Configuring Large High-Performance Clusters at Lightspeed: A Case Study

Philip M. Papadopoulos[†], Caroline A. Papadopoulos[‡], Mason J. Katz[†], William J. Link[†] and Greg Bruno[†]

[†] The San Diego Supercomputer Center
University of California San Diego
La Jolla, CA 92093-0505
{phil,mjk,bill,bruno}@sdsc.edu
http://www.rocksclusters.org

[‡] Physical Oceanography Research Division
Scripps Institution of Oceanography
University of California, San Diego
La Jolla, CA 92093-0505
caroline@ucsd.edu

April 11, 2003

Abstract

Over a decade ago, the TOP500 list was started as a way to measure supercomputers by their sustained performance on a particular linear algebra benchmark. Once reserved for the exotic machines and extremely well-funded centers and laboratories, commodity clusters now make it possible for smaller groups to deploy and use high-performance machines in their own laboratories. This paper describes a weekend activity where two existing 128-node commodity clusters were fused into a single 256-node cluster for the specific purpose of running the benchmark used to rank the machines in the TOP500 supercomputer list. The resulting metacluster sits on the November 2002 list at position 233. A key differentiator for this cluster is that it was assembled, in terms of its software, from the NPACI Rocks open-source cluster toolkit as downloaded from the public website. The toolkit allows non-cluster experts to deploy and run supercomputer-class machines in a matter of hours instead of weeks or months. With the exception of recompiling the University of Tennessee's Automatically Tuned Linear Algebra Subroutines (ATLAS) library with a recommended version of the GNU C compiler, this metacluster ran a "stock" Rocks distribution. Successful first-time deployment of the fused cluster was completed in a scant 6 hours. Partitioning of the metacluster and restoration of the two 128-node clusters to their the original configuration was completed in just over 40 minutes.

This paper describes early (pre-weekend) benchmark activities to empirically determine reasonably good parameters for the High-Performance Linpack (HPL) code on both Ethernet and Myrinet interconnects. It fully describes the physical layout of the machine, the description-based installation methods used in Rocks to re-deploy two independent clusters as a single cluster, and gives the benchmark results that were gathered over the 40-hour period allotted for the complete experiment. In addition, we describe some of the online monitoring and measurement techniques that were employed during the experiment. Finally, we point out the issues uncovered with a commodity cluster of this size.

The techniques presented in this paper truly bring supercomputers into the hands of the masses of computational scientists.

1 Introduction

The TOP500 [10] [3] is a open competition that, every six months, ranks the top 500 entries in descending order according to the results of running the LINPACK benchmark. The LINPACK benchmark solves a dense system of linear equations and outputs the number of floating point operations per second achieved during the calculation.

In 1995, the pioneers of commodity clusters, notably Culler [1] and Sterling [8], observed the technology trends and theorized that one day machines built from commodity components would one day sit among the world's fastest machines. Although the first cluster appeared on the TOP500 list in 1998 and proved the theory correct, this paper presents how NPACI Rocks [7], the open-source cluster toolkit, was utilized to put a cluster on the November 2002 TOP500 in just one weekend.

2 Rocks Overview

In November of 2000, the SDSC Grids and Clusters group released the first version of the NPACI Rocks cluster toolkit. This software was the initial result of our work to make it possible for application scientists to build and manage their own commodity clusters. Although this first version still required some cluster expertise to build a cluster, we had greatly simplified the task due in part to our collaboration with the UC Berkeley Millennium Project led by David Culler.

As our software has matured, we have reached the point where non-cluster experts can indeed build and manage their own clusters. Application scientists have used the toolkit to build clusters ranging from small-scale testbeds (e.g., 8-16 nodes) to world-class resources. This section gives an overview of the design and technologies that enable anyone to do what was once the exclusive domain of dedicated supercomputer centers and well-funded government laboratories.

