CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat: Pract. Exper. 2002; **00**:1–20 *Prepared using* cpeauth.cls *[Version: 2001/03/05 v2.01]*

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

Philip M. Papadopoulos,* Mason J. Katz, and Greg Bruno

San Diego Supercomputer Center, University of California San Diego, La Jolla, CA 92093-0505, U.S.A.

SUMMARY

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 distribution takes a fresh perspective on management and installation of clusters to dramatically simplify software version tracking and cluster integration.

NPACI Rocks 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 2.2.1 of the toolkit is available for download and installation. Over 100 Rocks clusters have been built by non-cluster experts at multiple institutions (residing in various countries) providing a peak aggregate of 2 TFLOPS of clustered computing.

KEY WORDS: clusters, beowulf, cluster management, scalable installation, kickstart

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

*Correspondence to: San Diego Supercomputer Center, 9500 Gilman Drive, University of California, San Diego, La Jolla, CA 92093-0505, U.S.A.

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 [2] 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 I) making it a preferred mode for even a modest-sized cluster. Additionally, how many files can really be updated while all services on a node remain online? Some files can be changed while the system is running, but the line is not clear. Clearly, fundamental changes to the operating environment require a reboot (e.g., new kernel, new glibc) and changes to any shared object or service require that *all* processes that are using the file or service in question must terminate before the update can occur to avoid a program crash.

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 a Tokyo-based research group assembled in 1992. RWCP has addressed a wide range of issues in clustering from low-level, high-performance communication [17] to manageability. Their SCore software provides semi-automated node integration using Red Hat's interactive installation tool [18], 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 a distribution provider.
- **Scalable Cluster Environment** - The SCE project is a clustering effort being developed at Kasetsart University in Thailand [19]. 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 a 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). On the whole, SCE and Rocks are orthogonal – the two groups are discussing plans to meld the base OS environment installation features from Rocks with the sophisticated management suite from SCE.
- **Cfengine** - Cfengine [2, 3] is a policy-based configuration management tool that can configure UNIX or NT hosts. After the initial operating environment is installed by

hand or another tool (cfengine doesn't install the base environment), cfengine is used to instantiate the initial configuration of a host and then keep that configuration consistent by consulting a central policy file (accessed through NFS). The central policy file is written in a cfengine-specific configuration language (which resembles makefile syntax) that allows an administrator to define the configuration for all hosts within an administration domain. Each cfengine-enabled host consults this file to keep its configuration current.
To deal with hardware and software heterogeneity, cfengine defines *classes* to delineate unique characteristics.

- **Open Cluster Group** - The Open Cluster Group has released their OSCAR toolkit [14]. 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.

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 [15]. 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 18 months, the Rocks-based "Meteor" cluster at SDSC, has evolved from a homogeneous system to one that has seven different types of nodes, two different CPU architectures, manufactured by three vendors with three 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. A framework of XML files is used to describe core components of the base operating environment. Each component specifies one or more Red Hat packages and optionally a post installation script. Components are drawn together based on the type of system one wishes to install. Although we have used this idea to break down Red Hat Kickstart files into a more managable form, the end result for node installation is a Red Hat compliant text-based Kickstart file. By applying standard programming techniques to monolithic Kickstart files we have created a single framework of XML files to describe all variants of hardware (e.g., x86 and IA-64) and software (e.g., compute, frontend, NFS server, web server) 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 often demand homogeneous hardware or, even worse, a specific brand of homogeneous hardware. This reduces choice for a cluster user and 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. Often, cluster administrators feel compelled to customize the kernel to support high-performance computing. For us, the stock

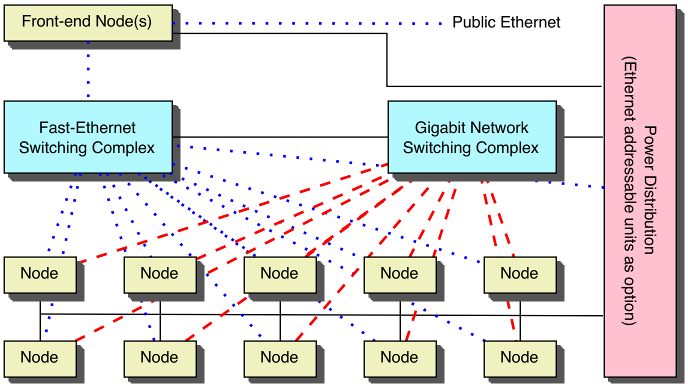

Figure 1. Rocks hardware architecture. Based on a minimal traditional cluster architecture.

