cluster usage and experimentation.

Local RAID Disks If greater local disk throughput is desired, a commodity IDE or SCSI RAID controller may be employed, so long as the RAID controller is supported by the Fedora Core Linux distribution. During the partitioning step of BSACI, the appropriate disk device names will need to be changed to match the Linux device names presented by the RAID controller.

Shared Storage Shared storage, such as a storage area network (SAN) or network attached storage (NAS) installation, is not supported by BSACI, although more experienced Linux system administrators may certainly employ it. To get BSACI to utilize SAN or NAS storage, the system administrator will need to manually intervene during the cluster installation. For a SAN, host bus adapter drivers must be installed on all systems accessing the SAN, disk storage must be allocated and mapped to each system, logical disks partitioned, formatted, and mounted, and the `/etc/fstab` and associated files edited appropriately. For NAS, filesystems must be allocated and mapped, and the associated files and mount points modified to record where the filesystems have been mounted.

4.1.3 The Data Network

Topology The cluster data network is designed in a similar fashion as the cluster compute nodes—both are composed of commodity hardware. Although there are various network interconnection designs, the simplest and most widely available is packet-switched Ethernet, and this is the network topology which BSACI is designed to support.

Network Interface Hardware Although any data interconnection which appears to the Linux operating system as an Ethernet networking device can be used, the most cost-effective data interconnection at the time of this writing is Ethernet over copper wire, operating at 1 gigabit per second (1Gbps). Network interface cards for 1Gbps Ethernet are broadly supported in Linux and are widely available, as are high-quality 1Gbps network switches. The only requirement that must be met is that the network interface cards selected for use be supported by the Fedora Core distribution of the Linux operating system.

It is critical that the network cards and systems used as compute nodes support the Intel Pre-Boot Execution Environment (PXE) [38] standard. BSACI relies on the functionality provided by the PXE cards in each compute node to detect and subsequently boot and configure the operating system on each node. At the time of this writing, most network interface cards capable of operating at 1Gbps speeds typically include PXE boot capabilities, and systems which have on-board 1Gbps

Ethernet controllers almost always support PXE. If unsure, check that the system BIOS is configured to boot from the network card, and ensure the network card itself supports PXE.

Confirming Connectivity Each compute node should be connected via 1Gbps-capable cabling to a 1Gbps network switch. Although multiple switches may be used in a variety of formats, the design and analysis of Ethernet switching topologies is beyond the scope of this document. We will assume that all nodes are connected via a single 1Gbps Ethernet switch. Once connected and powered on, the appropriate network link indicators should light on all connected switch ports and network interface cards, demonstrating a low-level network link. Any network connections which do not have link lights at both ends are faulty, and should be investigated and corrected before proceeding.

4.2 Preparing the master node

Once the hardware has been set up, the first software components of the cluster must be installed and configured. BSACI relies wholly on the master node, since all software is installed there. Collections of software for the compute nodes are built and distributed from the master node to the compute nodes.

4.2.1 Installing Fedora Core

Fedora Core 5 [23, 24] is the Linux distribution which must be installed on the master node. A typical installation is to be performed, selecting defaults for most installation questions. After identifying a locale and keyboard settings, the user may be asked about upgrading an existing system. If so, users should select the option to "Install (or re-install) Fedora Core".

Disk Partitioning Advanced users may opt to create various disk partitions on the master node, but this is not required. The simplest approach is to remove all partitions on the first disk and create a default partition layout on it. Additional disks may be partitioned and mounted at a later time. The only requirements for disk partitioning are that the `/` (root) filesystem have at least five gigabytes of free space, or that a separate `/tftpboot/` filesystem with at least two gigabytes of free space is created. This is because `/tftpboot/` is where all files for node installation will be decompressed and built into node installation tarballs.

Network Devices Most cluster installations are not entirely isolated from the rest of the world. Instead, they are connected to other networks via a second network connection in the master node. For a master node which is connected to "the outside world" via a second network card, users must be sure to identify which network device name corresponds to the internal network that the rest of the cluster is attached to, and which network device corresponds to the external network. During the Fedora

Core installation, information about both network devices will be required.

Network Configuration Both the internal and external network devices should be configured using static IP addresses and subnet masks, and should be set to start at boot. The IP addresses assigned on the internal network are not critical, since they will only be visible by systems in the cluster. Most users will find RFC 1918[25] network addressing standards to be appropriate for the internal interface.

Applicable default gateway addresses for communication with external networks should be entered, along with the addresses of any domain name (DNS) servers. Finally, the hostname should be manually configured, and should consist of at least one period (.) symbol. Often, users will choose names such as "master.cluster" for the master node, to be followed later with "node1.cluster", "node2.cluster", etc., for each compute node. Other users may opt to use fully-qualified domain names, if their network administrators have assigned such names for their cluster machines.

Additional Configuration During the first boot of the newly installed Fedora Core system, several configuration options will be presented. The following selections should be made:

- **Firewall: Disabled.** The internal network should not require firewalling, since it is private. For the interface on the external network, the system administrator can manually edit firewalling after the cluster has been set up. Note that BSACI will apply a restrictive default firewall rule set to the external interface during

cluster installation. This rule set allows inbound traffic on TCP ports 22 and 81 only.

- **SELinux: Disabled.** Several cluster software components will not work if SELinux is enabled (in the 'enforcing' or 'permissive' state). SELinux must be disabled at boot time for full cluster functionality. Selecting the 'disabled' state at this point during the installation will ensure SELinux does not interfere with proper cluster operation in the future.

- **Network Time Protocol (NTP): Enabled.** The system clock of the master node will be published to all compute nodes in the cluster, and they will synchronize their system clocks with the master node's clock. It is a best practice to keep the master node's system clock synchronized with the true time of the outside world. The NTP server(s) configured must be reachable by the master node. If the master node is isolated, then NTP can be disabled, but this is not an optimal configuration.

- **System User(s): At least one non-`root` user should be created.** Several cluster-oriented tools, such as the Portable Batch System, will not run as `root` (UID 0) for security reasons. A non-`root` user account must be created for using these tools.

After these configuration options have been selected and applied, the installer may recommend a reboot of the system. Once the system has finished rebooting and

is displaying the user login prompt, the next steps of the cluster installation may proceed.

4.3 Installing the cluster

Once the prerequisites are complete and the master node is configured, it is time to begin installing the cluster software itself. The Boise State Automated Cluster Installer is designed to be as close to a "single-click" installation as possible. The entire script is triggered by mounting the BSACI installation disk and executing a single shell script.

4.3.1 The initial `RUNME.sh` setup script

Note that a user must have `root` user privileges to install the cluster software. Users must log in as `root`, insert the media into their CD/DVD drive, and mount it. There are two ways to proceed at this point. Users may choose between the two installation methods either because of personal preference, or technical limitations.

Executing directly from CD/DVD If hard disk space is at a premium and the user has convenient local access to the master node, installing directly from the CD/DVD installation media may be preferable. In this case, the `root` user must mount the media in such a way that executing scripts are allowed. A typical command, assuming the CD/DVD drive is named `/dev/cdrom0` is:

```
mount -o ro,exec /dev/cdrom0
```

A user executing this command should successfully mount the installation media in a manner which allows executing of scripts. A successful mount can be confirmed be executing the command

```
mount | grep cdrom0
```

and looking for the device `/dev/cdrom0` and its associated mount point to appear. Assuming the media is mounted in `/mnt/media/cdrom`, the user would then start the installation process by executing the `RUNME.sh` script located in the root directory of the installation media. This can be done by issuing the command

```
/mnt/media/cdrom/RUNME.sh
```

Executing from local filesystem Some users may not wish to install directly from CD/DVD media, opting instead to copy the BSACI files in their entirety from the installation media to a directory on the master node's filesystem. This provides faster overall access to installation files, and is more convenient for situations where the master node is located in a remote machine room or similar location that makes insertion and removal of physical media inconvenient. A user must ensure they have enough free disk space to copy the entire installation media to the disk, and must ensure that the destination filesystem allows executables.

Once the files are copied, the `RUNME.sh` script located in the root directory of the installation media must be executed. If the user copied the entire contents of the BSACI installation media to `/root/bsaci/`, then the command to start the cluster

installation is

```
/root/bsaci/RUNME.sh
```

What `RUNME.sh` does The `RUNME.sh` script ensures it has basic access to read some initial files from the installation media. It then creates a generic configuration file, initializes a few values indicating what directory the BSACI files are in, and starts the first installation script, `MASTER-SCRIPT-1.sh`.

4.3.2 `MASTER-SCRIPT-1.sh`: preparing the master node

This first installation script goes through several steps which prepare the master node for later steps, including building, customizing, and deploying the node tarball. First, however, the script warns the user of the irreversible nature of the BSACI cluster setup process, and asks for confirmation before proceeding. It then reminds the user of a few prerequisites which should be satisfied before installing the cluster. If the user chooses to proceed, the script performs basic sanity checks, then installs the YACI software and several other cluster-oriented software packages which will be used during and after the installation. It also installs several Fedora Core software packages which are included in the distribution, but never installed. It then copies a base collection of RPMs which will be installed on the compute nodes, placing these files in the `/tftpboot` directory hierarchy created when YACI was installed.

Once the necessary software and files are on the system, the installation script will ensure there are no daemons listening on TCP ports 80, 81, or 3307. It will

then set up a temporary web server to listen on that port for the remainder of the cluster installation process. This web server will provide a graphical user interface to easily guide users through the remainder of the cluster configuration and deployment process.

4.3.3 Using the web wizard to enter configuration details

The web wizard is a graphical HTML/AJAX application which provides a friendly, easy-to-use interface that guides users through the configuration and deployment of the cluster software. After the first installation script finishes, the user will be prompted with a message similar to the following:

```
Congratulations, the first part of setup is complete!

The remainder of the setup process will be performed using a
web-based interface.  Please point your browser at the IP of any interface
configured on this system to continue the setup process.
(For example, http://192.168.0.1, http://172.16.2.221, etc)
```

The user may direct his/her browser to any of the addresses listed, assuming the chosen address is reachable from the browser the user is operating.

Once the web wizard page opens, all other attempts to access the page from another browser will be denied. This introduces some basic security, isolating all access to the configuration wizard to a single web browser session. The user is presented with a screen, a frame showing progress through the cluster installation steps, some configuration questions to answer, and three navigation buttons, labeled “Back”, “Help”,

and "Next". The "Help" button will pop up a window with a page of instructions specific to the screen the user is currently seeing. The "Back" button allows the user to return to a previous step, if there was one. The "Next" button allows the user to proceed to the next configuration step. It is not enabled until the user provides satisfactory information to all questions on the current step.

Part of the dynamic technology used in this web-based wizard allows each field of data to be checked as part of an asynchronous background thread in the browser, independent of what the user is currently doing. Results are returned to Javascript code in the browser, and this code determines if the "Next" button may be enabled, if the current data is invalid, or if an error message should be displayed to the user. As each field is edited and the focus moves to the next field (or the focus moves to the "Next" button), all data is asynchronously checked for validity. Once all data is valid, the "Next" button is automatically enabled, allowing the user to click it and proceed.

Once the "Next" button has been clicked, all data from the current screen is submitted to the web server for further scrutiny. It is checked again, and if the data is still valid, then the user will see a new web page. The progress frame on the left of the user interface will be updated, and the configuration data from the previous page will be saved to a configuration file on the master node's `/tmp` filesystem.

Using this interface, the user enters data about the master node's network interfaces, associated IP addresses and subnet masks, DNS servers, default gateways,

network time protocol servers, locale and time zone, and internal network DHCP pool. The user also specifies the hard disk device names, partitions, filesystem mount points, cluster network devices, and base name for the compute nodes.

4.3.4 Detecting compute nodes

After initial data collection, the web user interface initiates the process of detecting compute cluster nodes and collectively naming them. Asynchronously, as a background process, another script (`MASTER-SCRIPT-2.sh`) begins building the node tarball.

The user is asked to provide a base name for the compute nodes. This name will be appended with a unique number for each node in the cluster, thus naming each of the nodes. For example, if the base name is "worker," then the compute nodes will be named "worker1," "worker2," etc. Leading zeros are included as necessary to ensure that all names are of uniform length. If there were 100-999 machines, then the names would be "worker001," "worker002," etc.

Once the user chooses to detect compute nodes, a script will execute, showing the hardware addresses of all unique compute nodes found thus far. At this point, the user must reboot each compute node. As the nodes reboot, their PXE boot cards will send out DHCP lease request packets, searching for a boot configuration server which has not been set up yet. The master node detects each of these broadcast packets, and logs them to the screen as they are detected. In this manner, a complete list of

compute nodes is built.

As soon as the user finishes detecting compute nodes, the foreground process rejoins the background process, waiting for the node installation tarball to finish building before the next step of the process can continue.

