NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters

Philip M. Papadopoulos, Mason J. Katz, Greg Bruno
The San Diego Supercomputer Center
University of California San Diego
La Jolla, CA 92093-0505
[phil,mjk,bruno]@sdsc.edu
http://rocks.npaci.edu

Abstract

High-performance computing clusters (commodity hardware with low-latency, high-bandwidth interconnects) based on Linux, are rapidly becoming the dominant computing platform for a wide range of scientific disciplines. Yet, straightforward software installation, maintenance, and health monitoring for large-scale clusters has been a consistent and nagging problem for non-cluster experts. The NPACI Rocks toolkit takes a fresh perspective on management and installation of clusters to dramatically simplify software version tracking, and cluster integration.

*The toolkit incorporates the latest Red Hat distribution (including security patches) with additional cluster-specific software. Using the identical software tools used to create the base distribution, users can customize and localize Rocks for their site. Strong adherence to widely-used (*de facto*) tools allows Rocks to move with the rapid pace of Linux development. Version 2.1 of the toolkit is available for download and installation. To date, 10 clusters spread among 5 institutions have been built using this toolkit.*

1. Introduction

Strictly from a hardware component and raw processing power perspective, commodity clusters are phenomenal price/performance compute engines. However, if a scalable "cluster" management strategy is not adopted, the favorable economics of clusters are changed due to the additional on-going personnel costs involved to "care and feed" for the machine. The complexity of cluster management (e.g., determining if all nodes have a consistent set of software) often overruns part-time cluster administrators (who are usually domain application scientists) to either of two extremes: the cluster is not stable due to configuration problems, or software becomes stale (security holes, known software bugs remain unpatched).

While earlier clustering toolkits expend a great deal of effort (i.e., software) to compare configurations of nodes, Rocks makes complete OS installation on a node *the basic* management tool. With attention to complete automation of this process, it becomes faster to reinstall all nodes to a known configuration than it is to determine if nodes were out of synchronization in the first place. Unlike a user's desktop, the OS on a cluster node is considered to be soft state that can be changed and/or updated rapidly. This is clearly diametrically opposed to the philosophy of configuration management tools like Cfengine [7] that perform exhaustive examination and parity checking of an installed OS. At first glance, it seems wrong to reinstall the OS when a configuration parameter needs to be changed. Indeed, for a single node this might seem too heavyweight. However, this approach scales exceptionally well (see Table 1) making it a preferred mode for even a modest-sized cluster. Because the OS can be installed from scratch in a short period of time, different (and perhaps incompatible) application-specific configurations can easily be installed on nodes.

One of the key ingredients of Rocks is a robust mechanism to produce customized (with security patches pre-applied) distributions that define the complete set of software for a particular node. Within a distribution, different sets of software can be installed on nodes (for example, parallel storage servers may need additional components) by defining a machine specific Red Hat Kickstart file. A Kickstart file is a text-based description of all the software packages and software configuration to be deployed on a node. By leveraging this installation technology, we can abstract out many of the hardware differences and allow the Kickstart process to autodetect the correct hardware modules to load (e.g., disk subsystem type: SCSI, IDE, integrated RAID adapter; Ethernet interfaces; and high-speed network interfaces). Further, we benefit from the robust and rich support that commercial Linux distributions must have to be viable in today's rapidly advancing marketplace.

Wherever possible, Rocks uses automatic methods to determine configuration differences. Yet, because clusters are unified machines, there are a few services that require "global" knowledge of the machine – e.g., a listing of all compute nodes for the hosts database and queuing system. Rocks uses a MySQL database to define these global configurations and then generates database reports to create service-specific configuration files (e.g., DHCP configuration file, /etc/hosts, and PBS nodes file).

Since May of 2000, we've been addressing the difficulties of deploying manageable clusters. We've been driven by one goal: **make clusters easy**. By *easy* we mean easy to deploy, manage, upgrade and scale. We're driven by this goal to help deliver the computational power of clusters to a wide range of users. It's clear that making stable and manageable parallel computing platforms available to a wide range of scientists will aid immensely in improving the parallel tools that need continual development.

In Section 2, we provide an overview of contemporary clusters projects. In Section 3, we examine common pitfalls in cluster management. In Section 4, we examine the hardware and software architecture in greater detail. In Section 5, we detail the management strategy which guides everything we develop. In Section 6, you'll find deeper discussions of the key NPACI Rocks tools, and in Section 7, we describe future Rocks development.

2. Related Work

In this section we reference various clustering efforts and compare them to the current state of Rocks.

- **Real World Computing Partnership** - RWCP is research group started in 1992, and based in Tokyo. RWCP has addressed a wide range of issues in clustering from low-level, high-performance communication [14] to manageability. Their SCore software provides semi-automated node integration using Red Hat's interactive installation tool [3], and a job launcher similar to UCB's REXEC (discussed in Section 4.1).