2.1 Hardware Architecture

Figure 1 shows a traditional architecture commonly used for high-performance computing clusters as pioneered by the Network of Workstations project [1] and popularized by the Beowulf project [8]. This system is composed of

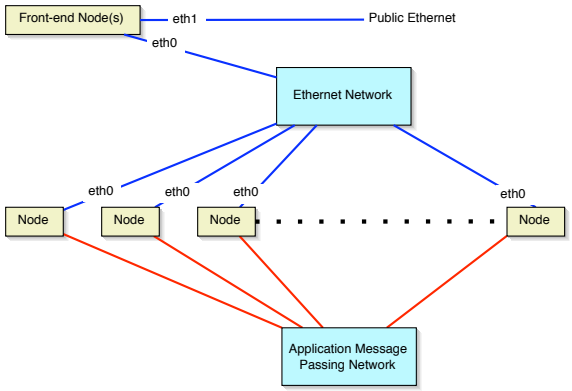


Figure 1: **Rocks hardware architecture.** Based on a minimal traditional cluster architecture.

standard high-volume servers, an Ethernet network and an optional off-the-shelf high-performance cluster interconnect (e.g., Gigabit Ethernet or Myrinet). We have 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.

2.2 Software Architecture

NPACI Rocks is a complete cluster-aware Linux distribution based upon Red Hat with additional packages and programmed configuration to automate the deployment of high-performance Linux clusters[1]. The Red Hat distribution was chosen because of two key mechanisms found within it: 1) It's software packaging tool (RPM), and 2) it's script-driven software installation tool that describes a node's software stack (kickstart). By utilizing RPM and kickstart, we've developed mechanisms that support fully-automated node installation – an important property when scaling clusters. Details on both RPM and kickstart are found in section 2.3.

Although the focus of Rocks is on flexible rapid system configuration (and re-configuration), the steady-state behavior of Rocks has the look and feel much like any other commodity cluster containing *de-facto* cluster standard services (e.g., PBS, Maui, Ganglia and MPI).

[1]For a summary of other cluster-building tools, see [6].

2.2.1 Description-Based Software Installation

The process of installing software on any system always contains two components: the installation of software packages and the configuration of software packages. Often the configuration of a package is to simply accept the defaults, and other times a wildly different configuration from the default behavior is required. The traditional single desktop approach to this process is to install software and then, through a process of manual data entry, the software is configured to one's requirements. A common extension of this process to the parallel world of clusters, is to hand configure a single node and replicate state of this "golden image" onto all the nodes in a cluster. Although this works with homogeneous hardware and static cluster functional requirements, we have found that clusters rarely have either of these attributes.

Rocks treats software installation and software configuration as separate components of a single process. This means that hand configuration required to build a "golden image" is instead automated. The key advantage is the dissemination of intellectual property. Building cluster "golden images" is more often than not an excercise in replicating the work done by other cluster builders. By automating this configuration task out-of-band from system installation, the software configuration is specified only once for all the Rocks clusters deployed world-wide[2].

Installation of software packages is done in the form of package installs according to the functional role of a single cluster node. This is also true for the software configuration. Once both the software packages and software configuration are installed on a machine, we refer to the machine as an **appliance**. Rocks clusters often contain *Frontend*, *Compute*, and *NFS* appliances. A simple object-oriented framework (expressed in XML) is used to allow cluster architects to define new appliance types and take full advantage of code re-use for software installation and configuration [6].

2.2.2 Staying on the Software Curve

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.

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 official Red Hat distribution (including updates), Rocks-developed software, other Rocks community-developed software and third-party software. This set of software is compiled to create a new distribution that includes only the most recent versions of all the software packages[3]. The resulting distribution has the identical structure as a Red Hat distribution only with some new and some updated packages.

By managing the software packages independently from software configuration the same configuration can be applied to different software distributions. Or in the case of cluster testbeds, the same software distribution can be used with multiple software configurations. We have used both of these models in the software development and production roles of NPACI Rocks.