4.3.5 `MASTER-SCRIPT-2.sh`: building the node tarball

As soon as the user begins detecting compute nodes, a background process begins the time-intensive task of assembling a tarball file which contains all the files necessary for the complete install of a compute node. While the core scripts from the YACI framework perform most of the heavy work, installing the preselected list of RPM packages into a `chroot`'ed environment, there are several configuration files which must be customized. Some of these files are customized based on data the user provided during the early steps of the web installation wizard, while others are automatically probed from the system by scripts. However, a significant number of these configuration files are based upon the number of nodes in the compute cluster, and this number is not yet known.

Instead, the complete node installation tarballs are built, minus the configuration files which are based on the number of compute nodes. Fortunately, these files can be placed on the respective compute nodes separate from the rest of the node installation tarball, although a significant portion of YACI and BSACI code had to be rewritten to allow this two-stage deployment of configuration files.

As soon as the user finishes detecting compute nodes, the user interface advances to show the progress of the script which builds the node tarball. In this way, the foreground "thread" which detects compute nodes rejoins the background "thread" which builds the node tarball.

After the node tarball is fully built and the foreground and background processes have rejoined, the final count of compute nodes is known. The remaining files which depend on knowing the number of compute nodes are built, and are compressed into a small secondary tarball for deployment after the main node tarball has been deployed. This secondary tarball is read and decompressed by each node into its `/` (root) filesystem during its first boot.

4.3.6 `MASTER-SCRIPT-3.sh`: deploying the node tarball

The third script ensures that the foreground and background processes have joined, then it restarts various system services which have been reconfigured by the cluster installation software. Once these services have successfully restarted, the user is told to begin rebooting the compute nodes. As they reboot, the nodes acquire DHCP addresses and information regarding the location of the server that contains their pre-boot environment. The nodes PXE boot from the network card, download the YACI installer, partition and format their disks, then decompress the node tarball to them.

After the node tarball is decompressed, each node reboots. This time, the PXE

environment detects that an install has successfully completed for that node, so the node boots from its local disk instead of the network boot server. This first boot after the operating system installation performs some initial setup, including generation of `ssh` keys for the node, copying over the second-stage customized files, and copying some node-specific data to the master server. Once these tasks have completed, the install for that specific node is complete. A brief message similar to the following appears on the GUI or CLI, informing the user that the installation for the named node has successfully completed.

```
Node 'node17' install & reboot completed successfully. (For more information,
see /tftpboot/logs/node17.log).
```

The master node keeps track of the number of unique compute nodes which have successfully completed their installation, and will wait until all compute nodes have reported success before allowing the user to proceed to the final steps of the installation process.

4.3.7 Final steps

On the master node, some final cleanup steps are performed. Permissions which were changed during the installation are reset to normal, and the temporary web server files are marked for deletion. The master node has a count of the number of compute nodes, and the final script waits until each node has completed its installation and reboot before continuing. Once the `ssh` key data from all compute nodes is collected,

the installer knows that all compute nodes have successfully installed and booted. It then finishes deleting temporary files, builds a list of node SSH keys (this will enable cluster users to enjoy prompt-free `ssh` sessions from the master node to the compute nodes), and informs the user that it is ready for a reboot.

When the user has rebooted the master node, the cluster is ready for use. To try it out, the user should log in using a non-privileged account, and try to compile and run a parallel (PVM or MPI) program. For convenience, the BSACI distribution includes the file `/srcs/example-code.tar.gz`, which has a collection of simple applications for PVM and both flavors of MPI implementations (MPICH2 and OpenMPI). The file `/README.example-code` in the distribution includes instructions on how to use this code, and each program's subdirectory includes a brief `README` file which lists commands to compile and run the programs. These instructions include steps for reserving nodes, starting the message-passing daemons, executing the programs in parallel, and shutting down/cleaning up the environment later.

4.3.8 Using the Command-Line Interface

The command-line interface, or CLI, shares a process similar to the graphical user interface which is described in the preceding sections. Both the CLI and GUI interfaces share a great deal of low-level code, and some intermediate scripts are the same as well. However, the code which directly interfaces with the user differs between these two interfaces.

Initial Setup The initial setup of the CLI mode is exactly the same as the setup of the GUI mode; however, when the initial `RUNME.sh` script prompts the user to

```
PRESS ENTER TO EXIT THIS SCRIPT
```

the user must instead enter the letters `CLI` and press the enter key. The script will complete a few additional steps, then present the following prompt to the user:

```
CLI MODE!
Open another shell, edit & save the configuration file,
(/tmp/BSU_CLUSTER_INSTALLER_DATA_FILE) then...
PRESS ENTER TO PROCEED WITH PHASE 2
```

Before pressing enter at this prompt, the user *must* edit the configuration file listed, and save the resulting changes.

Editing the master parameters file The master parameters file contains all details that govern how the various installation scripts will generate and modify the necessary configuration files on all cluster systems. Normally, the web-based wizard interface will sanity-check inputs and use the results to enter data into this file. However, an expert user may edit the master parameters file directly, eliminating the need to use an intermediate graphical interface.

The master parameters file is located at `/tmp/BSU_CLUSTER_INSTALLER_DATA_FILE` on the master node. The file is a plain text file, with a format as described in Figure 3.2. The various entries in this file are described in Appendix A. After a user edits

and saves the master parameters file, he or she may then proceed with phase two of the installation, by pressing the enter key at the

```
PRESS ENTER TO PROCEED WITH PHASE 2
```

prompt. Note that almost no error checking of this file is performed. It is critical that the file be error free, or the installation process must be restarted. For this reason, it is recommended that a user unfamiliar with the format and meaning of the entries in the master parameters file first attempt an installation using the graphical user interface. During the installation process, the machine-generated master parameters file may be studied and used as a template for the appropriate format when modifying the file by hand in later installations.

Identifying compute nodes Once the user proceeds to phase two of the command-line installation, another script will execute, creating and modifying various configuration files according to the data found in the master parameters file. If all goes well, the user will see a message similar to the following:

```
Generating passwordless SSH keys for users, and
running 'create_image' to create a tarball node image...
 ...
 ...
Waiting for SSH account creation to complete...
```

At this time, the master node is creating SSH keys for users, and at the same time is beginning to build the node tarball in the background. When the SSH keys are built, it is time to identify all compute nodes.

The user will be shown a brief message:

```
Getting MAC addresses...
(press ENTER when all nodes have been detected)
```

As each node is detected, its hardware address will be shown on screen. Once all nodes have been detected and displayed by the script, the user should press enter to proceed with the installation.

Building and deploying the node tarball As soon as the compute nodes have been detected, the script will re-join the background process, which has been building the node tarball in the background. The system will wait for the tarball build process to complete. As soon as the process completes, system services will be restarted in preparation for node deployment, various temporary files will be removed, and some system changes for the duration of the install will be reverted. The following message will be displayed:

```
OK. The nodes are now downloading data over NFS.  When the final node
is finished and has rebooted, you should probably reboot this server.
This installation script is complete.
```

Final steps Actually, the script is not quite complete, as it will continue monitoring the output of all nodes as they finish the installation, reboot, and perform some final processing during their first boot. The script will not exit until SSH keys and data

from each compute node has been sent to the master node, indicating that each compute node has successfully rebooted.

When all nodes have reported, the script will clean up the final leftover temporary files, set services to start upon reboot, and exit to a shell. At this point the master node must be manually rebooted. After the master node reboots, the cluster should be fully installed and operational. The user should log in using a non-privileged account, and try to compile and run a simple parallel (PVM or MPI) program to test proper cluster functionality.

4.4 Restarting a cluster installation

For various reasons, the installation of a cluster may be interrupted and need to be restarted. For this reason, a simple shell script (`/scripts/stuff/cleanup.sh`) has been provided which attempts to return the system to a relatively clean pre-installation state when run by the `root` user. Services started by the cluster installation process are stopped, software which was installed is removed, and some data directories and temporary-use files are deleted. While this tool does not return the system completely to its pre-installation state, it does return the system to a point where the BSACI tool can be run again without further manual intervention.

Chapter 5

MAINTENANCE

5.1 Maintaining the cluster

Although the subject of maintaining an operational Linux compute cluster could fill an entire book, this document will only devote part of one chapter to the subject. This section is not an exhaustive treatment of cluster maintenance. Instead, it presents the reader with some ideas about how the included tools can be used to make maintaining the cluster faster, easier, and more automated—all goals toward which any good system administrator strives.

5.1.1 Shared Network Filesystems

One of the most critical components to easily maintaining a cluster is the use of shared network filesystem mount points for certain parts of the master and compute node filesystems. By exporting a directory from the master node as a shared NFS filesystem, it can be accessed transparently from all compute nodes as if it were a local filesystem. This becomes especially important when the `/home`, `/opt`, and `/usr/local` directories are transparently shared across all systems.

By sharing these directories, user files and software, and custom local software installations can be accessed simultaneously by all nodes in the cluster, without the path to the file being different from node to node. This helps maintain a uniform environment across the cluster, limiting maintenance of those shared filesystems to a single point—the master node. It also simplifies the task of writing software to run on the cluster, since programmers can be confident that any files in `/home` will always refer to the same filesystem, regardless of which computer is executing their code.

5.1.2 The Parallel Distributed Shell `pdsh`

One of the most powerful tools for system maintenance is `pdsh`. With `pdsh`, a system administrator can type a single command and have it execute in parallel on all systems in the cluster, or a subset of them. The subtle power provided by this tool comes from the fact that all compute nodes in the cluster are installed as virtually identical systems. Therefore, a single command or series of commands to perform a particular task can often be executed, unchanged, on all nodes in the cluster.

Checking system vital signs, such as free disk space or memory, spawning a system cleanup script, and upgrading to newer software packages are all examples of some of the tasks which can be executed by a single unchanged command running on all systems in the cluster. The time savings of `pdsh` increase linearly as the number of nodes in a cluster increases, since each command still only needs to be executed one time for the entire cluster, instead of once per node.

Two lesser-known advantages of `pdsh` are its scalability and security when using a modern remote transport protocol. It can be configured to use either `rsh` or `ssh` as a transport when executing commands on remote systems. By default, BSACI supports `ssh`, since it has several advantages over `rsh`. `ssh` encrypts all communications between nodes, providing privacy, integrity, and strong authentication of remote systems and users. It also uses any non-privileged TCP source port for communications, meaning `ssh` has over 64,000 ports from which to choose, allowing it to scale to managing very large clusters. By comparison, `rsh` offers no privacy or integrity, and authentication which is trivial to defeat. `rsh` is also restricted to TCP source ports in the range 512-1023, meaning it cannot scale well when executing parallel commands on clusters of more than a few hundred nodes.

5.1.3 Portable Batch System (OpenPBS) tools

The Portable Batch System includes several tools which can aid a system administrator in the day-to-day operation and maintenance of a compute cluster. The primary function of a job batching system is to allow maximum utilization of computing resources, while maintaining fair access to all users and ensuring that users' jobs do not interfere with one another. Batch queuing systems sometimes require manual intervention on behalf of the system administrator to continue fulfilling their primary function. Runaway jobs, hung or "stuck" job queues, and user errors can contribute to minor maintenance headaches.

The `qsub` tool helps automate the complex task of submitting a new job to the batch system for queuing. A specially-formatted batch job script defines the executable(s) which will be run during the job, the batch system resources which the job will utilize, any environmental or command-line parameters which will be passed to the job executable(s), and other options which modify the behavior of the batch processing system on a per-job basis. `qsub` communicates with the job scheduler, registering the job in the correct queue, setting PBS-specific job and environment parameters, parsing the job script to look for additional queue directives, and telling the server what notifications to send out when the job completes.

With the `qstat` tool, users and system administrators can readily check the status and resource utilization of jobs in queue, using a variety of reporting formats to track and identify jobs and users which may be unfairly monopolizing cluster resources. Queues which are "hung" or are administratively shut down can also be identified and corrective measures taken. `qstat` tracks and reports on a wide variety of cluster usage metrics, including actual used resources and requested resources, which are invaluable when troubleshooting certain cluster performance issues.

`qalter` allows a system administrator to modify the attributes of one or more jobs from the command line. Job parameters which can be altered include the time(s) when the job is eligible to be scheduled for execution, which user account is associated with the job, how frequently the job will be checkpointed during execution, where the job's standard input/output/error streams are redirected, what resources are needed

by the job, how to alert users and system administrators upon completion of the job, the job's priority, and many others.

The `qsig`, `qdel`, and `qdisable` tools allow for control and removal of executing batch jobs. `qsig` will send a specified signal to the job leader of a batch job. `qdel` deletes one or more batch jobs after killing them with `SIGTERM` and `SIGKILL`. `qdisable` disables the addition of jobs to an existing batch execution queue, allowing the jobs already in the queue to continue executing until the queue is emptied.