- **Scyld Beowulf** - Scyld Computing Corporation's product, *Scyld Beowulf*, is a clustering operating system which presents a single system image (SSI) to users through the *Bproc* mechanism by modifying the following: the Linux kernel, the GNU C library, and some user-level utilities. Rocks is not an SSI system. On Scyld clusters, configuration is pushed to compute nodes by a Scyld-developed program run on the frontend node. Scyld provides a good installation program, but has limited support for heterogeneous nodes. Because of the deep changes made to the kernel by Scyld, many of the bug and security fixes must be integrated and tested by them. These fundamental changes require Scyld to take on many (but not all) of the duties of distribution provider.

- **Scalable Cluster Environment** - The SCE project is a clustering effort being developed at Kasetsart University in Thailand [16]. SCE is a software suite that includes tools to install compute node software, manage and monitor compute nodes, and a batch scheduler to address the difficulties in deploying and maintaining clusters. The user is responsible for installing the frontend with Red Hat Linux on their own, then SCE functionality is added to the frontend via a slick-looking GUI. Installing and maintaining compute nodes is managed with a single-system image approach by network booting (a.k.a., diskless client). System information is gathered and visualized with impressive web and VRML tools. In contrast, Rocks provides an entire, self-contained, cluster-aware installation built upon Red Hat's distribution. This leads to consistent installations for both frontend and compute nodes, as well as providing well-known methods for users to add and customize cluster functionality. Also, Rocks doesn't employ diskless clients, avoiding scalability issues and functionality issues (not all network adapters can network boot).

- **Open Cluster Group** - The Open Cluster Group has produced a developer's release of their OSCAR toolkit [10]. OSCAR is a collection of common clustering software tools in the form of tar files that are installed on top of a Linux distribution on the frontend machine. When integrating compute nodes, IBM's Linux Utility for cluster Install (LUI) operates in a similar manner to Red Hat's Kickstart. OSCAR requires a deep understanding of cluster architectures and systems, relies upon a 3rd-party installation program, and has fewer supported cluster-specific software packages than Rocks.

- **VA Linux** - VA Linux sells software cluster bundles. The installed software is configured at their facility when the system is ordered using "Build-to-Order Software" (BTOS) [6]. However, the BTOS system does not provide the rich scripting ability of Kickstart, and requires the buyer to hand-configure each node. Further, there is no software upgrade strategy. VA Linux does provide a rich management system (VACM) [11], however, it relies on Intel's proprietary Emergency Management Port, and requires a dedicated management network of serial cables to be installed. In comparison, Rocks runs on a variety of hardware and needs no secondary management network.

- **Extreme Linux** - Extreme Linux is a community movement started in early 1998 at the "Extreme Linux Workshop". At that workshop, Red Hat and NASA CESDIS jointly released a CD containing a distribution to help build Beowulf-class clusters. This was really a collection of now-standard cluster tools like MPI and PVM. Since then, Extreme Linux has held five more workshops tasked with defining the current and future role of Linux high-performance clustering. A follow-up CD has not been released and the packaging efforts appear to have halted.

3. Pitfalls

We embarked on the Rocks project after spending a year running a single, Windows NT, 64-node, hand-configured cluster. This cluster is an important experimental platform that is used by the Concurrent Systems Architecture Group (CSAG) at UCSD to support research in parallel and scalable systems. On the whole, the cluster is operational and serves its research function. However, it stays running because of frequent, on-site, administrator intervention. After this experience, it became clear that an installation management strategy is *essential* for scaling and for technology transfer. This section examines some of the common pitfalls of various cluster management approaches.

3.1. Disk Cloning

The CSAG cluster above is basically managed with a disk cloning tool, where a model node is hand-configured with desired software and then a bit-image of the system partition is made. Commercial software (ImageCast in this case) is then used to clone this image on homogeneous hardware. Disk cloning was also espoused as the preferred method of system replication in [13]. While clusters usually start out as homogeneous, they quickly evolve into heterogeneous systems due to the rapid pace of technology refresh as they are scaled or as failed components are replaced. As an example, over the past 10 months, the Rocks-based "Meteor" cluster at SDSC, has evolved from a homogeneous system to one that has five different types of nodes from two vendors with two different types of disk-storage adapters. Further, a handful of these machines are dual-homed Ethernet frontend nodes, and most compute nodes have Myrinet adapters, but not all.

Node heterogeneity is really a common state for most clusters and being able to transparently manage these small changes makes the complete system more stable. Additionally, while the software state of a machine can be described as the sequential stream of bits on a disk, a more powerful and flexible concept is to use a description of the software components that comprise a particular node configuration. In Rocks, software package names and site-specific configuration options fully describe a node. This text-based description (Red Hat Kickstart file) of the most complex machine, the frontend node, is less than 20 KB (compute node description files are 12 KB). At the start of this project, we assumed that maintaining a separate description file for each compute node type would be essential. However, this assumption turned out to be false. Using the distribution's tools for hardware probing and a small amount of additional probing (e.g., checking for the existence of a Myrinet card on the PCI bus), a single Kickstart file can be used for all five node types in the Meteor cluster.