Red Hat kernel has served us well – it has supported various hardware platforms and delivered acceptable performance to a variety of scientific applications (e.g., GAMESS [22], AMBER [23], NAMD [24] and NWChem [25]).

We acknowledge that kernel customization can increase application performance and is necessary in order to support unique hardware. For these cases, we have expanded upon Red Hat's additions to the standard Linux kernel makefile that includes an `rpm` target. To include a new kernel RPM into Rocks, the cluster administrator crafts a `.config` file, rebuilds the kernel RPM (with `make rpm`), copies the resulting kernel binary package back to the frontend machine and binds it into a new distribution (using `rocks-dist`, described in Section 6). Then the new kernel RPM is instantiated on all desired nodes by simply reinstalling them.

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 [4] and popularized by the Beowulf project [1]. 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. 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. † 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 administrator 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 6.3). 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 messages 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*, The Portland Group's High Performance Fortran compilers and Intel's Linux C/C++ and Fortran compilers.

Some of the libraries found on the frontend are basic linear algebra subprograms (ATLAS [20] from UTK's Innovative Computing Laboratory and Intel's *Math Kernel Library*) 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.

†A hard power cycle on a Rocks compute node forces the node to reinstall itself.

For interactive and 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 programmable methods for adding or updating packages.

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) [7]. The frontend node exports all user home directories to compute nodes via NFS. We are searching for an alternative that is scalable, fault-tolerant and transparent (i.e., exports the ubiquitous interface `open/read/write/close`).

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. Actively Managing Node Configuration with HTTP and XML

All nodes are installed using Red Hat's kickstart tool which is driven by a text-based configuration file. This text file contains all the package names to install and "post processing" commands. Traditionally, kickstart files are static entities created by system adminstrators and tailored, by hand, for each machine type.

In Rocks, we actively manage kickstart files by building them on-the-fly with a CGI script. This script merges two major functions to produce the resulting kickstart file: it constructs a general configuration file from a set of XML-based configuration files and applies node-specific parameters by querying a local SQL database.

```
<?XML VERSION="1.0" STANDALONE="no"?>
<KICKSTART>
        <DESCRIPTION>Setup the DHCP server for the cluster</DESCRIPTION>
        <PACKAGE>dhcp</PACKAGE>
        <POST>
                <!-- tell dhcp just to listen to eth0 -->
                awk '                                                   \
                        /^DHCPD_INTERFACES/ {                           \
                                printf("DHCPD_INTERFACES=\"eth0\"\n");  \
                                next;                                   \
                        }                                               \
                        {                                               \
                                print $0;                               \
                        } ' /etc/sysconfig/dhcpd &gt; /tmp/dhcpd
                mv /tmp/dhcpd /etc/sysconfig/dhcpd
        </POST>
</KICKSTART>
```

Figure 2. XML node file: DHCP server configuration.

The XML-based configuration files are divided into two types, *nodes* and *graphs*. A *node file* is a small single-purpose module that specifies the packages and per-package post configuration commands for a specific service. † For example, Figure 2 is the node file for a DHCP server.

An XML-based *graph file* links all the defined modules together with directed edges. An edge represents a relation between two modules. The roots of the graph represent "appliances", such as *compute* and *frontend*. Figure 3 is a snippet from the graph file and Figure 4 visualizes the graph components: appliances, modules and the relationship between modules (the edges). This single XML-based software installation infrastructure "describes" *all* node behaviors.

At installation time, a machine requests its kickstart file via HTTP from a CGI script on the frontend server. This script uses the requesting node's IP address to drive a series of SQL queries that determine the appliance type, software distribution, and localization of the node.

The script then parses the XML graph file and traverses it, parsing all the node files based on the appliance type. Using Figure 4 as an example, if the machine was configured to be a *compute* appliance, the traversal of the graph would be the *compute*, *mpi*, and *c-development* node files. The end result of the parser is a Red Hat compliant text-based kickstart file which is returned to the requesting machine.