2.3 Software Installation

Kickstart is the Red Hat provided mechanism for automating system installation. Kickstart, together with Red Hat's software packaging format (RPM), has enabled cluster builders to specify *a priori* the exact software package and software configuration of a system. This textual description is then used to build a software image on the target platform. Although managing a single (or set of) Kickstart files can be simpler than managing cluster "golden images", Kickstart is limited in terms of programability. The lack of a macro language and a code re-use model potentially requires a unique Kickstart file for every node in a cluster.

Rocks solves this problem by generating Kickstart files

[2] A site-specific cluster database is used to customize individual clusters with local configuration variables.

[3] There is also a mechanism to force a specific version of a software package to be used.

on-the-fly based on the programmatic target system description, and the site-specific configuration data stored in an SQL database. Integrating a cluster node requires the node to boot an installation kernel using network boot (PXE) or a physical boot media (e.g., CDROM, hard disk or floppy). This installation kernel requests a Kickstart file over HTTP and executes the Kickstart file installing the appropriate software packages and applying software configuration. Although the intended and most common implementation of this process is to serve static Kickstart files, Rocks replaces the static file with a script (CGI) to dynamically produce a node-specific Kickstart file.

3 TOP500 Configuration

3.1 Hardware

At the Scripps Institution of Oceanography (SIO), there are two independent, but nearly identical, 128-node commodity clusters – both are used by Dr. Detlef Stammer's research groups to explore ocean state estimation techniques using an adjoint model formulation. The 128 compute nodes in the first cluster (named 'Ecco'), are IBM xSeries 330 rack-mounted servers. Each compute node has the following configuration:

- Dual 1-GHz Intel Pentium III nodes, that is, there are two 1-GHz Intel Pentium III CPUs per server (for a total of 256 CPUs)
- 1 GB of main memory (PC133)
- ServerWorks LE chipset
- 18.2 GB SCSI disk drive
- 100 Mbit Ethernet port
- Myrinet PCI host copper-based interface (M3S-PC164B-2)

There are 4 racks, each containing 32 compute nodes. In each rack, the nodes' 100-Mbit Ethernet links are connected to a set of stacked Intel Express 10/100 switches (530T and 535T). The stacked switch in each rack is connected via Gigabit Ethernet to a central Intel Netstructure 470T Gigabit Ethernet switch. A 128-port Myrinet switch (M3-E128) connects all the compute nodes' Myrinet adapters. This network is used for application message passing, as shown in Figure 1. All the compute nodes are connected via Ethernet to a frontend. The frontend has a similar configuration to the compute nodes, except it has two 36.4 GB disk drives and does not contain a Myrinet interface. The second 128-node cluster (named 'Compas'), is identical to the Ecco cluster except that fiber-based connections are used for the Myrinet network instead of copper.

Upon arrival at the machine room on Friday evening, the first order of business was to physically merge the two clusters into one metacluster (called 'Compas-Ecco'). Both frontends were shutdown and a new node was connected to serve as the single head node for the Compas-Ecco metacluster. Then a single metacluster Ethernet network was built by connecting both cluster's private-side networks. This was done with a single gigabit link connected to each of the cluster's central switches (Intel Netstructure 470T).

Next, a single metacluster Myrinet network was built by removing one 8-port fiber blade from the Compas switch and replacing it with an 8-port copper blade. After decommissioning 8 cluster nodes from each cluster (bringing the total node count to 240), a total of 16 copper ports were available for a cross-connect. Connecting both switches with 8 copper cables, transformed two disjoint full-bisection Myrinet networks (128 Gbit/sec) into a single fat-tree network with a bisection bandwidth of 16 Gbits/sec. The configuration is illustrated in Figure 2.

3.2 Software

After physically merging the clusters, efforts turned toward software installation. The new frontend was installed with "stock" Rocks version 2.2.1 which took about 30 minutes.