Other tools which interact with the Portable Batch System operate at the node level. `pbsget` allows users to allocate computing resources via PBS, then spawn an interactive shell session which has access to these resources. Users who employ `pbsget` can take advantage of an interactive alternative to the batch-based environment which PBS typically provides. This is a boon when developing and debugging parallel code, as it allows for a rapid test/debug development cycle.

For both users and system administrators, `pbs_usage` allows cluster usage to be displayed. The total number of jobs executed, the number of wall-hours of processor time utilized, and the total hours of processor time utilized are displayed. System administrators can see a summary of cluster usage for each user on the system, while individual users can see only their own usage statistics.

Finally, the `pbsnodes` utility allows the cluster administrator to mark individual compute nodes as down, free, or offline, allowing for administrative maintenance to occur on some compute nodes, while the rest of the cluster remains active. This tool

also allows users and system administrators to see a snapshot of cluster utilization at any time, with a list of all cluster nodes and their current state (exclusively locked by a running job, free, down, or offline).

5.1.4 ClusMon

ClusMon is a project started by Conrad Kennington while he was at Boise State University. Although ClusMon has several applications, its primary purpose is for monitoring the physical health of each node in the cluster, and alerting a system administrator when problems arise. Maintaining the hardware of a compute cluster is as important as maintaining the cluster software. Arguably, maintaining the cluster hardware may be more of a demanding task than maintaining the software, since the number of potential problems in the cluster increases as the number of compute nodes in the cluster increase, while the number of potential problems caused by software only increases when more software is added.

When used to notify system administrators of malfunctioning nodes, ClusMon is critical to maintaining the cluster in good working order after the initial installation of the cluster. ClusMon continually monitors the status of node hardware temperature sensors, cooling fan speeds, and processor, memory, and disk usage. Should any of these data samples fall outside expected bounds, there is likely a problem with the node which should be investigated and repaired. System administrators are notified of the failure time, location, and type, and if warranted, other actions may be taken

(such as marking a node offline, or disabling a batch queue).

However, ClusMon is more than just a reactive alerting tool that notifies system administrators of problems. Because it stores past data samples and averages in a database, it can be used to report historical trends and to perform trend analysis to predict future bottlenecks or problems proactively. A system administrator who uses ClusMon to perform this type of proactive analysis can identify and correct potential problem areas before they advance to a "system down" situation. For example, a node which has been showing a trend that its local temperature is increasing may indicate a dirty air filter, poor air conditioning circulation, or a cooling fan which is beginning to fail. By proactively identifying this trend, a system administrator can examine the node, identify the cause of increased system temperatures, and take corrective action prior to the system crashing.

ClusMon makes real-time monitoring and analysis easy, because it includes an integrated web-based monitoring and reporting interface, including a dashboard-style real-time monitoring page. Every few seconds, every node in the cluster reports its current operating conditions to the database server, and the web interface periodically queries the database server to collect the latest sample data. These data are displayed in the form of performance gauges that indicate collective cluster memory, processor, and disk usage, as well as a chart listing the individual usage for each node. Clicking on a node brings up more detail, allowing for easy drill-down to accurately identify problem areas.

In this installation, ClusMon has been given a dedicated instance of Apache, a dedicated instance of MySQL, and a dedicated installation of the Java 1.5 runtime environment. This means that any changes to the system's versions of these packages will not affect the cluster monitoring tool, because it uses dedicated, isolated instances of them. In this case, the ClusMon Apache (web server) instance is listening on TCP port 81 and the ClusMon MySQL (database server) instance is listening on TCP port 3307. This avoids "clobbering" the commonly-used Apache and MySQL ports (80 and 3306, respectively), keeping them available for other uses.

5.1.5 Patches and Software Upgrades

Frequently, performance problems, software bugs, and security holes in system software are found, fixed, and a new version of the affected software released. A critical task for anyone maintaining a compute cluster is staying abreast of new software releases, analyzing them, and applying selected software upgrades to the cluster. In an environment where some cluster users must protect the privacy of their data from other cluster users, system administrators must ensure that software upgrades which fix security holes are applied in a timely fashion.

RPM Upgrade Packages While there are several approaches to installing updated software across multiple systems, the following approach works well for the majority of software updates in an RPM-based system such as Fedora Core:

1. Download the updated system software as RPM packages, and store the pack-

ages on a filesystem which is shared across all nodes in the cluster (for instance, `/home` or /usr/local).

2. Log in to the master node as `root`. Install the software on the master node (called `masternode` in this example, using the command `rpm -Uvh <`*`package`*`.rpm>`. Check to be sure it installed without errors.

3. Repeat the above process on a single compute node (called `computenode1` in this example), checking to be sure the installation is successful.

4. If the initial installations were problem-free, then the software update should be applied to the remaining nodes in the cluster. A command such as `pdsh -a -x masternode,computenode1 rpm -Uvh <`*`package`*`.rpm>` will complete the installation in parallel across all remaining compute nodes (except `masternode` and `computenode1`, which were already upgraded).

With any change to a production system, it is a best practice to first attempt the change or upgrade on a "test" system. If possible, perform the steps of the above upgrade on a "test" cluster first, then test to ensure the software and the cluster behaves as expected after the upgrade. If the test installation is successful, then the upgrade process may be performed on the production cluster.

Patches and Unpackaged Upgrades Some upgrades are not available as RPM packages, and must be applied in other ways. While the general approach to applying

an unpackaged upgrade is similar to the approach when applying an RPM-packaged upgrade, many details are notably different.

In this case, having a good "test" platform, preferably a small cluster, is invaluable. The system administrator must still download and store upgrade files on a shared network filesystem that is available to all systems in the cluster. After this, different steps apply.

The system administrator may need to compile source code into a binary executable or kernel module, then distribute the compiled code to all nodes. Alternatively, code may be ready for deployment, but packaged as a tarball file that must be decompressed into a specific destination directory. Or, the upgrade may consist simply of a binary or text patch to an existing file on the system. Each of these scenarios requires a different approach, and will require some manual intervention.

Often, the system administrator performs some manual steps to prepare for the installation (compiling, untarring, etc). The system administrator may also write a short script which will be executed as part of the installation (for example, a script which copies files and resets their ownership and permissions, or which applies a patch file to a directory). After this script is written, it is tested, then when ready, it is used for deployment to all compute nodes. A typical sequence for upgrading a cluster using non-packaged software might look something like this:

1. Download the updated files to a development machine which is configured similarly (or identically) to the cluster nodes.

2. Perform any preparatory tasks (e.g. unpacking, compiling, etc).

3. If necessary, write an installation script which performs additional installation actions, such as copying files to the appropriate locations, resetting file permissions and ownership, restarting services, etc. Test this script until it is functional and robust!

4. Identify the files which must be accessed as part of the installation process, and move these files to a shared filesystem that is accessible by all nodes in the cluster.

5. Log in to the master node as `root`. Perform a test installation on the cluster master node. Check for errors.

6. If all is well, perform a second test installation, this time on one of the cluster compute nodes. Again, check for errors.

7. If all is well, then install on the remaining compute nodes, using `pdsh` to execute installation commands (such as running the installation script) in parallel on each node of the cluster.

As stated earlier, with any change to a production system, it is a best practice to first attempt the change or upgrade on a "test" system. This is especially critical when the system administrator is writing new custom code for installation of the unpackaged software, since undiscovered bugs in the installation script may severely damage the cluster, especially if they are running with `root` permissions.

If possible, perform the steps of the above upgrade on a "test" cluster first, then test to ensure the software and the cluster behaves as expected after the upgrade. If the test installation is fully successful, then the upgrade process may be performed on the production cluster.

5.1.6 To upgrade or to re-install? That is the question.

As the cluster ages, newer versions of the Fedora Core distribution will be released. When these releases occur, a decision must be made regarding when to upgrade the cluster to this new Fedora Core version, and how to best perform the upgrade when the scheduled time comes.

Conservative system administrators will often choose to remain at least one full release behind the current latest Fedora Core release, to allow most of the bugs to be found and fixed prior to upgrading the cluster. However, some users may opt to upgrade sooner, so they can take advantage of newer features, better hardware support, and improved performance.

Regardless, at some point the cluster will need to be upgraded to a later Fedora Core release. When this time comes, there are two approaches: either perform an upgrade of the installed RPM packages for all systems in the cluster, or start over, wiping out the current cluster and reinstalling the master node and compute nodes "from scratch" using the newer Fedora Core release.

In-place Upgrade When performing an in-place upgrade, the system is upgraded from one release of Fedora Core to a newer release, by upgrading individual RPM packages to the versions distributed as part of the new Fedora Core release. After upgrading existing packages, any new RPM packages which are part of the new release are also installed, and any old RPM packages which have been deprecated in the new release are removed.

The process is essentially a large-scale version of the software upgrade process described in Section 5.1.5. All software packages for the new release are downloaded and saved, then each package is upgraded on all cluster nodes. New packages are identified and installed on all cluster nodes, and old packages are removed.

Note that this is a complex process, which must be attempted on a "test" cluster first. The nature of Fedora Core and the RPM package management system are such that new dependencies are often introduced, and these dependencies must be resolved by installing, removing, and upgrading other packages. At times, dependencies cannot be resolved using standard RPM tools, and software packages must be installed, removed, and upgraded by hand. When the upgrade is complete, some "cruft" (old, orphaned files that are no longer tracked in the RPM database) will typically be left over on the system, and will need to be identified and cleaned up manually.

While this is a time-consuming and error-prone process, it can work well if thoroughly tested and documented on a non-production cluster first. Custom software and user files remain in place, and need not be backed up or restored to other systems.

Wipe and Re-Install Instead of an in-place upgrade, other users opt for a fresh re-install of the cluster, using the new Fedora Core distribution. This is another time-intensive process, but it often has better results than an in-place upgrade.

Because each new Fedora Core release introduces significant changes to the operating system, including configuration file changes, the BSACI scripts may need to be modified slightly to accommodate the changes. Again, a test cluster is invaluable, since these script changes must be tested, corrected, and re-tested until they work correctly in a variety of situations. The processes and recommendations given in Section 5.2 should be followed when updating the cluster installer.

After the cluster installation using the new Fedora Core release and the updated BSACI code has completed successfully on the test cluster, it is time to upgrade the production cluster. The cluster should be fully backed up, and the backups verified for readability and restorability. This ensures that user data, custom software, license keys, and custom configuration data are all able to be restored to the new cluster after it is rebuilt.

The cluster installation may then proceed, just as if the cluster was new. Drives will be partitioned and formatted, data will be pushed out to all compute nodes, and the new cluster software will be configured. When the cluster installation is complete, the backed-up data (user files, custom software, etc) from the old version of the cluster must be restored.

Briefly, the steps for a wipe and re-install cluster upgrade are:

1. Download new Fedora Core release.

2. Update the Boise State Automated Cluster Installer to be compatible with the new Fedora Core (see Section 5.2).

3. Test the new BSACI until it works reliably and predictably.

4. Back up current cluster, and test to be sure data can be restored from backup.

5. Re-install the cluster as though it were new, using the updated BSACI code.

6. Restore user data, custom software, license keys, etc. from backup.

5.2 Maintaining the Boise State Automated Cluster Installer

As newer software packages are released and eventually completely new Fedora Core versions are released, the amount of work necessary to update a freshly-installed cluster to a fully patched and updated state will increase. This largely-manual work is time consuming, and it defeats the purpose of building an automated cluster installer as a time saving device. It also introduces the phenomenon known as "bit rot," which is the progressive decay of unmaintained software into a state of uselessness, due to untracked changes in the software and the environment in which the software is used.

To avoid "bit rot" and to remain a useful time saving tool, the Boise State Automated Cluster Installer must be maintained and periodically refreshed to ensure it tracks and integrates changes in newer software packages and Fedora Core releases.

This section of the document provides information and guidelines which are intended for use by anyone who is maintaining the cluster installer software. This is also useful information for someone who is trying to include their own custom or updated software packages as a part of the cluster installer's standard installation. The concepts presented in this section are the same, regardless of whether the new packages being included with the cluster installer are due to a new Fedora Core distribution being released, due to a number of updated software packages being required, or due to a system integrator, system administrator, instructor, or developer wanting to include some specific packages of his or her own.

The following steps will provide a user with guidelines, best practices, and typical scenarios that are applicable when updating software packages that are included with the cluster installer. In general, the instructions provided are not to be followed as verbatim steps, since each scenario will vary somewhat. Instead, the instructions are guidelines and reminders of what to look for and what to do. If certain instructions are intended to be followed verbatim, the document will explicitly state this intention.