This method has proven to be extremely flexible, as heterogeneous hardware is no harder to support than homogeneous. For example, in our cluster at SDSC, one XML graph file supports the dynamic kickstart file generation for three processor types (IA-32, Athlon and IA-64), three

†We develop and distribute the default set of node and graph files that are automatically installed when a user creates a frontend node. Users can modify (or add) a node or graph file to tailor the cluster to their needs (explained in Section 6.2.3).

```
<?XML VERSION="1.0" STANDALONE="no"?>
<GRAPH>
        <DESCRIPTION>Default Graph for NPACI Rocks</DESCRIPTION>

        <EDGE FROM="frontend" TO="mpi-devel"/>
        <EDGE FROM="frontend" TO="dhcp-server"/>

        <EDGE FROM="compute" TO="mpi"/>
        <EDGE FROM="compute" TO="c-development"/>

        <EDGE FROM="mpi-devel" TO="c-development"/>
        <EDGE FROM="mpi-devel" TO="fortran-development"/>
</GRAPH>
```

Figure 3. An excerpt from the XML graph file.

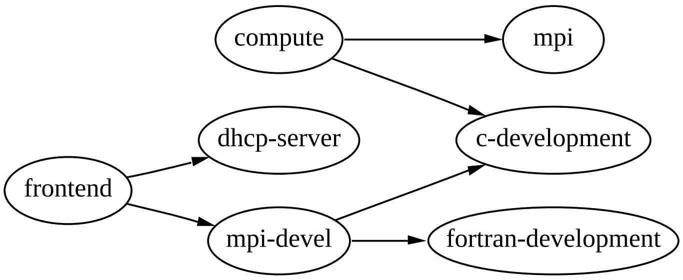

Figure 4. A visualization of the XML graph description.

storage types (SCSI, IDE and integrated RAID devices) and two network types (Ethernet and Myrinet).

6.2. 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.

6.2.1. 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 distribution 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 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 (detailed in Section 6.3).

6.2.2. Extensibility

Figure 5 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 repeatability – a Rocks distribution can be run through the identical process to produce an enhanced Rocks distribution. This allows a user, 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 6.

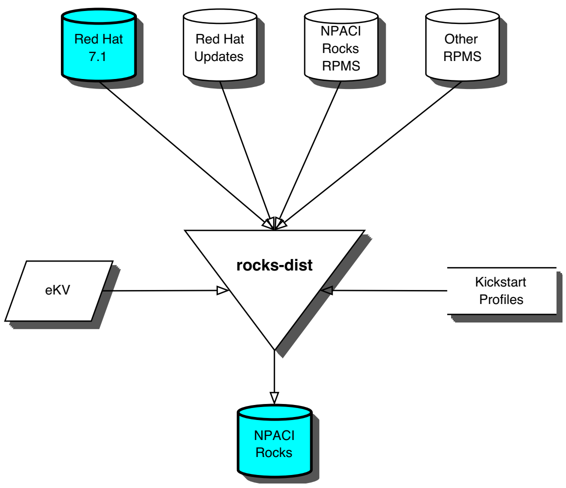

Figure 5. Building a Rocks distribution with rocks-dist.

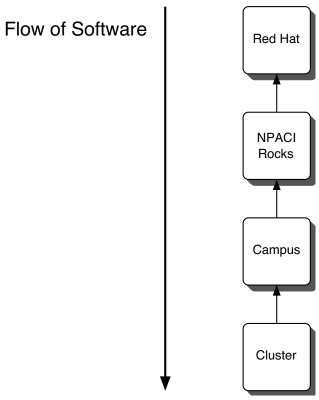

Figure 6. Object-oriented model of rocks-dist.

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.

6.2.3. 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 6) and creates a new tree comprised mostly of symbolic links to the mirrored software. Inside this tree is a build directory that contains the XML configuration infrastructure. Users can customize this new distribution (by editing the XML modules or graph) and manually adding new RPMs that rocks-dist did not integrate. By creating multiple distributions and editing the XML configuration infrastructure, 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.

6.3. 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 7).