3.2. Installing Each System "By Hand"

Installing and maintaining nodes by hand is another common pitfall of neophyte cluster administrators. At first glance it appears manageable, especially for small clusters, but as clusters scale and as time passes, small differences in each node's configuration negatively affects system stability. Even savvy computer professionals will occasionally enter incorrect command line sequences, implying that the following questions need to be answered:

- "What version of software X do I have on node Y?"
- "Software service X on node Y appears to be down. Did I configure it correctly? When my script attempted to update 32 nodes ran, was node X offline?"
- "My experiment on node X just went horribly wrong. How do I restore the last known good state?"

The Rocks methodology eliminates the need to ask these questions (which rotate generally around consistency of configuration).

3.3. Proprietary Installation Programs and Unneeded Software Customization

Proprietary and/or specialized cluster installation programs seem like a good idea when presented with the current technology of commercial distributions. However, going down the path of building a customized installer, means that many of the hardware detection algorithms that are present in commercial distributions must be replicated. The pace of hardware updates makes this task too time-consuming for research groups and is really unneeded "wheel reinvention". Proprietary installers, such as the one used and marketed by VA Linux, often demand homogeneous hardware or, even worse, a specific brand of homogeneous hardware. This reduces choice for a cluster user and

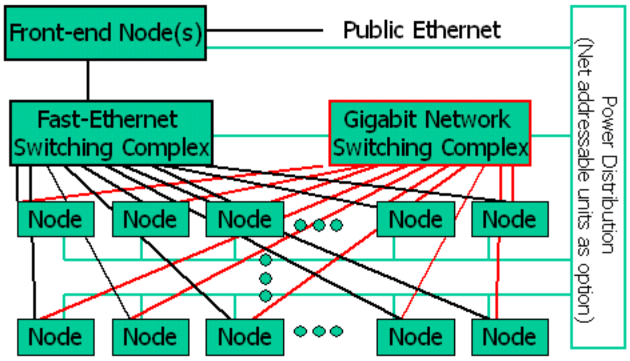

Figure 1. Rocks hardware architecture. Based on a minimal traditional cluster architecture.

can inhibit the ability to grow or update a cluster. While Rocks does not have all the bells and whistles of some of these installers, it is hardware neutral. Also, specialized cluster installation programs often don't incorporate the latest software, a pitfall which is described, and addressed, later in this paper.

Unneeded software customization is another pitfall. New cluster administrators seem unable to resist tinkering with kernel configurations and other components. Sometimes customized kernels are warranted (e.g., patching to allow access to hardware performance counters), but often are unnecessary. While nothing in Rocks precludes custom kernels, the default configuration uses the stock Red Hat configured kernel.

4. Rocks Cluster Hardware and Software Architecture

To provide context for the tools and techniques described in the following sections, we'll introduce the hardware and software architecture on which Rocks runs.

Figure 1 shows a traditional architecture commonly used for high-performance computing clusters as pioneered by the Network of Workstations project [15] and popularized by the Beowulf project [12]. This system is composed of standard high-volume servers, an Ethernet network, power and an optional off-the-shelf high-performance cluster interconnect (e.g., Myrinet or Gigabit Ethernet). We've defined the Rocks cluster architecture to contain a minimal set of high-volume components in an effort to build reliable systems by reducing the component count and by using components with large mean-time-to-failure specifications.

In support of our goal to "make clusters easy", we've focused on simplicity. There is no dedicated management network. Yet another network increases the physical deployment (e.g., more cables, more switches) and the management burden, as one has to manage the management network. [1] We've made the choice to remotely manage compute nodes over the integrated Ethernet device found on many server motherboards. This network is essentially free, is configured early in the boot cycle, and can be brought up with a very small system image. As long as compute nodes can communicate through their Ethernet, this strategy works well. If a compute node doesn't respond over the network, it can be remotely power cycled by executing a hard power cycle command for its outlet on a network-enabled power distribution unit. [2] If the compute node is still unresponsive, physical intervention is required. For this case, we have a *crash cart* – a monitor and a keyboard.