5.2.1 Acquire the most up-to-date packages

The first step in building an updated cluster installer is to include all updated software packages, both custom packages and Fedora Core packages for the targeted Fedora Core release which the cluster installer is built against. We need to build a local mirror of the most recent versions of all system and custom RPM software packages.

1. Mirror the `release` RPMs for the current Fedora Core version. This means downloading the targeted (current) Fedora Core release RPMs from a mirror, and saving these files in a directory called `dist/base`.

2. Mirror the `update` RPMs for the current Fedora Core version. Download all update RPMs from a mirror, and save these files in a directory called `dist/updates`.

3. Create the `dist/extras` directory. If the cluster installer uses any RPMs from the "extras" repository (for example, some mathematical packages such as `blacs`, `blas`, or `lapack`), download and save the RPM files in this location.

4. Create the `dist/others` and `dist/others/master` directories. Any custom software which has not already been included in one of the above directories will go here. Packages which are to be installed on the compute nodes will be stored in the `dist/others` directory, while packages which are to be installed on the master node will be stored in the `dist/others/master` directory.

5. Remove duplicate packages from the above directories. Examine each directory, removing alternate and outdated versions of RPM packages in each directory. When this filtering step is complete, there will be only one version (the most recent) of each package, and that package will only exist in one location. (The only exception are "others" packages—these can exist in *both* the `dist/others` and `dist/others/master` locations if necessary).

The most time intensive portion of this process is the filtering step, where only the most recent version of an RPM package remains in the `dist/` directory tree, and all other versions are deleted. While this should be an automatable step, the exact naming conventions of Fedora Core RPM packages make scripting and testing a time-intensive task. While a script was built to automate this process, it was found to have several tricky bugs which sometimes resulted in keeping non-recent software package files.

It should be noted that YACI does include several scripts that also attempt to make the maintenance of RPM lists a bit easier. These scripts include:

- `/tftpboot/tools/update_rpmdir`, which attempts to merge the contents of an "update" RPM directory with an existing directory of RPMs. This could be useful when adding and updating packages in the `/tftpboot/dist/updates` and `/tftpboot/dist/extras` directories, for example. While not 100% automated, this script has good potential.

- `/tftpboot/tools/update_rpmlist`, which updates the `rpmlist` file to match the contents of the RPM repository. While this is a useful tool, the layout of the BSACI repository, being split up into several directories, makes it a bit more difficult to use.

- `/tftpboot/tools/create_rpmlist`, which turns out to be quite a useful tool if used properly. One way to get a guaranteed good build is to perform a minimal Fedora Core installation on one system, and tailor the selection of packages

installed on that system to reflect exactly what a compute node would require. Then, once all packages and dependencies are satisfied, and all updates are applied, simply run this script on that node. It will generate a list of all RPM packages installed on the system, and this list can be used to drive the YACI installer process.

- `/tftpboot/tools/test_rpmlist` is another good tool, especially when making changes to an RPM list file and needing to check basic dependency information. This script performs a "dry run" installation, solving dependencies and actually installing the RPMs in a sandboxed environment. This sandboxed environment can then be examined later, if necessary. Any obvious errors, such as failed dependencies or missing libraries, will be caught when running this script.

5.2.2 Create a list of required packages

Once all Fedora Core packages are downloaded, a subset of RPMs which will form the base installation of the compute nodes must be identified. Once identified, these RPM packages will be installed in an empty `chroot`'ed filesystem, creating a new system image, and the results will be stored in a tarball for future deployment to the compute nodes. The list of RPMs used to create this system image will be stored in a file called `rpmlist`. A sample version of this file may be found on the master node (after installing the BSACI software), in the `/tftpboot/site-specific-local` path. It may also be extracted from the `srcs/tftpboot-script-overlay.tar` tarball on

the BSACI installation media.

While there are several approaches to creating an updated `rpmlist` file, the approach we present involves updating an existing `rpmlist` file to include the latest RPM software packages, plus any new dependencies, and minus any deprecated packages. Since we already have an existing `rpmlist` file from the current BSACI release, we will use this file as the starting point for updates and modifications. The process is iterative, and generally proceeds as follows:

1. Get the `rpmlist` file from the current BSACI release. As noted above, there are a few ways to get this file.

2. For each line in the `rpmlist` file, search the `dist/` directory tree created earlier for an RPM file which is of the same name, but possibly a newer version. Create a new `rpmlist` file and copy these new RPM file names into the file, one line at a time, in the same order as they appear in the source `rpmlist` file.

3. Try to build a node image using the new `rpmlist` file, and watch for errors.

   (a) Log in as `root` on a master node which has had the BSACI software installed, run the `create_image` script, found under the `/tftpboot/scripts` directory, using the new `rpmlist` file name and a "test" node type. Here, we assume the new `rpmlist` file is named `rpmlist-new`, and it is stored in `root`'s home directory):

```
su - root
cd /tftpboot/scripts
create_image ~/rpmlist-new test
```

(b) Watch the build process for dependency errors, missing RPMs, and version mismatches. Correct these errors by identifying missing dependencies and adding the RPM files to the new `rpmlist` file, confirming RPM versions, and resolving version mismatches.

(c) Clean up any files left over from the previous build attempt:

```
rm -f /tftpboot/tarfiles/test.*
rm -rf /tftpboot/images/test
```

(d) Repeat the build process until it completes successfully. When is successfully completes, there will be a file called `test.tgz` or `test.tar` located under the `/tftpboot/tarfiles` directory.

4. Re-examine the new `rpmlist` file, removing any RPMs which are no longer necessary or which have been deprecated. This helps control "cruft" and keeps the image file size small.

5. Try rebuilding the node image again, making sure that it still completes successfully. Repeat the previous step until all unnecessary RPMs have been removed and the node image still builds successfully.

Congratulations! Once of the more time consuming parts of updating the cluster installer is getting a functional, updated `rpmlist` file prepared. Now that the file

is built, make a backup copy of it somewhere so all that work isn't lost. Further testing of the `rpmlist` file is still necessary, but the hardest part of updating the file is complete.

5.2.3 Test the new node tarball

Note that this step is *optional*. Although this step is optional, it is recommended to catch any discrepancies early on in the installation process. Although a node tarball may be built successfully, there is no guarantee that the resulting filesystem image will actually run as expected (although it usually will). Identifying and correcting a faulty node image build early on may save a lot of potentially wasted time later on if and when a node refuses to boot.

Moderate to advanced familiarity with Linux is expected: Since the installation will not be automatic, certain key files (such as `/etc/fstab`) will not have been defined by the installation, and disk partitioning and formatting must be done by hand. There also may be troubleshooting steps which require familiarity with the Linux boot process.

The general steps to test the initial build on a compute node are as follows:

1. Create a blank, formatted partition on the compute node. This partition should be large enough to store the entire node filesystem after decompressing it from the node tarball, and it should be a native (ext2 or ext3) formatted filesystem.

2. Create and format a second partition on the compute node as swap space. This

swap partition should be the same size as the swap partitions will be when the compute nodes are put into production.

3. Mount the native-formatted filesystem, and decompress the node tarball onto it. Tools such as Knoppix[28] and Netcat[29] may be useful for this.

4. Edit the `/etc/fstab` in the native-formatted filesystem, so it reflects the locations and device names of the root and swap partitions. Also edit the system name (in `/etc/hostname`) and network configuration (various files under `/etc/sysconfig/network*`) to reflect how the compute node will be configured when deployed.

5. Edit the bootloader configuration to boot the system from the new filesystem. Alternatively, use a boot floppy or boot CDROM to boot the kernel version that was included as part of the node tarball. Mount the newly-created filesystem as the root filesystem.

The intent of this testing is to get the filesystem tarball mounted and booted, then check for obvious errors which may be caused by the tarball. Things to look for include: unsupported and/or unconfigured hardware devices, critical missing system files, and version mismatches between the Linux kernel and any third-party modules. Things that are expected to fail, which should not cause alarm, include: many services, especially cluster-related services, advanced network functionality (such as host name resolution), and mounting of additional filesystems.

5.2.4 Test the BSACI configuration scripts

As newer Fedora Core releases are made, the location, format, and handling of various configuration files changes. These changes must be expected to cause some problems with the cluster installer, necessitating the need to update the cluster installer code.

There are five ways the cluster installer interfaces with system configuration files, though an update to the Fedora Core operating system will probably impact only two. The cluster installer uses several `probe-*` scripts to read the current system state and record probed values in the installer's master parameters file. Later, the cluster installer uses several `config-*` and `do-*` scripts to alter configuration files which will be deployed to the master and compute nodes. The other touch points include a script which uses system utilities to detect compute nodes, several "utility" scripts which the cluster installer uses internally, and a small collection of patches to the YACI code and configuration files. Of these, the first two (the `probe-*` and `config-*`/`do-*` scripts) are by far the most likely to be impacted by changes in Fedora Core, since they directly read and write the distribution's configuration files. We will focus on how Fedora Core changes affect the operation of these two cluster installer components.

The `probe-*` scripts These scripts rely on the `bash` shell and staple `awk`, `grep`, and `sed` text-manipulation utilities, along with few other tools, to read and parse various files. The locations or functions of these files may change when a new Fedora Core is

released, or the internal format of these files may change. Any of these changes will require matching changes to the `probe-*` scripts which attempt to read the files.

Generally, a change is minor, such as a file format being slightly altered, or the name/location of a file changing. A quick edit to either the script which reads the configuration file, or to the master parameters file which contains the path to the configuration file, will often be enough to update the script. Sometimes, however, more elaborate changes are required.

Although they are relatively mature, sometimes the text-manipulation tools released as part of Fedora Core have changes which alter their behavior in subtle ways from one version to the next. When this happens, the scripts using these tools must be debugged and analyzed, and the tool usage altered to account for this change.

In some cases, an entirely new script will need to be created, because configuration data must be generated in some format that doesn't already exist. In this case, an existing script can be used as a starting point, and the appropriate modifications to the master parameters file (entries denoting where the input and output file(s) are, and listing any probed data that is needed) must be made. The script is then tested and re-tested with varying inputs until it works reliably. It will then be tested further, as part of a complete test installation. But that will be addressed later in this document.

The `config-*` and `do-*` scripts Similar to the `probe-*` scripts, these scripts rely on the `bash` shell and standard text-processing tools to get most of their job done.

They take input data, in the form of probed data stored in the master parameters file, combine it with existing system configuration files and cluster installer data files, and output new configuration information and changes to existing configuration files. The `do-*` scripts also perform the menial tasks of copying and renaming various files so they reside in the correct locations in the master node's filesystem.

Many changes that affect these scripts are minor. The format of a configuration file will change somewhat, requiring the script which writes data into the file to be edited in order to maintain the new format. The location of configuration files may also change, requiring a quick change to either the associated script or the master parameters file to update the file location. These kinds of changes, particularly the location of configuration files, seem to happen with some frequency between successive Fedora Core releases. An individual maintaining the BSACI code should expect to perform at least one or two of these script edits each time the cluster installer is updated to keep pace with a new Fedora Core release.

As noted previously, the text-manipulation tools released as part of Fedora Core may include changes which alter their behavior in subtle ways. When this happens, the scripts using these tools must be debugged and analyzed, and the tool usage altered to account for this change.

Although Fedora Core tries to follow the Linux Standard Base (LSB)[26] for filesystem layout hierarchy, it is not, as of Fedora Core 5, LSB compliant[27]. Since part of LSB compliance involves having standard locations for system configuration files,

one can reasonably expect a distribution which is compliant to have a fixed location for configuration files. This is true of any LSB compliant distribution, beginning with LSB version 3.0, which provides strict backward compatibility. However, because Fedora Core is *not* compliant with the LSB standard, the location and format of configuration files can and does change between releases. More importantly, new configuration files and locations are created, and older files and locations are deprecated.

What this means for the BSACI maintainer is that new scripts may need to be created to handle configuration files which did not previously exist in earlier Fedora Core releases. Based on experience, this is a common occurrence. In this case, an existing script can be used as a starting point, and the appropriate modifications to the master parameters file (entries denoting where the input and output file(s) are, and listing any probed data that is needed) must be made. The script is then tested and re-tested with varying inputs until it works reliably. Further testing will occur later, as part of a complete installation.

5.2.5 Test the graphical and command-line user interfaces

After making changes to the scripts and/or master parameters file, both the graphical and command-line user interfaces should be tested to ensure that they still function correctly, especially as they interface with the newly written code. If code within the graphical or command-line user interfaces had to be added (for instance, the user

must be asked to set an additional configuration variable), then both user interfaces must be tested for proper behavior and for their ability to gracefully and robustly handle erroneous and invalid input.

Because the graphical interface is web-based, testing requires the use of a web browser. Currently, Mozilla Firefox[39] versions 1.5 and 2.0 are the primary supported browsers. If part of the changes are intended to make the web interface compatible with other browsers or other Firefox versions, then the graphical interface must be tested against each supported browser to ensure no regressions were introduced.