A Rocks frontend is installed with 640 software packages (RPMs). One of these packages is ATLAS [11] – the Automatically Tuned Linear Algebra Subroutines from the Innovative Computing Laboratory at the University of Tennessee which provides a library of efficient mathematical functions (e.g., matrix multiply). Rocks is built upon stock Red Hat releases and, to simplify maintainability, strives to use all the packages that Red Hat provides. Rocks version 2.2.1 is based on Red Hat 7.2 which includes gcc version 2.96 which, in turn, is known to

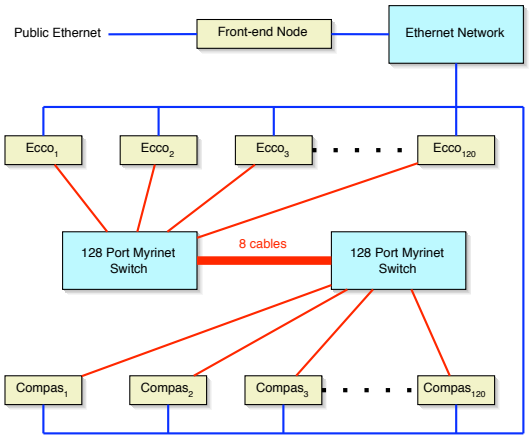


Figure 2: **Myrinet-connected Metacluster.** In Ecco's and Compas's original configuration, 128 compute nodes are connected to a single Myrinet switch. Unifying the Myrinet networks for the metacluster required switch-to-switch cross connections. This was accomplished by removing 8 nodes from Ecco and 8 nodes from Compas and using those ports as cross connections, thus, the resulting Myrinet-connected metacluster contains 240 nodes instead of 256.

produce object code that is significantly slower (25-75% degradation) than other versions of gcc. Based on advice from the ATLAS developers, we chose to rebuild the ATLAS package with gcc 2.95 as ATLAS was originally tuned using that version. We installed the new ATLAS package on the frontend, rebuilt the High-Performance Linpack (HPL) [5] package (to relink it with the new ATLAS libraries), then used `rocks-dist` to compile a new distribution. This distribution served as the software stack for all compute node installations. Of the 640 packages installed on a Rocks frontend, these were the only two packages that differ from a stock installation[4].

Then, we started installing compute nodes. It was determined that a fresh installation of all nodes would easily assure that all of the cluster nodes had a common software configuration [5]. By powering up each cabinet one at a time, the frontend assigns physical location information to each compute node (in this case, cabinet location). The frontend (100-Mbit Ethernet network) identified and completed installation of 32 nodes in about 20 minutes. Initial install of the 256-node cluster in this phased manner of one-rack-at-a-time consumed about 2.5 hours of wall clock time.

4 Tuning

Once ATLAS and HPL were rebuilt and installed, empirical testing was used to find parameters that delivered a majority of the cluster's peak performance. First, HPL's *blocking factor* (HPL's `NB` parameter [6]) was examined. After initial runs on a single node, we found a blocking factor of 80 delivered good results (see Table 1).

After finding an acceptable blocking factor for a single node, empirical experiments were run to determine which blocking factors and matrix sizes would work well across multiple nodes. In a tuning document, the authors of HPL remark that small multiples of the blocking factor (`NB`) are likely to work well as the size of the problem (`N`) is scaled

[4]This action will be unnecessary in the next release of Rocks, as it will be based on Red Hat 8.0 which contains gcc version 3.2.

[5]At the end of this experiment, nodes were reinstalled with configurations from their original frontends.

[6]This is the size of the sub-matrix that HPL internally uses during its computation. The key is to find a value that best uses the processor's memory hierarchy as HPL will reuse data within the blocking factor during its computation phase.

NB	Seconds	GFlops	% of Peak
64	328.04	1.118	55.90
80	**289.65**	**1.266**	**63.30**
100	292.35	1.254	62.70

Table 1: **Finding the Blocking Factor on a Single Node.** The matrix size for all tests is 8,192. The Pentium III is a single-issue machine (that is, it can issue at most one instruction per clock cycle), therefore, peak performance per 1 GHz CPU is 1 GFlop. Both CPUs on the compute node were used for this test, therefore, peak performance for the node is 2 GFlops.

up. In Table 2, one observes that the single-node `NB` value of 80 delivers relatively good results, but as the problem size scales, a blocking factor of 160 (a small multiple of 80) further improves performance by approximately 6% on a 16K X 16K Matrix.