This strategy has been effective, even though a crash cart appears to be non-scalable. However, with modern components, total system failure rates are small. When balanced against having to debug or manage a management network for faults, reducing network complexity appears to be "a win". The downside of using only Ethernet, is that an administrators is "in the dark" from the moment the node is powered on (or reset) to the time Linux brings up the Ethernet network. Our experience has been if Linux can't bring up the Ethernet network, either a hardware error has occurred with a high probability that physical intervention is necessary, or a central (common-mode) service (often NFS) has failed. Hardware repairs require nodes to be removed from the rack. For a common-mode failure, fixing the service and then power cycling nodes (remotely) solves the dilemma. To minimize the time an administrator is in the dark, we've developed a service that allows a user to remotely monitor the status of a Red Hat Kickstart installation by using telnet (see Section 6.2). With this straightforward technology, details of a node power-on-self-test is the only status that cannot be viewed from a remote location. However, it appears that vendors will soon provide the capability to view BIOS over integrated Ethernet devices. This technological change is being driven by the needs of Internet Service Providers to cut costs in deploying large web farms.

4.1. Frontend Node

The frontend is installed with widely-used, standard software to support cluster application development and parallel application execution. We've experimented with *gcc, g77* and *The Portland Group's High Performance Fortran* compilers and we are currently investigating Intel's Linux C/C++ and Fortran compilers.

Some of the libraries found on the frontend are Intel's

[1] Recursion sucks. [1]

[2] A hard power cycle on a Rocks compute node forces the node to reinstall itself.

Math Kernel Library which contains math functions (FFTs, matrix multiply, etc.) that are tuned for Intel processors as well as various parallel machine message passing libraries MPICH (Myrinet and Ethernet device support) and PVM (Ethernet device support).

To support job launching in production environments, we've packaged the Portable Batch System (PBS) and the Maui scheduler. PBS is used for its workload management system (starting and monitoring jobs) and Maui is used for its rich scheduling functionality. When the frontend is installed, PBS and Maui are automatically started and a default queue is defined.

For development environments, Rocks includes *mpirun* from the MPICH distribution and REXEC (remote execution) system from UC Berkeley [5]. REXEC provides transparent, secure remote execution of parallel and sequential jobs. It has a sophisticated signal handling system which provides remote forwarding of signals. REXEC also redirects stdin, stdout and stderr from each parallel process and it propagates a local environment including environment variables, user ID, group ID and current working directory.

4.2. Compute Nodes

Compute nodes are the workhorses – they execute all the jobs. Peak performance is dictated by the number and type of compute nodes and normally there are many more compute nodes than frontend nodes. Since there can be wide range for the number of compute nodes, our software has been designed to accommodate this variable.

5. Management Strategy

Our management strategy is based on the following philosophy:

> It must be trivial to deploy any version of software on any node of the cluster, regardless of the cluster size.

To implement this vision, we've defined these rules:

- All software deployed on Rocks clusters are in RPMs.
- Require 100% automatic configuration of compute nodes.
- It's essential to use scalable services (HTTP, NIS, etc.).

Red Hat has developed two key technologies which directly support this philosophy: Red Hat Package Manager (RPM) and the *Kickstart* installation tool. Analogous to Sun Microsystems' packages, an RPM package contains all the files (e.g., binaries, header files, init scripts, man pages) for deploying a particular software module. More importantly, RPMs can be installed using the command line which promotes adding or updating packages via scripts.

Red Hat has written a sophisticated, customizable script which automates package installation from Red Hat distributions called *Kickstart*. Nodes installed in this manner are driven by a user created configuration file that essentially contains the answers to all the questions posed by a standard interactive installation. Kickstart files also can contain scripts that are run during the installation. Rocks leverages this scripting feature to achieve 100% automatic configuration of compute nodes.

Rocks clusters are fundamentally about managing two systems: a frontend node and a compute node. The frontend node requires the skills of a savvy UNIX user, as this is a machine which runs many of the services found on any robust server. In contrast, the compute node is a minimal server, and simply serves as a container to run parallel jobs. Simplifying the role of a compute node, treating their base OS as stateless, and requiring 100% automatic configuration makes scaling-out tenable. Each compute node added to the system only increments the total management effort by a small amount.

Another requirement for scaling out is only using scalable services and utilizing dynamic services for frequently changing state which must be communicated to compute nodes (user information, network configuration, etc.). For installation, compute nodes use Kickstart's HTTP method to pull RPMs across the network. For configuring Ethernet devices on compute nodes, the Dynamic Host Configuration Protocol (DHCP) is essential. User account configuration (e.g., passwords and home directory locations) are synchronized from the frontend node to compute nodes with the Network Information Service (NIS).

We have employed one unscalable service, the Network File System (NFS) [8]. The frontend node exports all user home directories to compute nodes via NFS. We are searching for a scalable alternative.

What does this all mean? The strategy described above simplifies cluster management, promotes experimentation and provides a method to keep production machines on current software. As mentioned in Section 3, we used to spend a large amount of time checking if compute nodes were consistent. Now, rather than wondering, we simply reinstall by sending a message over the network. After a compute node completes its reinstallation, currently 5-10 minutes, the node is consistent.

Our strategy promotes experimentation. Developers can change configuration and try services in a wanton manner because any number of compute nodes can be restored to a known good state in 5-10 minutes.