Basic graphical interface testing includes verifying that each page of the web based graphical interface renders correctly, that AJAX code operates properly without logging Javascript errors, and that users can use the "Next", "Back", and "Help" buttons of the user interface without confusing the web application or causing it to generate invalid output. Advanced testing and output validation includes examining the master parameters file and any other files generated by the graphical interface, and ensuring that the format and content of these files is correct. It also means ensuring that status updates and error conditions are presented to the user with meaningful messages.

For the command line interface, basic testing essentially means ensuring that any additional or changed messages still fit the screen format used by the rest of the command line interface. Advanced testing and output validation are the same as for the graphical interface: ensure that all output files are formatted correctly, that the

master parameter file is of a valid format and contains correct information, and that the user is properly informed of status updates and error conditions.

Testing the user interfaces can be a time-consuming process, simply because of the number of variables which must be tested. The maintainer should plan for a minimum of 3-4 hours per interface variation (e.g. each web browser used to test the graphical interface is one variation) for testing, and must restart testing for all variations when one test results in a code change. As a result, testing of user interfaces should only be performed once the back-end code (scripts, node tarball, etc.) is mature and unlikely to change, since any change in back-end code which affects the user interface code will require a new round of user interface testing.

5.2.6 Test other custom built packages

If there are custom (non Fedora Core) software packages which are included as part of the cluster installer, these packages should now be tested for proper behavior, and to ensure that they install cleanly on the system. The cluster installer itself relies on several custom-built software packages, such as custom-packaged Apache, PHP, MySQL, Java, ClusMon, `beosh`, and OpenPBS, to name a few. Cluster maintainers may add other packages which contain software that is site or application specific. Each of these packages must be tested against a new Fedora Core build to ensure that the packages install cleanly and the installed software behaves as expected.

Should there be a problem, the source of the problem must be identified. Often,

RPM version dependencies will cause issues. Other issues may be triggered by system files and directories which have changed location or content, system tools and/or libraries which have changed behavior, and newer kernel versions. Once the source of each problem is identified, a workaround (preferably a workaround which only affects the custom package) must be found.

These workarounds will often involve repackaging the software after the fix is applied. Often the software will need to be patched and recompiled from source code. Occasionally, the only workaround to a particular problem involves changing the behavior of software which is installed as part of Fedora Core. In this extreme case, the Fedora Core software package must be removed and re-created as a custom package. Note that this can adversely impact a lot of other dependencies, and increases the future work of the cluster installer maintainer. It is a last-resort fix, and every other option should be thoroughly investigated prior to replacing a standard Fedora Core package with a custom one.

5.2.7 Test a complete installation

The critical test is the final test of a complete cluster installation. Testing the complete installation process from beginning to end assures that no errors managed to slip through the cracks, and that each updated software component interacts with the others as it should. Often, small glitches and oversights in the installation process will be discovered when testing the complete installation. These bugs can then be

corrected by adjusting the affected code and re-testing the cluster installation.

The complete installation test process should be performed at least twice, once using the graphical user interface, and once using the command line user interface. Although both interfaces share a lot of the same back-end code, there are significant differences between the code paths for the two installers once the bulk of the installation data has been copied to the master node's hard drive. Both code paths must be fully tested to ensure they behave as expected.

The process for testing the complete installation requires a "test" cluster. The general process is as follows:

1. Build the new BSACI software image.

   (a) Copy the complete current BSACI installation media to a working directory (for example, `/bsaci-new`). The following assumes the BSACI installation source media is mounted under `/media/cdrom`:

   ```
   cp -a /media/cdrom/ /bsaci-new
   ```

   (b) Compress the entire `dist/` directory (which contains all the new packages) into a tarball, and copy the tarball to `srcs` under the BSACI working directory:

   ```
   tar -cvf tftpboot-distro-overlay.tar dist
   mv tftpboot-distro-overlay.tar /bsaci-new/srcs
   ```

(c) Update the `tftpboot-script-overlay.tar` file with the new `rpmlist` file and any additional files from the `/tftpboot/local` and `/tftpboot/site-specific-loc` directories. Note that the new `rpmlist` file should be stored in the `/tftpboot/site-spec` directory so it will be included in the tarball.

```
cd /tftpboot
tar -cvf tftpboot-script-overlay.tar local site-specific-local
mv tftpboot-script-overlay.tar /bsaci-new/srcs
```

(d) Update the `apache-overlay.tar` file with any newly edited content for the web interface. Note that this content is stored by default under the `/tmp/bsu/apache` directory, although it temporarily changes the `/etc/sudoers` file as well.

```
cd /
tar -cvf apache-overlay.tar /etc/sudoers /tmp/bsu
mv apache-overlay.tar /bsaci-new/srcs
```

(e) If there have been any changes to the master parameter file (which is simply named `FILE`), it should also be copied to the `/bsaci-new/srcs` directory.

(f) If new YACI packages or patches are required, they should be put in the `/bsaci-new/srcs` directory, and the old packages and patch files removed.

(g) Update the Linux kernel, but note that at the time of this writing, PVFS2 depends on the 2.6.15 kernel shipped with Fedora Core 5, and will not compile with some later versions. This bug is expected to be fixed soon.

(h) All files in the `/bsaci-new/scripts` directory should be updated with the latest versions of the installation script files. New files should be added, and any files which have been deprecated may be deleted or moved to the `attic` subdirectory for historical reference.

(i) Check and update the various `README` files located in the `/bsaci-new/scripts` and `/bsaci-new` directories. These files should reflect any user-visible changes, particularly process changes, which were introduced by the updates to BSACI.

(j) At this point, the `/bsaci-new` directory tree contains an updated build of BSACI. The cluster installer may be launched directly from this location, or the files may be burned to CD or DVD media.

2. Prepare the master node on the test cluster for the installation of BSACI. Run the `cleanup.sh` shell script (located in the `/bsaci-new/scripts/stuff` directory) to ensure that extra packages and directories have been removed.

3. Install BSACI on the master node of a test cluster. If testing the graphical user interface, proceed with the web-based wizard. Otherwise, proceed with the command line user interface.

4. Follow through the installation process, identifying at least one compute node to install. Ideally, the installation should go to all compute nodes, since this will provide a more realistic test.

5. Pay special attention to the code which detects compute nodes and the code which builds the node tarballs, since these are the areas most likely to cause errors during an installation.

6. Once these steps complete and the software has been pushed out to the compute nodes, ensure that all the nodes in the cluster successfully reboot. If they do, then the cluster installation process was successful.

5.2.8 Test the final result

After the cluster installation has completed, the final test is to ensure that it all works. Often, small bugs and oversights will be identified in this step, as certain features and functionality which should be present in the cluster do not work as expected. This document does not present an exhaustive test plan, but performing the following basic tests will guarantee essential cluster functionality, and will identify any obvious oversights which need to be corrected.

1. Ensure you can log into the master node successfully as a normal user and as `root`.

2. Check network connectivity.

   (a) Ensure the master node can at least `ping` each compute node.

   (b) Ensure the master node can communicate with external systems, and that name resolution (DNS) is working correctly.

(c) Ensure the master node clock is synchronizing with external NTP sources. The command `/usr/sbin/ntpq -c peer` can help diagnose errors.

(d) From a compute node, check to be sure it can communicate with the outside world, using the master node as a gateway and network address translation device. A simple `ping` test will suffice.

3. Check basic cluster management and automation software.

   (a) Use `pdsh` to try communicating with each node. A classic test is to check the local time on each compute node (this also checks to be sure they are synchronizing their time with the master node). Try: `pdsh -a date`. Each node in the cluster should respond with the current date and time.

   (b) Use OpenPBS tools to verify that the Portable Batch System can communicate with the PBS agents on each compute node. Executing `pbsnodes -a` will return a list of all configured nodes in the cluster. The "state" entry for each node should read "`state = free`".

4. Try compiling and building a simple "Hello World" application that uses PVM or MPI. The software should compile and link. Note that you may need to execute `switcher` to switch MPI environments and libraries (e.g. between Open-MPI and MPICH2). See the contents of `example-code.tar.gz` for some simple PVM and MPI test applications.

5. Once a test program is compiled, try reserving a compute node and executing

it. Use `pbsget` to reserve interactive nodes, then run the test program, and type `exit` to release the nodes and return to a normal shell. The program should run successfully.

6. Finally, have some sample users run tests of their own, trying to simulate normal use of the cluster. If any errors are uncovered, identify their causes and what must be done to correct them.

7. Repeat the above cycle as necessary. When the cluster appears to be operating well, and any incidental errors are minor, the BSACI maintenance may be considered successful.

Once the BSACI build is fully successful, the new version may be published. Typically this involves building an ISO image of the installation media, testing a CD/DVD disk burned with this ISO image to be sure it works, then posting the ISO image for users.

5.2.9 Updating custom-compiled and embedded code

Because several of the BSACI components are embedded and isolated from the rest of the system, they may periodically need to be updated. Specifically, the ClusMon monitoring system relies on a custom-built version of Apache, compiled with a custom PHP module, and interfacing with a custom-compiled MySQL server. Clusmon also depends on Java 1.5, so a copy of the Java runtime environment is included, separate from the system's Java environment.

Why have an embedded environment in the first place? The reason these dependencies are kept isolated is simple—we know the state of the system when it is first installed, but it is quite possible that the user may wish to make changes and updates to the system in the future, and we will strive to keep the cluster-related software from interfering with these updates whenever possible. For instance, a user may decide to install a web-based bulletin board, which means they will be altering Apache, PHP, and possibly MySQL configurations. Or, they may have a Java application they want to run which explicitly requires the 1.4 runtime environment. If we did not isolate our software from the system-installed versions of this software, the user may inadvertently disable portions of the cluster software without even realizing it!

Unpacking the code

For our example, let's assume that we wish to compile an updated Apache web server. The source code for Apache is not part of the regular distribution. However, the version built for use with BSACI may be found in the BSACI source code repository. See Appendix E for details about how to access this repository. Once the code repository has been retrieved, look for the `development-tools` directory. Inside this directory are several subdirectories, listed as follows:

```
README
build-scripts/
built-packages/
```

```
notes/
other-scripts/
sources/
```

The `sources` directory contains the source code used to compile all the custom-built tools. In that directory, locate the `httpd-2.0.59.tar.bz2` tarball, and decompress it to a "work" directory:

```
cd /tmp/apache-build
bunzip2 -c httpd-2.0.59.tar.bz2 | tar -xv
```

Compiling the code

Once the Apache source code is unpacked, we need to compile it. The compilation commands used to create each package are also included in the source code repository, in the `build-scripts` directory. Copy the Apache build script into the top level of the "work" directory that we just unpacked the source code into:

```
cp build-scripts/build-apache-2.0-clusmon.sh /tmp/apache-build/build.sh
```

Once there, you may open the file in an editor if you're curious about any configuration and/or compilation settings. Otherwise, start the compile process:

```
cd /tmp/apache-build
sh build.sh
```

Installing and packaging the code

After the compilation process completes, the code needs to be installed on the system. We also need to be sure we include any externalities, such as init scripts or configuration files, which may not be installed by performing a `make install` of the code base. The involved way to do this is to examine which files and directories were included in the original tarball, compare that list to the files and directories installed by running a `make install` of the compiled code, and delete all common occurrences.

Then `make install` is run again, and the files it installs are added to a tarball, making sure that the same set of directories included in the original tarball is included when building the new one. This is a good way to build a clean package without picking up any cruft. It is, in fact, the method your author used when building his distribution packages.

Another, "cheaper" way to do this is to simply install the old package, the run the `make install` to overwrite the older files with newer ones, and finally `tar` the results. This is faster and simpler, and is the method given below.

First, locate the tarball containing the original compiled installation files for the Apache package. In our case, it is `/tftpboot/dist/others/master/httpd-clusmon.tar.bz2`.

```
cd /
tar -jxf /tftpboot/dist/others/master/httpd-clusmon.tar.bz2
cd /tmp/apache-build
make install
cd /
tar -cf /tmp/httpd-clusmon.new.tar /etc/init.d/httpd-clusmon /opt/bsaci/apache
```

```
bzip2 -9 /tmp/httpd-clusmon.new.tar
mv /tmp/httpd-clusmon.new.tar.bz2 /tftpboot/dist/others/master/
```

Note that the last step of the above process overwrites the original file. You probably want to keep a backup version of that file around for a little while, just in case there turns out to be a problem with your new packaged build.