After settling on a blocking factor, we investigated what problem size should be attempted. The starting point was looking at various matrix sizes on a single node. By examining the results in Table 3, a (sub)matrix size between 8,192 and 10,000 was selected for each node. These submatrix sizes fill most of the available 1 GB memory on each node.

Then we scaled the problem size up across multiple nodes. In Table 4, a per-node matrix size of 10,000 delivers the highest performance on our compute node configuration. This followed expectations as a 10,000 x 10,000 matrix of 64-bit floating-point values represents approximately 800 MB of storage and as mentioned above, each compute node contains 1 GB of main memory. Therefore, a 10,000 x 10,000 matrix will occupy approximately 80% of main memory. By leaving 20% of main memory for operating system functions, this matrix size strategy avoids swapping by the virtual memory subsystem which would negatively affect system performance. In general, if one starts with an $P \times Q$ processor mesh, then the overall matrix size chosen is approximately $10000 \times \sqrt{PQ}$.

Although the HPL benchmark scales quite well, it does benefit from faster networks. Results from using the 100-Mbit Ethernet network are in Table 5 and should be compared to the Myrinet results in Table 6. HPL performance over the 100-Mbit Ethernet network dropped 27% as the problem scaled from 2 to 32 CPUs. It appeared that if the Ethernet network was used for the final run, the resulting benchmark would simply be too slow. [7]

When scaling HPL from 2 to 32 CPUs using Myrinet, we observed a more acceptable decrease in performance of 7%. Myrinet was therefore the interconnect used for HPL.

5 Results

5.1 TOP500 Linpack Results

After tuning runs, jobs were submitted to the 240-node (480 CPU) Myrinet-connected metacluster using `mpirun`, the job launcher for Message Passing Interface (MPI) [4]. Because 16 Myrinet ports were used to cross-connect the clusters, a total of 240 nodes (instead of 256) were actually interconnected. Table 7 shows the results of scaling the problem size up to 240 CPUs.

While satisfied with 284 GFlops, there was still a half day before the original clusters (and configurations) were to be returned to the SIO researchers. Further experiments with the blocking factor (`NB`) and broadcast (`BCAST`) parameters (see Table 8) were conducted. The broadcast parameter determines how results from the CPUs are communicated among each other. There are six settings for `BCAST` but the HPL documentation states that settings 1, 3 and 4 should work well. Due to the long execution time of the benchmark, we chose not to experiment with broadcast setting 4.

By using a blocking factor of 200 and a broadcast parameter of 3, the final result of 285.9 GFlops was

[7] Which turned out to be true. The last entry in the November 2002 TOP500 list is 195.8 GFlops. Had the efficiency of HPL dipped to 38%, the 512 CPU metacluster would have registered only 194.56 GFlops.

CPUs	Matrix Size (N)	NB	Seconds	GFlops	% of Peak
4	8,192	64	188.09	1.949	48.73
4	8,192	80	171.35	2.140	53.50
4	8,192	100	176.03	2.083	52.08
4	8,192	120	162.95	2.250	56.25
8	16,384	80	733.61	3.997	49.96
8	16,384	120	704.45	4.163	52.04
8	**16,384**	**160**	**692.56**	**4.234**	**52.93**
8	16,384	180	707.23	4.146	51.83
8	16,384	200	697.88	4.202	52.53
32	40,000	125	2756.69	15.48	48.38
32	40,000	140	2763.76	15.44	48.25
32	**40,000**	**160**	**2674.72**	**15.95**	**49.84**
32	40,000	180	2728.98	15.64	48.88

Table 2: **Finding the Blocking Factor Across Multiple Nodes.** The matrix size is `N` x `N`. The blocking factor is `NB`. As the problem size scales, a blocking factor of 160 delivers the best results.