Finally, software on production machines can be systematically and continually upgraded. As described in the next section, we've built a tool that easily updates a Rocks distribution (a collection of RPMs from Red Hat, the community and NPACI). This tool can be used to apply the latest security advisories and bug fixes. After the updates are validated on a small test cluster, the production system can be upgraded by submitting a "reinstall cluster" job to Maui, as not to disturb any running applications. Once the reinstallation is complete, the next job will have a known, consistent software base.

6. Tools

6.1. Managing a Cluster-Enhanced Linux Distribution with Rocks-Dist

A major problem in the day-to-day administration of clusters is managing the software that runs on the nodes. This problem is twofold. First, how is the software initially deployed and what is the administration cost of the initial deployment? Second, how are upgrades to the system software achieved and how are software upgrades chosen?

The initial deployment of a Rocks cluster is performed using Red Hat's Kickstart installation program. However, since a Red Hat distribution is only a collection of RPMs, anyone can create a new distribution that can be installed with Kickstart. This is the foundation of the Rocks distribution: We start with a stock Red Hat release, apply Red Hat's updates and add a small number of RPMs. The new distribution is supported by Kickstart and remains true to the original structure of a Red Hat distribution, only the list of software packages has been expanded.

6.1.1. Keeping Up with Software

One of the widely-claimed benefits of open source software is the rapid pace of development. These benefits range from the quick closure of security holes in software, to rapid performance and functionality enhancements. Although these are tremendous benefits of Linux (and open source in general), this rapid turn around of software is also a great burden on system administrators. For example, in less than a year, Red Hat 6.2 for Intel had 124 updated packages. There were also 74 security vulnerabilities reported to www.securityfocus.com, for which several of the updated packages were targeted. On average, this amounts to one update every three days.

A goal of Rocks is to provide an agile cluster toolkit that rests on top of the latest Red Hat Linux software set. In this vein, the only manageable scheme for addressing software updates is to automatically track them. While Red Hat does not always "get it right" they must remain responsive and correct any errors to maintain their leading market share. We simply do not have the manpower, time, or interest to

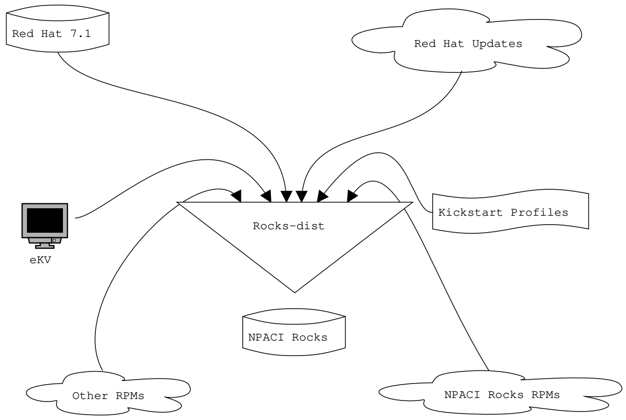

Figure 2. Building a Rocks distribution with rocks-dist.

inspect every software update and bless it. If Red Hat ships it, so do we.

To integrate our software with Red Hat's stock and updated packages, we created a program called **rocks-dist**. Rocks-dist gathers software components from the following sources and constructs a single new distribution:

- **Red Hat software** - The stock distribution and updated RPMs replicated onto a local mirror. Rocks-dist resolves version numbers of RPMs and only includes the most recent software. This behavior alone allows us to produce an always up-to-date Red Hat distribution that frees the user from having to update software after an initial installation.
- **Third party software** - Any desired software not included in Red Hat. Some of our database packages fall into this category.
- **Local software** - All RPMs built on site. For Rocks, this means all Rocks packages and Kickstart profiles for compute nodes and frontend nodes. This also includes our eKV enhancement to Kickstart to provide status and interactive control over the network during the Kickstart process.

6.1.2. Extensibility

Figure 2 illustrates the process of gathering software to produce a distribution. The resulting Rocks distribution looks just like a Red Hat distribution, only with more software. A consequence of this is repeat-ability - a Rocks distribution can be run through the identical process to produce an enhanced Rocks distribution. This allows a user,