That's it – you now have a new package for the ClusMon Apache server. To fully close the circle, re-pack the `tftpboot-distro-overlay.tar` file to include your new package:

```
cd /tftpboot
tar -cf <...>/srcs/tftpboot-distro-overlay.tar dist/ rpms/
```

This newly-repackaged file can be included as part of your standard test and installation procedure. Note that the process for compiling other code is very similar. The only (slight) difference is for the ClusMon Java runtime environment, which is included as part of the `clusmon.tar.bz2` tarball. To upgrade the Java runtime environment, the `clusmon.tar.bz2` must be unpacked, then in the `usr/local/clusmon` subdirectory, the new JRE (deleting the old one) must be unpacked. The `java` symlink in that directory must be updated to point to the new JRE, then the `clusmon.tar.bz2` tarball must be re-packed.

Chapter 6

CONCLUSIONS

6.1 What have we done so far?

The Boise State Automated Cluster Installer presents a new, more user-oriented tool for performing automated builds of Linux-based high-performance computing clusters. It builds upon existing tools and concepts, extending them to be useful to a broad range of users. It focuses specifically on being user-friendly and robust, and on integrating a set of higher-level parallel development and cluster management tools which are not integrated by other cluster installation systems.

Because of this focus, the Boise State Automated Cluster Installer is poised to become a useful tool for researchers, instructors, students, and casual users who might not otherwise attempt to build their own compute cluster. It is also a time-saving tool for more experienced cluster administrators, providing a single human-editable configuration file and a command-line based installation process.

It attempts to isolate and protect several core cluster software components from inadvertent damage by the user, allowing the user to find additional uses for the master node in the future, should such a need arise. It also presents a user-friendly inter-

face to users, allowing the rapid changing of development environments and parallel toolkits. Because it employs "multithreading" concepts during the installation process, BSACI can complete CPU and I/O bound tasks in the background while waiting for user input, which saves a significant amount of the installer's time. Finally, it provides more useful high-level parallel development tools to the users, including data abstraction libraries, management and monitoring utilities, and parallel/distributed filesystems.

6.2 Future Directions

While there are many advances which have been made by combining these various tools, there remain several areas which could provide ample opportunities for further work:

- Multicast installation support. YACI includes the tools to do this, but BSACI does not make use of it at this time. Multicast installation would save even more time, especially when installing clusters of more than a dozen or so nodes.

- Even more multithreading. It appears we could start building a generic node image almost immediately after the "stage 1" bootstrap finishes, then customize some of the files with data we've collected later on. This is in contrast to what we do now, building the node image after "stage 7". Another thought would be to pre-build the node image and distribute it as a tarball, so all that the cluster installer does is customize the configuration files. This puts more work on the

BSACI maintainer to build and test, and might restrict flexibility somewhat, but should make for a very streamlined and speedy installation.

- Better/broader web support, and less buggy web code. The code works well right now in Firefox, but it would be good to get it working in at least Internet Explorer and Konqueror, possibly Opera as well.

- Updated support for Fedora Core 6. This is a significant undertaking, not particularly difficult, but time consuming, due to all the testing required.

- Detection and support of AMD64 platform and packages – moving beyond generic "i386" and "i686" packages for the IA32 architecture.

- Improved methods of "auto updating" reliably across a running cluster. There are many tools which can help in this task, but the primary problem is that of keeping the cluster from breaking when updating, especially across Fedora Core releases. Most of this work ends up as manual testing right now.

- Addition of Kirsten Allison's parallel toolkit library (currently a work in progress). This came too late in the process to be added as a part of BSACI, but it would provide another very useful set of parallel development tools for cluster users.

REFERENCES

[1] D'Hooge, Trent. *Yet Another Cluster Installer.*
http://www.llnl.gov/linux/yaci/yaci.html

[2] SYS-CON Media. *Appro Delivers the Largest Linux Supercomputing Clusters to LLNL.*
http://linux.sys-con.com/read/300135.htm

[3] Finley, Brian. *Official SystemImager Manual.*
http://www.systemimager.org/documentation/systemimager-datasheet-1.5.0.pdf

[4] S. Dague, B. Finley, and J. Greenseid. *What is System Configurator?*
http://sisuite.org/systemconfig/

[5] Ford, Egan. *What is xCAT?*
http://www.alphaworks.ibm.com/tech/xCAT

[6] Lange, Thomas. *Fully Automatic Installer (Flyer)*
http://www.informatik.uni-koeln.de/fai/flyer.pdf

[7] The Open Cluster Group. *OSCAR 5.0.*
http://oscar.openclustergroup.org/

[8] The Open Cluster Group. *The OSCAR 5.0 Installation Manual.*
http://oscar.openclustergroup.org/public/docs/oscar5.0/
OSCAR5.0_Install_Manual.pdf

[9] Venema, Wietse. *The Postfix Home Page.*
http://www.postfix.org

[10] Geist, Al. *PVM: Parallel Virtual Machine.*
http://www.csm.ornl.gov/pvm/

[11] The Open MPI Team. *Open MPI: Open Source High Performance Computing.*
http://www.open-mpi.org

[12] World Wide Web. *MPICH2 Home Page.*
http://www-unix.mcs.anl.gov/mpi/mpich2/

[13] Altair Grid Technologies. *About OpenPBS.*
http://www.openpbs.org/about.html

[14] Computational Sciences and Mathematics, Pacific Northwest National Laboratory. *The GA Toolkit.*
http://www.emsl.pnl.gov/docs/global/

[15] The HDF Group. *HDF5 Home Page.*
http://www.hdfgroup.org/HDF5/

[16] World Wide Web. *Introduction to the Parallel Virtual Filesystem, Version 2.*
http://www.pvfs.org/pvfs2/

[17] World Wide Web. *What is the Gfarm Grid File System?*
http://datafarm.apgrid.org/document/

[18] Wikipedia contributors. *Parallel Virtual Machine.*
http://en.wikipedia.org/w/index.php?title=Parallel_Virtual_Machine&oldid=95101051

[19] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard.* University of Tennessee, Knoxville, Tennessee, 1994.

[20] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface.*
University of Tennessee, Knoxville, Tennessee, 1997.

[21] Gropp, William and Lusk, Ewing. *Why Are PVM and MPI So Different?*
Argonne National Laboratory, 1997.

[22] Mache, et. al. *How to achieve 1 GByte/sec I/O throughput with commodity IDE disks.*
http://www.lclark.edu/~jmache/sc2001.html

[23] The Fedora Project. *Fedora Project, sponsored by Red Hat.*
http://fedora.redhat.com/

[24] The Fedora Project. *Download the Fedora Project, Core 5.*
http://download.fedoraproject.org/pub/fedora/linux/core/5/

[25] Rekhter, et. al. *RFC 1918 - Address Allocation for Private Internets.*
http://www.faqs.org/rfcs/rfc1918.html

[26] Free Standards Group. *About the Linux Standard Base (LSB).*
http://www.freestandards.org/en/LSB

[27] Free Standards Group. *LSB Distribution Status.*
http://www.freestandards.org/en/LSB_Distribution_Status

[28] Knopper, Klaus. *What is Knoppix®?*
http://knopper.net/knoppix/index-en.html

[29] Giacobbi, Giovanni. *The GNU Netcat – Official Project Page.*
http://netcat.sourceforge.net

[30] White, Jared and Wilson, J. Max. *xajax PHP Class Library - The easiest way to develop asynchronous Ajax applications with PHP.*
http://www.xajaxproject.org

[31] The Apache Software Foundation. *The Apache HTTP Server Project.*
http://httpd.apache.org

[32] The PHP Group. *PHP: Hypertext Preprocessor.*
http://php.net

[33] Trolltech AS. *Qt - Cross-Platform C++ Development.*
http://www.trolltech.com/products/qt/features

[34] The GTK+ Team. *GTK+ - The GIMP Toolkit.*
http://www.gtk.org

[35] Wall, Larry and The Perl Foundation. *The Perl Directory.*
http://www.perl.org

[36] Jain, Kennington, and Mazzarelli. *Clusmon: A Beowulf Cluster Monitor.*
http://onyx.boisestate.edu/clusmon

[37] Jain, Amit and Vail, Mason. *beosh: The Beowulf Cluster Shell.*
http://cs.boisestate.edu/~amit/research/beosh/

[38] Intel Corporation. *Preboot Execution Environment (PXE) Specification, Version 2.1.*
http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf

[39] The Mozilla Foundation. *Firefox - Rediscover the Web.*
http://www.mozilla.com/en-US/firefox/

Appendix A

MASTER PARAMETERS FILE KEYS

The following table lists the name of each master parameters file key which is recognized by the installer software. A brief description of what the value represents and the data format the value should be supplied in (if not a single word) is also given.

Key Name	Description
INTIF	Device name of the internal network interface
EXTIF	Device name of the external network interface
INTIP	IP address of the internal network interface
EXTIP	IP address of the external network interface
INTMASK	Network mask of the internal network
EXTMASK	Network mask of the external network
INTNET	Network number of the internal network
EXTNET	Network number of the external network
DHCP_POOL_START	IP Address of the first entry in the DHCP pool
DNS_SERVERS	Space-separated list of IP addresses, one per server
NTP_SERVERS	Space-separated list of IP addresses, one per server
DNS_DOMAIN	DNS domain that the master node is part of
DNS_SEARCH	List of up to six DNS domains to search, separated by spaces
TIMEZONE	Time zone in continent/locality format, e.g. America/Boise
DISTRO	Name of the distribution being installed to. RUNME.sh automatically populates this.
NODE_NAME_BASE	Base name for compute nodes

NODE__NETDEV	Network device name for cluster network interface on compute nodes
NODE__DISKDEV	Device name of node disk device to partition
NODE__DISKSIZE	Size of node disk, in gigabytes
NODE__MEMSIZE	Amount of RAM in node, in megabytes
NODE__PARTLIST	List of partitions to create on the node disk, listed as a space separated series of device,mountpoint,filesystem,size,boot tuples e.g. "hda,/boot,ext2,100,yes"
FILENAME__RCSCRIPT_PATH	Location of initialization scripts (usually /etc/init.d)
FILENAME__RESOLVCONF	Location of the resolv.conf file (master node)
FILENAME__NTPCONF	Location of the ntp.conf file (compute node)
FILENAME__BASHRC	Location of the bashrc file (compute node)
FILENAME__NTPCONF_s	Location of the ntp.conf file (master node)
FILENAME__MACINFO	Location of the MAC.info file (master node)
FILENAME__DHCPDCONF	Location of the dhcpd.conf file (master node)
FILENAME__HOSTS	Location of the hosts file (all nodes)
FILENAME__HOSTSEQUIV	Location of the hosts.equiv file (all nodes)
FILENAME__HOSTSALLOW	Location of the hosts.allow file (all nodes)
FILENAME__XINETDRSH	Location of the xinetd.d/rsh file (all nodes)
FILENAME__XINETDRLOGIN	Location of the xinetd.d/rlogin file (all nodes)
FILENAME__ROOTRHOSTS	Location of root's .rhosts file (all nodes)
FILENAME__SECURETTY	Location of the securetty file (all nodes)
FILENAME__MOMPRIVCONFIG	Location of the MOM config (master node)
FILENAME__EXPORTS	Location of NFS exports file (master node)
FILENAME__NODES	Location of the PBS nodelist (master node)
FILENAME__PBSSERVERNAME	Location of the PBS servername (all nodes)
FILENAME__POSTCREATE	Location of YACI post-create script (master)
FILENAME__POSTINSTALL	Location of YACI post-install script (master)
FILENAME__VARIABLES	Location of YACI variables file (master)
FILENAME__PARTITION_LIST	Location of YACI partition_list file (all)
FILENAME__ETCMACHINES	Location of machines file (master)
FILENAME__PVMD	Location of pvmd binary (all)
FILENAME__PVFS2TAB	Location of PVFS2 pvfs2tab file (all nodes)
FILENAME__ETCPVFS	Location of PVFS2 config directory (all nodes)
FILENAME__PVFS2GENCONFIG	Location of PVFS2 config generator (master)
FILENAME__PVFS2SLOG	Location of PVFS2 server logfile (all nodes)
FILENAME__PVFS2STORE	Location of PVFS2 backing store (all nodes)
FILENAME__CLUSMON_HEADCFG	Location of ClusMon head daemon config (master)
FILENAME__CLUSMON_NODECFG_TEMPLATE	Location of ClusMon node configuration template file (compute node)

FILENAME__CLUSMON_NODECFG	Location of the master node's ClusMon node configuration file (master node)
FILENAME__CUSTOMIZED	Location of 2nd stage tarball file (all)
MASTER__SCRIPTDIR	Relative directory for executing scripts
MASTER__CDROM_ROOT	Where the CDROM root directory is
MASTER__SCRIPT_TARFILE	Name of the tarball containing scripts
MASTER__DISTRO_TARFILE	Name of the tarball containing node RPMs
MASTER__APACHE_TARFILE	Name of the tarball containing Apache content
MASTER__APACHE_FLAGFILE	Path where the Apache "in use" marker file is to be stored
MASTER__DISTDIR	Path where node RPM files are stored
MASTER__RPMLIST	Path to file listing all node RPMs to install
MASTER__POST_CREATE_SCRIPT	Alternate location for YACI post-create script (master)
MASTER__QMGR_CONFIGFILE	Path to PBS QMGR initial configuration

Appendix B

BUILD AND TEST ENVIRONMENTS

Primary testing for this software was done using a hodge-podge collection of computers. I built and tested a *lot* of BSACI code using cluster at my home. The primary test build/test cluster was in my garage, and consisted of four compute nodes and one master node, configured as follows:

- Master Node:
  - One Intel® Pentium-4® processor running at 1.5 GHz
  - 512 MB system memory
  - Two 18 GB, 7200 RPM SCSI disks, arranged in RAID-0 (striping)
  - One Intel® PRO/1000 1000Base-T (gigabit) ethernet controller, used to interface with the cluster network
  - Two on-board Intel® 100Base-T ethernet controllers, one of which was used to interface with the outside world
- Compute Nodes:
  - One Intel® Pentium-3® processor running at 667 MHz

  - 128-256 MB system memory
  - One 20-40 GB 7200 RPM IDE disk
  - One Intel® PRO/1000 1000Base-T (gigabit) ethernet controller, with PXE boot ROM enabled

- Cluster Network:
  - All gigabit ethernet over copper
  - One Netgear 8-port unmanaged gigabit ethernet switch
  - Master node performs network address translation for outbound traffic from compute nodes

Because of the limited storage of the master node and the large amount of data used when developing this project (each complete working copy of the release code could use 11-12GB just for various tarballs, system images, and collections of RPM files), additional working storage space was claimed from other systems in my home network, operating over NFS.