Matrix Size (N)	Seconds	GFlops	% of Peak
2,048	5.08	1.129	56.45
4,096	36.65	1.261	63.05
8,192	**274.38**	**1.336**	**66.80**
10,000	**503.78**	**1.324**	**66.20**

Table 3: **Determining the Problem Size for a Single Node.** All tests are conducted on both CPUs on a single node. The blocking factor (`NB`) is 160. A matrix size between 8,192 and 10,000 delivers good results.

Per-Node Matrix Size	Total Matrix Size (N)	Seconds	GFlops	% of Peak
2,048	23,170	74.49	111.3	43.48
4,096	46,340	473.08	140.2	54.77
8,192	92,681	3311.59	160.3	62.62
10,000	**113,137**	**5869.91**	**164.5**	**64.26**

Table 4: **Determining the Problem Size Across Multiple Nodes.** This test was conducted on 128 nodes (256 CPUs) with a blocking factor (`NB`) of 160. A per-node matrix size of 10,000 delivers the highest performance.

CPUs	Matrix Size (N)	Seconds	GFlops	% of Peak
2	8,192	289.65	1.266	63.30
8	16,384	692.56	4.234	52.93
32	32,768	1584.16	14.81	46.28

Table 5: **HPL Performance Over the 100-Mbit Ethernet Network.** The blocking factor (NB) is 160. Scaling the problem up from 2 CPUs to 32 CPUs results in a decrease in efficiency of 27%.

CPUs	Matrix Size (N)	Seconds	GFlops	% of Peak
2	8192	274.38	1.336	66.80
8	16,384	580.99	5.047	63.09
32	32,768	1192.45	19.67	61.47

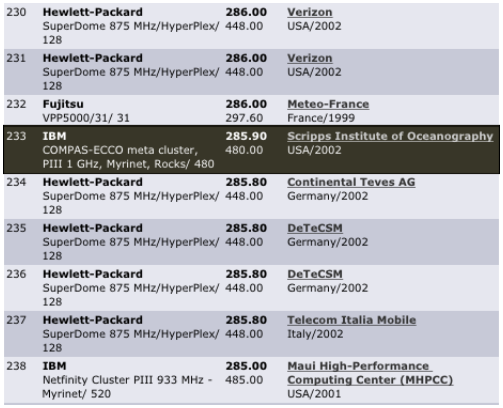
Table 6: **HPL Performance Over the Myrinet Network.** The blocking factor (NB) is 160. Scaling the problem up from 2 CPUs to 32 CPUs results in a decrease in efficiency of 7%.

Matrix Size (N)	Seconds	GFlops	% of Peak
15,000	16.90	133.1	27.73
30,000	98.21	183.3	38.19
60,000	624.02	230.8	48.08
120,000	4127.16	279.1	58.15
150,000	**7922.17**	**284.0**	**59.17**
151,000	8162.24	281.2	58.58
153,000	8385.06	284.8	59.33
155,000	8761.34	283.4	59.04

Table 7: **HPL Performance on the Metacluster.** The blocking factor (NB) is 160 and the number of CPUs is 480. While the matrix size of 153,000 outputs a slightly higher GFlop value, we selected a matrix size of 150,000 for the experiments conducted in Table 8, as time was limited and HPL completes almost 8 minutes faster with a matrix size of 150,000 vs. 153,000.

Matrix Size (N)	NB	BCAST	Seconds	GFlops	% of Peak
150,000	180	1	8160.99	275.7	57.44
150,000	180	3	8107.64	277.5	57.81
150,000	200	1	7910.33	284.4	59.25
150,000	**200**	**3**	**7870.43**	**285.9**	**59.56**

Table 8: **Experimenting with NB and BCAST Parameters on the Metacluster**. The number of CPUs for all runs is 480.