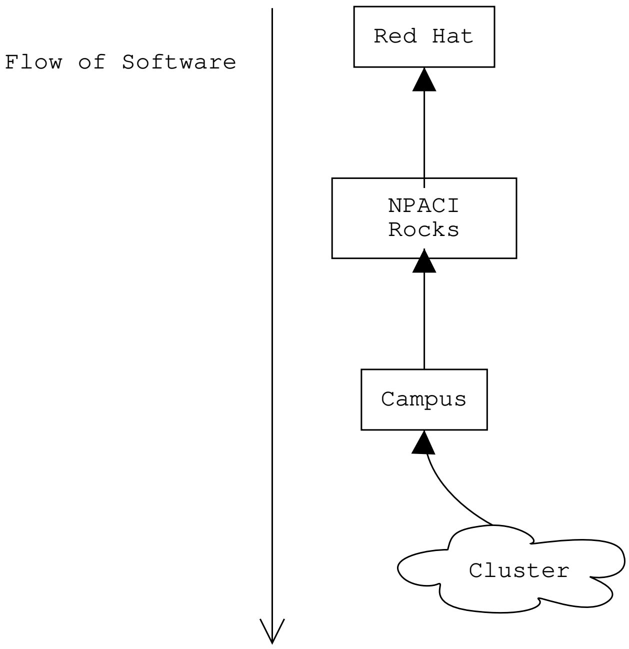

Figure 3. Object-oriented model of rocks-dist.

such as a university campus, to add local software packages to Rocks, and have all departments build clusters based off the campus' distribution. This object-oriented approach is illustrated in Figure 3.

We envision a hierarchy of Rocks distribution hosts, each adding software packages for child distributions. This method of distributing software allows us to focus on our Rocks components while remaining highly extensible for end users.

6.1.3. Kickstarting and Heterogeneity

Software homogeneity is not always a desirable goal. For instance, during development of Rocks, we had the need to isolate developers from one another and allow different distributions to be installed on compute nodes of a shared cluster. This need is not isolated to software development, even production users may want custom software deployed on nodes for a specific job run, without affecting other users on the cluster.

When building a new distribution, rocks-dist replicates the software from its parent distribution using wget over HTTP (see Figure 3) and creates a new tree comprised mostly of symbolic links to the mirrored software. Inside this tree is a build directory that contains Kickstart files. Users can customize this new distribution by editing the

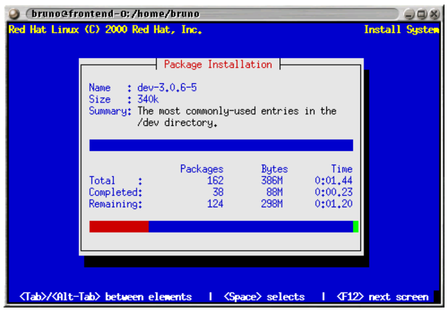

Figure 4. Shoot-node and eKV. Red Hat's Kickstart screen is redirected over Ethernet.

Kickstart files and manually adding new RPMs that rocks-dist did not integrate. By creating multiple distributions and editing the Kickstart files, the user can create unique configurations for subsets of cluster nodes. Further, because each distribution is composed mainly of symbolic links, each distribution is lightweight (on the order of 25MB) and can be built in under a minute.

The strategy is similar to diskless clients which boot their system image off NFS. However, by pushing the software to the nodes, we incur a single network bandwidth penalty which does not recur every time the node boots. Further, we can create variants of the software images in minimal space and time.

6.2. Reinstallation

Reinstallation is the primary mechanism for forcing the base OS on the root partition of compute nodes to a known state. As a side note, all non-root partitions are preserved over reinstalls, and therefore, can be used as persistent storage. After building a distribution with rocks-dist, the distribution is applied to compute nodes via reinstallation.

A compute node reinstalls itself when an administrator invokes **shoot-node**, or after a hard power cycle (e.g., power failure). Shoot-node is a command-line tool that, over Ethernet, instructs a compute node to reboot itself into installation mode. It monitors the node's progress and pops open an xterm window which displays the status of the Red Hat Kickstart installation. This status is redirected over the Ethernet by **eKV** (Ethernet Keyboard and Video). This is accomplished by slightly modifying Red Hat's Kickstart installation program, **anaconda**, to capture standard output and present it on a telnet-compatible port. Should something go wrong, we've also inserted code that allows users to interact with the installation through the same xterm window that reports the installation status (Figure 4).

Using HTTP to distribute RPMs, scales well. Table 1 shows the total time to reinstall a number of nodes concur-

Nodes	Total Reinstall Time (minutes)
1	10.3
2	9.8
4	10.1
8	10.4
16	11.1
32	13.7

Table 1. Reinstallation performance. The HTTP server is a dual 733 MHz PIII with 100 Mbit Ethernet. Compute nodes are 733 MHz - 1 GHz with Myrinet. Times include the time taken to rebuild the Myrinet driver. Each node transfers approximately 150 MB of data from the server.

rently. In this experiment the HTTP server is dual 733 MHz PIII with fast Ethernet, and the compute nodes are 733 MHz - 1 GHz with Myrinet. Times include the time taken to rebuild the Myrinet driver. Each node transfers approximately 150 MB of data from the server.