For mobile development and testing of relatively simple changes, I employed a 3-node virtual cluster, running under a VMWare virtual machine environment on my laptop. This allowed for fast turnarounds when testing minor tweaks and changes to RPM packaging and the cluster installation process. One key advantage of this approach was the ability to "snapshot" the virtual machine disks in a particular state.

This allowed me to install the base Fedora Core operating system on the virtual master node once, then snapshot the virtual disk in the final installed state, so I could return to this pristine environment with a single command whenever I was ready to start my testing over.

Unfortunately, VMWare is very slow at executing I/O bound processes, and most of the cluster installation process is I/O bound. VMWare also severely constrained my available memory to 96 MB per node, so by no means was VMWare a panacea for all my BSACI testing needs!

For deployment testing, Boise State University provided a 7-node test cluster called `rookery`, which consists of the following hardware:

- Master Node:
  - One Intel® Celeron® processor running at 1.7GHz
  - 1 GB system memory
  - One 30 GB IDE disk
  - Two Intel® PRO/1000 1000Base-T (gigabit) ethernet controllers, one for interfacing with the internal cluster network and the other for interfacing with the outside world
- Compute Nodes:
  - Three Intel® Celeron® systems, each with one processor running at

1.7GHz

* 512 MB system memory
* One 30 GB IDE disk
* One Intel® PRO/1000 1000Base-T (gigabit) ethernet controller

– Three Intel® Pentium-3® systems, each with one processor running at 1.2GHz

* 512 MB system memory
* One 60 GB IDE disk
* One Intel® PRO/1000 1000Base-T (gigabit) ethernet controller

- Cluster Network:
  - All gigabit ethernet over copper
  - One Hewlett-Packard ProCurve® 24-port managed gigabit ethernet switch
  - Master node performs network address translation for outbound traffic from compute nodes

This provided a realistic environment that included typical hardware for a lower-end cluster installation. It also provided a testbed where other cluster users could try their hand at the cluster, since it was on site in the Boise State Computer Science department.

Appendix C

INSTALLATION README FILES

The following are the contents of various text files which are included as part of the standard BSACI distribution. These files provide tips, guidelines, and requirements for users to allow them to get the most out of their installation. For the more technical users, these files also provide a glimpse into the inner workings of the installation software.

C.1 README

```
Welcome to the Boise State Automated Cluster Installer!

This software is designed to make some steps of cluster installation and
initial setup easier, through automation of many cluster installation
procedures.  The software is intended to be run on a single system running a
fresh install of Fedora Core 5 with all "roles" (e.g. Office & Productivity,
Software Development, Web Server, etc.) selected during the installation.
This single system will become the master node of a new cluster, and will
automatically build and push out compute node OS images to all compute nodes.

----------

Following are some guidelines and general requirements which must be satisfied
in order for the cluster installer software to function properly.  Unless
otherwise specified, each requirement is specific to the machine which
```

functions as the master node.

0. Mount the installation media (if a CD-ROM or DVD-ROM) using the iso9660 filesystem. The udf filesystem doesn't work correctly with executable files, such as shell scripts. (In some cases, Fedora won't mount ISO9660 filesystems with execute rights either. If that happens, just manually mount the media using "mount <device> /media".) When the media is mounted, run the "RUNME.sh" script to get the bootstrap process started.

1. The hardware for all nodes (the master node and all compute nodes) should be similar for best results, although it is not absolutely required that they be identical systems. The master node requires two physical network interfaces (one for the "cluster" network, one for the "public" network), while the compute nodes only require a single network interface.

2. To properly prepare the master node, install Fedora Core 5 as the operating system, with the following selections:

- When prompted about what general groups of software packages to install (e.g. "Office & Productivity", "Software Development", "Web Server", etc.), select each of the groups by ticking the appropriate check box. Do not select the option to "Customize" or "Customize Now" the package selection.
- Install Fedora Core 5 without firewalling enabled. The cluster installer will adjust your firewall rules.
- Install Fedora Core 5 with SELinux in "disabled" mode.

3. Be sure the master node has two physical network interfaces (one internal, for the cluster network, and one external, for communicating with public systems). Although the installer tries to detect interfaces with various names, it will be best if these interfaces are labeled 'eth0' and 'eth1' by the kernel. You will need to know which interface name corresponds with the internal network, and which corresponds with the external network.

4. BOTH the internal and external network interfaces must be configured with static IP addresses / subnet masks / network addresses (not DHCP!), and both interfaces must be set active at boot time. It is recommended, but not required, that the internal network be configured as an RFC1918 private network.

5. The hostname for the master node MUST be a proper, fully-qualified domain name (do not use DHCP to name your master node). E.g. a "short" name, or the default "localhost.localdomain" will not work. This is especially important

for PBS. So, make sure the hostname is a FQDN when the installer prompts you, and do not tell the installer to automatically select a machine name. You want to manually specify the name.

6. The master node should be configured with working DNS servers, in such a manner that the master node can resolve public host names correctly. It is also a good idea to configure the master node with working network time protocol (NTP) servers, since an accurate time reference is important for correlating security logs and for synchronizing some types of computations.

7. Don't forget to create user accounts on the master for all users who will be using PBS. Due to security reasons, root cannot use PBS! These accounts (in fact, *all* accounts created on the master node prior to running the installer) will be copied to each compute node in the cluster.

Other notes about the installation (probably not exhaustive):

- This setup can be run in either of two modes:
    - The default is a web-based GUI, with a small shell script called 'RUNME.sh' on the root of the installation media. This script bootstraps the web server and content, then instructs you to visit a URL hosted by the master node to complete the installation. When finished, the installer will attempt to remove all the web-based code, and will shut down the web server. The GUI requires a relatively modern browser with good Javascript support. It has been tested with Gecko-engine browsers (e.g. Firefox 1.5, 2.0). For more details, see the README.GUI file located on the root of the installation media.
    - An alternate install is completely based on shell scripts found in the /scripts/ directory of the installation media. For more details, see the README.CLI file located on the root of the installation media.

- The installation media includes a copy of some core RPMs from Fedora Core 5, plus updates and extra packages, current as of March 2007. These files are extracted to /tftpboot/dist/ during the installation process, and will remain there for future use. Updates can be applied to these directories, as long as the YACI 'rpmlist' file and the contents of /tftpboot/rpms/ are updated to reflect new packages. See the YACI documentation, including the source of the YACI 'update_rpmdir' and 'update_rpmlist' scripts for further details.

- The installation has been tested to run on a clean, freshly-installed Fedora Core 5 system with all general software groups selected (but no custom changes made). Trying to install using a subset of the Fedora Core 5 packages, or trying to install on a "dirty" system could result in unpredictable behavior. Similarly, installing to a Fedora Core 5 system which has had update packages applied may not function correctly. Any errors of this sort will typically be indicated during the installation process.

- Portable Batch Scheduling (PBS) is installed, with a "wide open" generic configuration. This should work out of the box for a dedicated cluster. If a shared cluster is being created, then the PBS configuration will need to be altered to set up limits.

- Parallel Virtual Machine (PVM) and XPVM are installed for parallel computation. These include specific customizations by Boise State faculty. Additionally, MPICH2 and OpenMPI provide MPI capabilities. These also have been slightly customized to tie in with PBS for allocating compute nodes.

- Several libraries which are helpful for parallel development are included. HDF5, SZIP, Global Arrays, and Env-Switcher are some of these tools. Most of these tools are installed under /opt, and the 'switcher' utility will allow the user to change environment variables to make use of them. Read the system message of the day for more information.

- Two parallel/distributed filesystems are included: PVFS2 and GFarm. PVFS2 is a high-throughput parallel filesystem designed for use in reliable cluster networks, while GFarm strives to be more of a distributed, fault-tolerant system. Both filesystems store their portions of the global data space in hidden directories under /tmp, so be aware of deleting any directories under /tmp with 'pvfs' or 'gfarm' in the name. Read the documentation for each filesystem to get fully acquainted with their use and features.

- ClusMon is included as a cluster-monitoring tool. This includes a dedicated MySQL server, dedicated Apache server (listening on port 81/tcp), a dedicated Java runtime environment, and some daemon processes. On the master node, the dedicated environment (under /usr/local/clusmon and /opt/bsaci) means that future system changes/upgrades to Apache, MySQL, or Java will not affect the operation of this monitoring software.
    - Note that ClusMon uses lm_sensors to monitor hardware temperature and fan speed sensors on all nodes, including the master. Because no standard naming convention for sensors exists, you may have to edit

```
            the /usr/local/clusmon/clusmon.node.config file on each system to
            ensure that the proper sensor names are being read for CPU thermal
            and fan speed measurements.
          - The MySQL instance is isolated using a dedicated port (3307/tcp)
            which will not conflict with other MySQL instances on the default
            port.  It has a randomly-generated password, which is stored under
            /opt/bsaci/mysql/etc for reference.

          - The Apache instance is isolated using a dedicated port (81/tcp)
            which will not conflict with standard web-hosting ports (80,443).
            The firewall will be configured to allow access to this Apache
            instance for remote monitoring purposes.

- IPtables is set up in a NAT configuration, using connection tracking.  The
  only inbound traffic which is allowed from the public network is SSH and WWW
  (tcp/22 and tcp/81) to the master node.  All network traffic originating
  from the private network or the master node is unrestricted, and all compute
  nodes access the network by using the master node as their default gateway.

- The master node exports /home/, /opt, /tftpboot, and /usr/local/ over NFS,
  and all compute nodes mount these filesystems locally as /home/, /opt, and
  /user/local/.  (/tftpboot doesn't get mounted, except during initial node
  installation).  This offers greater flexibility for users when writing
  software and running it on the cluster, since these three directories are
  the same on all systems within the cluster.

- YACI is configured to install node images using an NFS share of /tftpboot/.
  It is *not* configured to multicast the image.  This could be a future
  improvement to the installer software.  This is not a critical limitation
  when installing smaller clusters, which is what this tool is primarily
  intended to do.

----
_PK 2007/03/16
```

C.2 README.GUI

```
The BSU Automated Cluster Installer operates in two phases:
        - A "bootstrap" phase, which is a shell script that performs basic
          preparation
        - An installer that actually proceeds with the full installation,
          making system changes, etc.

The "RUNME.sh" script on the installation media is the bootstrap script that
gets things started, in preparation for the GUI installer.  After this bootstrap
script has successfully run, it will direct the user to a URL where the
installation can be finished with the help of a web-based GUI.

The web-based GUI tries to perform a significant amount of auto-probing, to
save some work for the user, and primarily acts as a way for the user to view
and confirm the settings which already exist.  The primary purpose of the web
interface is to create an easy method for a user to generate a configuration
file that is properly formatted for use with the automated configuration and
installation scripts.

Once this script has been created, the web interface will guide the user through
the process of detecting nodes, building YACI images, configuring and starting
the necessary services, and finally deploying the images to compute nodes. All
the technical details of the above process are hidden under the covers of the
web GUI.

---------

The web GUI requires use of a browser with good Javascript support, because it
uses Javascript extensively.  The recommended browser is Mozilla Firefox 1.5 or
2.0, although any browser based on a recent Gecko rendering engine should work
well.  Other browsers have not been tested, and particularly Microsoft Internet
Explorer is known to be broken.  A screen resolution of 1024x768 or higher is
recommended.

To start the web GUI, simply ensure that Fedora Core has been installed with all
general software "roles" or groups selected, then mount the installation media
and execute the "RUNME.sh" script.  The script will install necessary Apache
files, several YACI and PBS RPMs, then copy all Fedora Core binary RPMs to the
system's hard drive, under /tftpboot/dist/.  It will then start the web server,
```

and print out a message informing the user that the rest of the process may be completed by pointing a web browser at one of the server's configured network interfaces.