Using HTTP to distribute RPMs scales well. It offers many simple methods in which to support our software management strategies for large clusters. Table I shows the total time to reinstall a number of nodes concurrently from a single HTTP server. In this experiment, the HTTP server is a dual 733 MHz PIII with 100 Mbit Ethernet and the compute nodes are 733

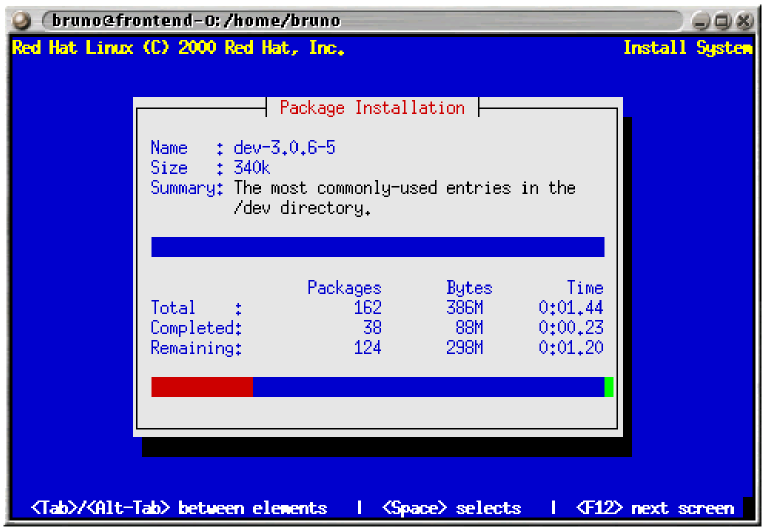

Figure 7. Shoot-node and eKV. Red Hat's Kickstart screen is redirected over Ethernet.

MHz - 1 GHz PIIIs with Myrinet. Times include the time taken to rebuild the Myrinet driver. Each node transfers approximately 225 MB of data from the server. In this experiment, of the total time (600 seconds), approximately 223 seconds is devoted to downloading and installing RPMs (the remainder of the time is spent in rebooting and post configuration), therefore, each reinstalling node demands 1 MB/sec of network bandwidth (225 MB / 223 sec). Assuming the web server can utilize 70% of the 100 Mbit Ethernet connection, (i.e., provide 7 MB/sec), the web server described above should be able to support 7 concurrent reinstallations at full speed. By running a micro-benchmark that consisted of serially downloading all the RPMs a compute node downloads during its reinstallation, we found the web server sourced 7-8 MB/s. Additionally, our empirical data validates the above model as an 8-node concurrent reinstallation takes almost the same time as a 1, 2 and 4-node reinstallation.

In order to support full-speed reinstallation at scale, many simple methods can be employed. First, one can scale the network connection to the web server. By adding a Gigabit Ethernet connection to the web server, it will theoretically be able to support 10 times the number of concurrent full-speed reinstallations. †

†In practice, Gigabit Ethernet will support 7.0-9.5 times the number of concurrent full-speed reinstallations over Fast Ethernet. [26]

Table I. Reinstallation performance. The HTTP server is a dual 733 MHz PIII with 100 Mbit Ethernet. Compute nodes are 733 MHz - 1 GHz PIIIs with Myrinet. Times include the time taken to rebuild the Myrinet driver. Each node transfers approximately 225 MB of data from the server.

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

Another method is to replicate the web server and use HTTP load balancing (many hardware and software solutions exist). † By deploying N web servers, one can support N times the number of concurrent full-speed reinstallations that a single web server can support. ‡

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 – over the last year, there have been 16 updates to the "stable tree" (kernel version 2.4). 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 heavy-weight solution adds only a 20-30% time penalty on reinstallation.

6.4. Deriving Cluster Configuration Via an SQL Interface

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.

†Replicating an installation web server is straightforward – downloading RPMs is strictly read only.
‡Disk performance is a non-issue. 225 MB of RPMs easily fits into 512 MB of main memory (and one GB of main memory for servers is common).

Table II. An example of the *Nodes* table stored in the Rocks MySQL database.