230	**Hewlett-Packard** SuperDome 875 MHz/HyperPlex/ 128	**286.00** 448.00	**Verizon** USA/2002
231	**Hewlett-Packard** SuperDome 875 MHz/HyperPlex/ 128	**286.00** 448.00	**Verizon** USA/2002
232	**Fujitsu** VPP5000/31/ 31	**286.00** 297.60	**Meteo-France** France/1999
233	**IBM** COMPAS-ECCO meta cluster, PIII 1 GHz, Myrinet, Rocks/ 480	**285.90** 480.00	**Scripps Institute of Oceanography** USA/2002
234	**Hewlett-Packard** SuperDome 875 MHz/HyperPlex/ 128	**285.80** 448.00	**Continental Teves AG** Germany/2002
235	**Hewlett-Packard** SuperDome 875 MHz/HyperPlex/ 128	**285.80** 448.00	**DeTeCSM** Germany/2002
236	**Hewlett-Packard** SuperDome 875 MHz/HyperPlex/ 128	**285.80** 448.00	**DeTeCSM** Germany/2002
237	**Hewlett-Packard** SuperDome 875 MHz/HyperPlex/ 128	**285.80** 448.00	**Telecom Italia Mobile** Italy/2002
238	**IBM** Netfinity Cluster PIII 933 MHz - Myrinet/ 520	**285.00** 485.00	**Maui High-Performance Computing Center (MHPCC)** USA/2001

Figure 3: **Final Result.** The Compas-Ecco metacluster ranks number 233 on the November 2002 TOP500 list.

achieved. This result was submitted to the TOP500 organizers and the Rocks-powered Compas-Ecco metacluster now ranks number 233 on the November 2002 TOP500 list (see Figure 3).

In the approximate 40 hours that it took to run the experiment, over 24 hours were devoted to benchmark runs. The 256-node metacluster was installed twice (once using 100 Mbit Ethernet and once using Gigabit Ethernet), and the original 128 node clusters were each installed twice for benchmarking the installation performance. The rest of this paper describes the underlying system software that allowed us to quickly reconfigure these machines into their desired configurations.

5.2 Software Installation Results

The Rocks toolkit uses installation as a simple method for users to keep cluster nodes with a consistent set of software. While administrators may want to make small changes while the cluster is "online," we find it easier and, in general, faster to do a complete re-installation of nodes for larger changes. Indeed, in our fused-cluster experiment, we found that the initial host name assigned to a machine was actually buried quite deeply into the standard configuration. More than an hour was consumed trying to find all the places where the name is registered on installation. Duplicate names (the separate clusters used the same internal node naming scheme) led to some confusion when the clusters were connected as one. In this case, it was simply faster in terms of wall clock time to reinstall all nodes instead of searching deeply into the file system to make changes. In the above example, other methods (such as cfEngine [2]) that "patch" existing installations could have been layered on top of Rocks to achieve the same result. But, we see this approach as an additional layer of complexity and management as our simple, fast installation method performs the functions of these other configuration utilities as well as providing a method for installing, updating and configuring cluster software ranging from one package to the entire software stack.

On Friday evening, it was determined that the shortest time to completion to getting the metacluster configured as a single cluster was to reinstall all the nodes. By staging each rack of machines, the complete system was rebuilt in about two and a half hours (20 minutes/rack) using a single 100 Mbit-connected frontend. Each node downloads approximately 250 MBytes of package data during its installation. Assuming that a Linux-based webserver can serve 10 MB/sec, total web traffic for 32 nodes is 32 x 25 seconds or about 13.3 minutes to serve out data for all nodes. In aggregate, package downloading from a single 100-Mbit frontend would consume about 107 minutes for the entire 256-node cluster.

One method of scaling performance is to replicate package servers and then point nodes to different HTTP-based package servers. Doubling the number of package servers would half the aggregate download time, as the bandwidth of the package server effectively doubles. A second method is simply to scale the package server from 100 Mbit to Gigabit Ethernet. After completing benchmark runs Sunday morning, installation benchmarks were run, but against a frontend connected by Gigabit Ethernet instead of 100 Mbit. Simple TCP/IP benchmarks using either `ttcp` or `iperf` show that single stream performance can routinely deliver more than 800 Mbits (100 MB/sec). A conservative estimate is that a Linux-based HTTP server can deliver 400 Mbits or 40 MB/sec. In this case, packages downloaded from a single frontend would consume about 27 minutes of total network time when serialized across 256 nodes.