Ideally, and as recommended in the README, this software should be installed on a blank, fresh install of Fedora Core with all general software packages installed, but no customizations installed.

C.3 README.CLI

```
The BSU Automated Cluster Installer operates in two phases:
        - A "bootstrap" phase, which is a shell script that performs basic
          preparation
        - An installer that actually proceeds with the full installation,
          making system changes, etc.

The "RUNME.sh" script on the installation media is the bootstrap script that
gets things started, in preparation for the GUI installer.  After this
bootstrap script has successfully run, it will direct the user to a URL where
the installation can be finished with the help of a web-based GUI.

For users who wish to use the CLI based installer, a bit more manual
intervention is required.  However, this can allow for even further automation
of the cluster installation, for users who wish to take advantage of this
capability.  The "RUNME.sh" file is still used, and web GUI content still gets
copied to the server, but the user can alternatively trigger CLI-based actions
after the script has completed most of its tasks.

---------

The file "scripts/NEWORDER" (which is a bit out of date, but still generally
applicable) describes the general order of how the install process proceeds.
When using the CLI, the intermediate steps which would normally be handled by
the web application are eliminated.  Instead, the user is expected to hand-edit
a properly-formatted and error-free configuration file, which is then used by
the final steps of the install process.
```

The steps to perform a GUI install are as follows:

1. Mount the installation media, and run the "RUNME.sh" bootstrap script. Files will be copied to your hard drive, and paths set up.
2. IMPORTANT! When the "RUNME.sh" script asks you to "PRESS ENTER TO EXIT THIS SCRIPT", type in the letters "CLI" and press enter. This continue executing a hidden portion of the "RUNME.sh" script.
3. "RUNME.sh" will now tell you to edit the configuration file (located under /tmp), and tell you to PRESS ENTER TO PROCEED WITH PHASE 2. DO NOT press enter at this time.
4. In another terminal, open the configuration file "RUNME.sh" told you about, and edit it:
    - Whitespace is important in this file. The first word of a line is the configuration variable name.
    - The second word on a line is the value of the variable.
    - If a variable can contain multiple values, then each value is a word, a single space separating each value, and all values on a single line.
    - Lines beginning with a '#' are ignored as comments
    - Variable names must be all one word (no whitespace or quotes)
    - Variable values must be all one word (no whitespace or quotes)

4a. Make sure the DISTRO variable has a value of Fedora-Core-5.

4b. Edit all other variables at the top of the configuration file to reflect your settings and configuration.

4c. Save changes to the configuration file, then switch back to the terminal where "RUNME.sh" is still running.

5. Press enter, and the installation script will proceed with the second phase of the installation, making changes to the system.
6. During this process, you will need to boot all nodes in order, when the script is "Getting MAC Addresses". When all addresses have been recorded, use "killall tcpdump" from another terminal to end the harvesting script.
7. Later, after the node tarball has been made, you will be prompted to PXE-boot all nodes. This will actually push the operating system image out to the compute nodes.
8. Finally, the script will end, and you will be reminded to reboot the master node.

This process relies upon having a properly formatted configuration file provided to the script after the initial bootstrapping phase has completed. With a little tweaking, this can be made a fully-automated (or nearly so) process, from start to finish.

Please note that the CLI installation process hasn't received much attention since the GUI was developed. There may be some bugs that have sneaked in to screw up the above process.

C.4 README.example-code

In the srcs/ directory, there is a tarball of simple demonstration applications which can be compiled using the PVM, MPICH2 and OpenMPI toolchains.

These demonstration applications can be used as a simple primer for those who are unfamiliar with the toolchains, and also as a sanity check to ensure the cluster is operating correctly.

To use these demonstration applications, simply (as a non-root user) unpack the srcs/example-code.tar.gz tarball, switch into one of the three directories

```
   mpich2-examples
   openmpi-examples
   pvm-examples
```

depending upon the environment you wish to test, and execute "make".

If you receive an error about the environment tools not being loaded, for instance when issuing "make" in the openmpi-examples directory, you must use the 'switcher' tool to change your environment. In this case:

```
   switcher mpi = openmpi-1.1.1
   switcher_reload
   make
```

should do the trick.

Also note that PVM will put all compiled binaries in ~/pvm3/bin/LINUXI386, so if you're not sure where those binaries went, that's the place to look!

Finally, each program subdirectory includes a README file which contains a brief series of instructions for how to compile and execute the included test application using the selected toolchain. These can provide basic syntax examples for how to run the demonstration applications in a parallel environment.

```
Have fun!
```

C.5 scripts/NEWORDER

```
*NOTE: as of 3/16/2007, this file is pretty out of date.  However, it
       describes a lot of basics about the order of how things run, so
       although it's not perfect, it's a good document.  Some of the
       file names & paths have changed, but the scripts can be reviewed
       to find those changes.  This still provides lots of good info.

-----

This is the "newer" order (in a nutshell), that allows a split,
so we can detect nodes AND build the node tarball,
simultaneously.

1) MASTER-SCRIPT-1 runs, copying files and setting stuff up.
2) MASTER-SCRIPT-2 runs.  It fires off some "config" scripts (the
   ones that don't need get_macs.sh) after all other info about the
   cluster has been collected.  At the end, it starts building the
   cluster node image, as a background job, logging to a file in /tmp.
3) MASTER-SCRIPT-3 runs.  It starts by collecting MAC addresses, and
   reminds the user to "killall tcpdump" to stop it.  (Perhaps a little
   script trickery to actually END tcpdump upon a keypress?)
   After the tcpdump process ends, it continues running the remaining
   "config" and "do" scripts (that do depend on MAC_INFO).  It then joins
   the background process which is building the node image, tailing the
   log file until the process is complete (how do we know?)  When done,
   it adds the files it created to the node tarball (how -- tar -r, or
   by adding them to post_install?), and continues with the YACI part of
   the deployment.

   [NOTE: tar -r still must read the whole tar file.  As a result, it
          probably will be faster to just include something in the
          post_install script to copy the files over, especially since
          do_modify_post_install can't run anyway until after get_macs.sh
          has completed.  Where to put the files?  /tftpboot/local/ ?
```

It has other uses. What about /usr/local/ssh-exchange/ ? Maybe rename it to something more generic, for example /usr/local/bsu-cluster-installer--file-exchange/, and put the files in a .tar.gz. Then a quick tweak to post_install, and the files can be extracted.]

Modifying the above to work with the web scenario is going to be a bit difficult. One nice thing is that it will allow us to ask the user if they want to rebuild the entire node tarball. If they're just changing the list of detected nodes, and nothing else, then they can still use the same node tarball, which will save them a lot of time. And it still reconfigures the necessary network files on the master & compute nodes, too!

Can we track if any parameters which would require a rebuild of the tarball have changed, and force a rebuild based on that? Probably pointless, as the user could have changed the rpmlist and needs the node to be rebuilt. Better to just ask the user, and default to "yes".

We could have the UI present a single page to the user, with a checkbox for a) change base name and re-detect MACs and (b) rebuilding the node tarball. Maybe a brief explanation of each option. When the user clicks "next", the selected actions start (either rebuilding the MAC list, rebuilding the tarball, neither, or both). Default is both (?).

If both will happen, then a new popup opens, with the "MAC detection" dialog. In the background, the main window opens with the shell script of the node tarball rebuild. If just MAC detection, then the popup opens, and the background window waits until the MAC window is closed. Otherwise, we wait until the background script is done, then the next page button is allowed. If no action is selected, then the entire page is skipped.

If a user clicks "back", then all instances of tcpdump and the other script are killed by the webserver. I think this is already being done for tcpdump instances. Or maybe it happens as the page loads. Either way will work I guess.

C.6 scripts/NEWORDER.notes

```
*NOTE: as of 3/16/2007, this file is pretty out of date.  It still describes
       some dependencies for the "parallel" installation process, but there
       are a few more dependencies (not listed below) which exist, and some
       of the filenames/paths have changed.  See the quickie-* scripts to
       get an up-to-date listing of all the dependencies.

       The "quickie-*.1.sh" scripts run *before* we know how many nodes there
       are (so the changes they make have no dependencies).  The
       "quickie-*.2.sh" scripts run *after* the MAC detection has happened,
       so we know exactly how many nodes there are (e.g. the things touched
       by these scripts _do_ have dependencies).

-----

Need FILENAME__MACINFO for:
- determining what the master node's name should be (# of '0's to append)
  (basename.sh)

- Building the dhcpd.conf file on the server.
  (config-dhcpd.conf.server.sh)

- Generating /etc/hosts for all systems.
  (config-hosts.all.sh)

- Generating the PBS "nodes" file on the server.
  (config-nodes.server.sh)

- Building the pvmd script (references basename.sh) on the server.
  (config-pvmd.server.sh)

- Building .rhosts file on all systems.
  (config-.rhosts_and_mom_priv_config.all.sh)

- "do_modify_post_install" references basename.sh.
  (do_modify_post_install.sh)

- MASTER-SCRIPT-2 (and MASTER-SCRIPT-2-1.sh) use it to count how many nodes to
  wait for before declaring the install a success.
```

(MASTER-SCRIPT-2.sh, MASTER-SCRIPT-2-1.sh)

Appendix D

DEVELOPMENT README FILES

These are files which are part of the development tree, contained in the project's Subversion source code repository. They provide detail to developers who are looking to update or improve upon the existing BSACI code base.

D.1 README.placeholder_files

```
Some of the larger files in this repository have been removed, in the interest
of saving disk/revision space.  They have been replaced with placeholder files
that describe the purpose of the original file, how to get the original file,
etc.  'find -iname *.placeholder' should locate these files for you.
  \end{singlespace}
```

D.2 development-tools/README

```
This directory contains source packages and build scripts necessary to compile
various packages required by the Boise State Cluster Installer.  Once the
packages are built, a "make install" on a clean machine will install them
to the appropriate directories, where they can then be collected into a
tarball.

Assiduous users will either "make install" to a fake root directory, and use
that directory to build the tarball, or examine the directories contained in
```

current tarballs, and use that information to select which directories to add to the new tarball after performing a "make install".

The "built-packages" directory is where tarball packages which have been built, installed, then collected into tarballs are stored. This directory is filled with placeholder files, since the original files may be obtained by unpacking the 'tftpboot-distro-overlay.tar' file from the BSACI installation media, then copying all files from the 'dist/others/master' directory to the 'master' subdirectory. All files from the 'dist/others' directory may be copied to the 'all_nodes' subdirectory.

The "build-scripts" and "sources" directories contain the compilation scripts and source tarballs, respectively. Untar a source tarball, copy the associated build script into the directory where the source was unpacked, and run it. Don't forget to "make install" after the compile finishes. Also note that there are some dependencies on the order in which these packages are compiled. E.g. the "hdf5" code depends upon an MPI library (MPICH2 or OpenMPI) being already compiled and installed, or it will not build. Etc.

The "other-scripts" directory includes several scripts which didn't really fit anywhere else. They are generally replacement code for scripts which are already on the system, forcing the various tools (MIPCH2, LAM, OpenMPI) to try to use the PBS batch scheduling system.

Appendix E

PROJECT MANAGEMENT

E.1 Source control

Development code (scripts, build tools, embedded source code, etc.) was moved into a Subversion version control database on March 17, 2007. This database tracks changes and provides history for all revisions to the cluster installer code base since that date.

E.2 Where to get the code

The development code is stored in a Subversion code repository. The primary code repository is located at `http://www.thedeacon.org/repos/bsaci`. If this repository is down, a second repository is located at Boise State University, `https://onyx.boisestate.edu/r` Please contact `amit@cs.boisestate.edu` for access.

To check out current development code from either of these Subversion repositories, use the `svn co <url>` command, where `<url>` is the address of the repository, as given above. Note that you may receive an "Access Denied" error message—if this happens, you lack proper authentication to retrieve data from the Subversion repository. Try a different repository, or contact the repository administrator to request access.

The production code (full distribution image, including RPM packages – around 465 MB) may be found on various sites, including `http://thedeacon.org/` and `http://onyx.boisestate.edu/`. Again, for exact locations on the Boise State `onyx` server, please contact `amit@cs.boisestate.edu`.