ID	MAC	Name	Membership	Rack	Rank	IP	Comment
1	00:30:c1:d8:ac:80	frontend-0	1	0	0	10.1.1.1	Gateway machine
2	00:01:e7:1a:be:00	network-0-0	4	0	0	10.255.255.253	Switch for Cabinet 0
3	00:50:8b:a5:4d:b1	nfs-0-0	7	0	0	10.255.255.249	NFS Server in Cabinet 0
4	00:50:8b:e0:3a:a7	compute-0-0	2	0	0	10.255.255.245	Compute node
5	00:50:8b:e0:44:5e	compute-0-1	2	0	1	10.255.255.244	Compute node
6	00:50:8b:e0:40:95	compute-0-2	2	0	2	10.255.255.243	Compute node
7	00:50:8b:e0:40:93	compute-0-3	2	0	3	10.255.255.242	Compute node
8	00:50:8b:c5:c7:d3	web-1-0	8	1	0	10.255.255.246	Web Server in Cabinet 1

Rocks clusters use a MySQL database for site configuration. The two key tables we provide are, 1) a site-specific configuration table and, 2) a nodes table (Table II). 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 network 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, determines the next *free* IP address, binds the hostname and IP address to its Ethernet MAC address, and inserts this information into the database. Insert-ethers then rebuilds service-specific configuration files by running queries against the database, and restarting the respective services.

We, like many people who run parallel machines [21], have our own set of rudimentary scripts to interactively control and monitor the nodes. Our first version of our script that killed runaway processes on compute nodes (`cluster-kill`), generated a list of nodes by matching the prefix *compute-* on each entry in `/etc/hosts`. This works fine on small clusters, or when there is a job running on all the compute nodes, but when a runaway job is on a subset of the nodes (e.g., running on the nodes in one cabinet), or when some nodes are down, the brute-force method quickly becomes tiring. By simply adding an SQL interface to the script makes it more powerful as the user can intelligently direct the script to a subset of the nodes. For example, if the user knows that all the runaway processes are in cabinet 1, then a little finesse can be added to cluster-kill by:

†Nodes are booted sequentially in order for insert-ethers to bind hostnames to physical locations, as insert-ethers associates a *rack* and *rank* with each hostname (see Table II). The serial nature of this procedure is only required when installing nodes. This procedure can be executed in parallel if a node's physical location is unimportant.

Table III. An example of the *Memberships* table.

ID	Name	Appliance	Compute
1	Frontend	1	no
2	Compute	2	yes
3	External	1	no
4	Ethernet Switches	4	no
5	Myrinet Switches	4	no
6	Power Units	5	no

```
cluster-kill --query="select name from nodes where rack=1" bad-job
```

Any SQL query, including joins, can be fed to cluster-kill. Another table defined in the Rocks database is the *memberships* table (Table III). By using the **Compute** field of the memberships table and *joining* it with the **Membership** field of the nodes table, one can kill a runaway job only on the compute nodes with the following:

```
cluster-kill --query="select nodes.name from nodes,memberships where \
     nodes.membership = memberships.id and \
     memberships.name = 'Compute'" bad-job
```

This is a trivial example, but it illustrates the potential power of multi-table joins.

7. Current Status and Future Work

As of June 2002, the latest Rocks release (version 2.2.1) is:

- Red Hat 7.2 base plus Red Hat's 327 security advisories and bug fixes.
- Assorted community software (MPICH, PVM, ATLAS BLAS, Intel math kernel library, etc.).
- NPACI software (the software described in this paper).

Rocks has been used to install 12 clusters on the UCSD campus (including two 256-processor clusters at the Scripps Institute of Oceanography) and at least a dozen groups from around the world have installed their own Rocks clusters (e.g., 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" a dozen 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. In response to a request we placed on our email discussion list, we've calculated that Rocks drives over 2 TFLOPS (peak) of clustered computing.

Rocks is installed with a floppy and a CD and the frontend Kickstart file is built from a simple web form (links to both the ISO image and web form are found at **http://rocks.npaci.edu/**). 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 [13].

We look forward to integrating future clusters. An announced cluster which will be running Rocks is a prototype machine for the Grid Physics Network (GriPhyN) project. The GriPhyN project is tasked with building a scalable computational environment that will drive Petabyte-scale storage. 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.

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 distribution which allows non-cluster systems experts to easily deploy and maintain their own high-performance cluster.