As mentioned earlier, compute node reinstallation time is between 5 and 10 minutes. The upper bound is for compute nodes with a Myrinet card, which rebuild the driver from source on its first boot after an installation. Driver rebuilds mitigate the problem of having to keep N Myrinet driver binary packages for N versions of the Linux kernel. Because the Linux kernel has module versioning enabled (the default for Red Hat compiled kernels), it will only load modules that were compiled for that particular kernel version. The Linux kernel moves quickly – for the year 2000, there were 5 updates to the ”stable tree”. Since we maintain the package for the Myrinet driver, we quickly grew tired of the cycle of: installing a node with the latest kernel and compilers, preparing the kernel tree for compilation, compiling the Myrinet driver, packaging the driver, then transporting the binary package back to our distribution server. The easiest way to manage kernel version changes is to have each compute node compile the Myrinet driver from a source RPM. The Myrinet driver module can be compiled, installed, and started without incurring a reboot. The seemingly heavyweight solution adds only a 20-30% time penalty on reinstallation.

6.3. Cluster Monitoring

We have only recently begun to address long-term monitoring of compute nodes. But, we believe we have provided a foundation to allow for gathering and processing of cluster state. This is achieved using standard UNIX tools and Ganglia from UC Berkeley.

6.3.1. System Logger

Compute nodes forward all syslog messages to a frontend machine. To address scalability, the name of the loghost is sent to compute nodes as a DHCP option, allowing nodes to report to different hosts. As the Linux kernel boots, all log messages are cached and sent to the syslog daemon once started, enabling an operator to see all the boot-time messages on the loghost.

6.3.2. SNMP

Compute nodes run SNMP with a Linux MIB to allow interactive probing of machines using standard tools. We provide a script called **snmp-status** to allow the user to inquire about the process status on any given node.

6.3.3. Ganglia

Ganglia is a lightweight, distributed, multicast-based monitoring system [1]. Each node runs a daemon called a dendrite. Dendrites collect software and hardware configuration, state changes, and a notion of ”health” on nodes. This information is multicast to collecting agents called axons. A user can then run the command line ganglia application to discover an axon agent and query the state of the cluster.

Once an hour, all the dendrites multicast static configuration information such as the number and speed of the CPUs, the kernel version, and the amount the RAM installed. Once every five seconds, the dendrites report what they view as significant changes in the percentage of CPU time, load averages, memory load, and number of running processes.

In comparison to snmp-status, Ganglia is a lightweight push of information, whereas snmp-status is a heavyweight pull. On balance, more detailed information can be obtained from SNMP.

6.4. Configuration Database

One of the most serious pitfalls of UNIX is the lack of a common format for configuration files. The Registry on Windows machines is an attempt to enforce a single extensible format for configuration. Although the Registry has its own problems, it is a step in the right direction. Rocks clusters use a MySQL database for site configuration. The two key tables we provide are, 1) a DHCP options table and, 2) a node table. From these tables we generate the /etc/hosts, /etc/dhcpd.conf, and PBS configuration files.

The “nodes table” houses the bindings between host names and Ethernet addresses. When the frontend machine is installed from the Rocks CD distribution, the database is created, and an entry for this machine is added to the database. Bindings for the compute nodes are added to the database by running the utility **insert-ethers** on the frontend machine, and sequentially booting compute nodes with the Rocks CD. Insert-ethers monitors syslog messages for DHCP requests from new hosts and when found, generates a hostname, binds the hostname to its Ethernet MAC address, and adds this information to the database. Insert-ethers then rebuilds service-specific configuration files by running reports from the database, and restarting the respective services.

For most nodes, such as compute nodes, only the tuple of (name, MAC) is required as IP addresses are generated dynamically when we build the /etc/hosts and /etc/dhcpd.conf files. For nodes that must publish their IP address (e.g., frontend nodes) we store the triple of (name, ip-address, MAC). Table 2 illustrates this flexibility.

7. Current Status and Future Work

As of July 2001, Rocks version 2.1 is:

- Red Hat 7.1 base plus Red Hat's 79 security advisories and bug fixes.
- Assorted community software (MPICH, PVM, Intel math kernel library, etc.).
- NPACI software (the software described in this paper).

We've used Rocks to install 5 clusters on the UCSD campus and at least 4 groups have installed their own Rocks clusters (Advanced Computing Center for Engineering & Science at UT Austin, Pacific Northwest National Labs, the Chemistry Department at Northwestern University, and a group at the Hong Kong Baptist University). We say "at least" four because the total number is unknown as our software doesn't require registration, therefore, we don't have a hard count of Rocks users – we only know our user base through direct communication.

Rocks is installed with a floppy and a CD (ISO image downloadable from **`http://rocks.npaci.edu`**). The frontend Kickstart file is built from a simple web form (**`https://rocks.npaci.edu/kickstart/`**), and is saved onto the floppy. After the frontend is installed, the same CD is used to bring up the individual compute nodes (the floppy is not needed for compute node installations). This scheme is similar to Scyld Computing Corporation's cluster installation found on their *Scyld Beowulf* CD [4].

We look forward to integrating future clusters. Two announced clusters which will be running Rocks are: 1) a prototype machine for the GriPhyN project, and 2) a production cluster for the Scripps Institute of Oceanography (SIO). The GriPhyN project (Grid Physics Network) is tasked with building a Petabyte-scale computational environment. Paul Avery, the principal investigator, has chosen to use Rocks to build a prototype Tier 2 server. When in production, the Tier 2 sites will provide roughly 1/3 of the cycles needed by high-energy physicists to analyze data coming from experiments at CERN's Large Hadron Collider. Detlef Stammer of SIO, will be using Rocks to build a 128-node cluster used to study ocean general circulation through ocean modeling and ocean state estimation.

We also look forward to attacking the issues we discover as we grow our cluster (as of July 2001, it stands at 82 compute nodes), and we're eager to integrate new software and hardware technologies into Rocks. We're actively working on a project which extends the NPACI Rocks toolkit to easily put clusters on the Globus Grid [9]. On the hardware side, as IA-64 nodes and InfiniBand interconnect components [2] arrive in the commodity space, we'll include their respective packages in future releases.

8. Conclusion

Armed with a management strategy which dictates the three mechanisms of 1) all software deployed are in RPMs, 2) 100% automatic configuration of compute nodes, and 3) utilizing only scalable services, we've built a toolkit which allows non-cluster systems experts to easily deploy and maintain their own high-performance cluster.

NPACI Rocks is a collection point for Linux cluster software – we encourage all participation from the community, especially bug fixes, future enhancement suggestions and new software RPMs.

9. Acknowledgments

We gratefully acknowledge the support of Compaq Computer Corporation, and especially our account representative Sally Patchen.

We've benefitted enormously from our collaboration with David Culler and the Millennium Group at UC Berkeley, most notably: Eric Fraser, Matt Massie, Albert Goto, Brent Chun and Philip Buonadonna.

Red Hat has made a fantastic contribution to the community with their Red Hat Package Manager and Kickstart.

We also thank IBM for equipment donations through their Shared University Research program (SUR).

We thank the reviewers for their insightful comments that helped us tune this paper.

And, most importantly, to Caroline, Melissa, and Monica, whose love brings balance.

ID	Model	Name	IP	MAC	Rack	Comment
1	2	frontend-0	192.168.1.1	00:50:8b:a5:4c:f4	0	Gateway machine
3	3	hp-procurve-0	192.168.4.1	00:01:e7:1a:be:00	0	Ethernet switch for Cabinet 0
5	1	compute-0-0		00:50:8b:e0:3a:a7	0	
6	1	compute-0-1		00:50:8b:e0:44:5e	0	
7	1	compute-0-2		00:50:8b:e0:40:95	0	

Table 2. An example of the nodes table stored in the Rocks MySQL database. Some hosts have IP addresses, while most hosts have dynamically generated IP addresses.

References

[1] Ganglia cluster monitoring toolkit. http://www.millennium.berkeley.edu/ganglia/.

[2] Infiniband trade association. http://www.infinibandta.org/.

[3] Parallel and distributed systems software laboratory, rwcp. http://pdswww.rwcp.or.jp/.

[4] Scyld beowulf. http://www.scyld.com/.

[5] 4th Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing. *REXEC: A Decentralized, Secure Remote Execution Environment for Clusters*, Jan. 2000.

[6] C. Antilla, L. Augustin, and B. Biles. Build-to-Order Software: Vision and short term implementation. http://www.valinux.com.

[7] M. Burgess. Cfengine a site configuration engine. volume 8. USENIX Computing Systems, 1995.

[8] B. Callaghan, B. Pawlowski, and P. Staubach. RFC 1813: NFS version 3 protocol specification, June 1995.

[9] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[10] O. C. Group. OSCAR: A packaged cluster software stack for high performance computing. http://www.openclustergroup.org, Jan. 2001.

[11] S. Mehat, Z. Sprackett, D. Johnson, J. Katzung, and C. Haitzler. VACM: Users and programmers manual. http://www.valinux.com.

[12] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF: A parallel workstation for scientific computation. In *Proceedings of the 24th International Conference on Parallel Processing*, pages I:11–14, Oconomowoc, WI, 1995.

[13] T. L. Sterling, J. Salmon, D. J. Becker, Savarese, and D. F. Savarese. *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters*. MIT Press, 1999.

[14] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. PM: An operating system coordinated high performance communication library. In *High-Performance Computing and Networking 97*, 1997.

[15] D. A. P. Thomas E. Anderson, David E. Culler. A case for networks of workstations: NOW. *IEEE Micro*, Feb. 1995.

[16] P. Uthayopas, T. Angsakul, and J. Maneesilp. System management framework and tools for beowulf cluster. In *Proceedings of HPCAsia2000*, Beijing, May 2000.