Many of the underlying mechanisms and techniques described in this paper are not new – they are well-understood, and often, widely deployed. What is new is the combination of these mechanisms in order to build a cluster-aware Linux distribution that, through one mechanism (reinstall), simplifies cluster management (which serves the application scientist) *and* promotes experimentation (which serves the cluster developer).

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.

ACKNOWLEDGEMENTS

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.

REFERENCES

1. Sterling T, Savarese D, Becker DJ, Dorband JE, Ranawake UA, Packer CV. BEOWULF: A Parallel Workstation for Scientific Computation. *Proceedings of the 24th International Conference on Parallel Processing* 1995; **I**:11–14.
2. Burgess M. Cfengine A Site Configuration Engine. *USENIX Computing Systems* 1995; **8**(3).
3. Burgess M. Recent Developments in Cfengine. *Unix.nl Conference, The Hague* 2001.
4. Anderson TE, Culler DE, Patterson DA. A Case for Networks of Workstations: NOW. *IEEE Micro* 1995; February.
5. Chun BN, Culler DE. REXEC: A Decentralized, Secure Remote Execution Environment for Clusters. *4th Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing* 2000; January.
6. Foster I, Kesselman C. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing* 1997; **11**(2):115–128.
7. Callaghan B, Pawlowski B, Staubach P. RFC 1813: NFS Version 3 Protocol Specification. ftp://ftp.internic.net/rfc/rfc1094.txt [June 1995].
8. Alexander S, Droms R. RFC 2132: DHCP Options and BOOTP Vendor Extensions. ftp://ftp.internic.net/rfc/rfc1533.txt [March 1997].
9. Antilla C, Augustin L, Biles B. Build-to-Order Software: Vision and Short Term Implementation. http://www.valinux.com.
10. Mehat S, Sprackett Z, Johnson D, Katzung J, Haitzler C. VACM: Users and Programmers Manual. http://www.valinux.com.
11. Maui Scheduler. http://mauischeduler.sourceforge.net.
12. Portable Batch System. http://pbs.mrj.com/.
13. Scyld Beowulf. http://www.scyld.com/.
14. Open Cluster Group. OSCAR: A packaged cluster software stack for high performance computing. http://www.openclustergroup.org [January 2001].
15. Sterling T, Salmon J, Becker DJ, Savarese D. *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters.* MIT Press, 1999.
16. Ishikawa Y, Tezuka H, Hori A, Sumimoto S, Takahashi T, O'Carroll F, Harada H. RWC PC Cluster II and SCore Cluster System Software–High Performance Linux Cluster. *Proceedings of the 5th Annual Linux Expo* 1999; 55-62.
17. Tezuka H, Hori A, Ishikawa Y, Sato M. PM: An Operating System Coordinated High Performance Communication Library. *High-Performance Computing and Networking 97* 1997.
18. Parallel and Distributed Systems Software Laboratory, RWCP. http://pdswww.rwcp.or.jp/.
19. Uthayopas P, Angsakul T, Maneesilp J. System Management Framework and Tools for Beowulf Cluster. *Proceedings of HPCAsia2000*, Beijing, China, 2000; May.
20. Whaley C, Petitet A, Dongarra J. Automated Empirical Optimization of Software and the ATLAS Project. *Parallel Computing*, May/June 2001; 22-29.
21. Ong E, Lusk E, Gropp W. Scalable Unix Commands for Parallel Processors: A High-Performance Implementation. To appear at *Euro PVM-MPI 2001 (PVM-MPI 01)*, Santorini (Thera) Island, Greece, 17-20 April 2002.
22. General Atomic and Molecular Electronic Structure System (GAMESS). http://www.msg.ameslab.gov/GAMESS/GAMESS.html.
23. Assisted Model Building with Energy Refinement (AMBER). http://www.amber.ucsf.edu/amber/amber.html.
24. NAMD - Scalable Molecular Dynamics. http://www.ks.uiuc.edu/Research/namd/.
25. NWChem - High Performance Computational Chemistry Software. http://www.emsl.pnl.gov:2080/docs/nwchem/nwchem.html.
26. Loeb ML, Rindos AJ, Holland WG, Woolet SP. Gigabit Ethernet PCI Adapter Performance. *IEEE Network* 1995; March-April 2001, **15**(2).