5.2.1 Initial Issues at Scale

When benchmarking the metacluster installation, our first reinstallation at a gigabit consumed about 40 minutes. However, we found two key errors at this scale: 1) database connections were left open by our Kickstart generator, and 2) the DHCP client used by Red Hat was conservative regarding DHCP retries, timed out and failed to retrieve its network settings. For database access, after about 100 Kickstart generations done in quick succession, the MySQL server returned errors regarding "too many open connections". The connections would eventually timeout and Kickstart generation would complete. The Rocks-enhanced Red Hat installer has a watchdog timer so that if Kickstart generation for an installing node fails, the node reboots and tries again. When sequencing by racks (that is, 32 nodes at a time), this problem did not occur.

The second problem occurred because the DHCP client was not tolerant to dropped responses. The DHCP client first sends a *discover* message and when the DHCP server receives the message, it responds with an *offer*. The client upon receiving the offer must request the offered address, to which the server responds with an *acknowledgment*. Under heavy traffic loads, it was found that the second phase DHCP acknowledgments often were dropped. The client should re-request the address before giving up, but the standard client only performed one such request and then gave up. Both of these issues have been addressed in the latest release of Rocks (version 2.3).

5.2.2 Monitoring and Reboot Performance

Figure 4 indicates several key metrics on a single 128-node cluster as monitored and visualized by the Ganglia Toolkit [9]. Ganglia development is led my Matt Massie at UC Berkeley, and Rocks team members are collaborators and are the key authors for the web presentation of data. The toolkit and graphs are standard in a Rocks installation, as end-users have summary overviews of the cluster enabled automatically and transparently. More detailed data is available simply by clicking on individual nodes (shown at the bottom of the figure). Ganglia provides a critical capability for system monitoring.

The topmost graph shows the number of online CPUs. The first dip of 64 CPUs is the time from start to finish of a 32-node reinstallation – about 15 minutes. The second dramatic drop in online CPUs illustrates the time to reboot a 128-node cluster – a touch over 2 minutes. Finally, the upper right graph shows memory usage. Toward the right-hand side, one can clearly see HPL starting up and using varying amounts of memory as different sized matrices are benchmarked. The total time base in these plots is 1 hour.

Figure 4: **Cluster Monitoring with Ganglia.**

6 Summary and Future Work

Rocks makes it easy to rapidly build high-performance computing systems, and this paper details the truth behind that statement. Over one weekend in August of 2002, two 128-node clusters were taken out of production and physically combined into one 256-node metacluster, the software stack on every node of the metacluster was reconfigured by installing a complete operating environment from the ground up, HPL was run repeatedly, the metacluster was repartitioned into its original two 128-node clusters, all nodes were reinstalled with their original operating environment and the clusters were returned to their respective production roles. Additionally, the resulting HPL benchmark placed the metacluster in the top half (number

233) on the November 2002 TOP500 list. Not bad for a weekend.

The speed at which Rocks can deploy and manage high-performance computing resources is one of the reasons Rocks has been selected as the software environment in support of several large Information Technology Research (ITR) initiatives funded by the National Science Foundation. Three notable ITR grants that will use Rocks include: 1) OptIPuter, a $13.5 million award, 2) Geosciences Network (GEON), an $11.25 million award and, 3) UCSD's Center for Theoretical Biological Physics, a $10.5-million award.

Rocks will continue to integrate and develop software technologies in order to support high-performance clusters. In April of 2003, Rocks was released for both x86 and Itanium 2 platforms. While the underlying operating environment is different for these two platforms (Red Hat 7.3 vs. Red Hat Advanced Workstation 2.1), the cluster environment (PBS/Maui, Sun Grid Engine, MPICH, compilers, etc.) are exactly the same. Additionally, at the Supercomputing 2002 conference, demonstrations of MPI applications utilizing Intel's pre-production 10-Gigabit Ethernet were shown.

We continue to develop services and methods that enable researchers to deploy large-scale clusters. Recently, the BIO-X project at Stanford University (http://biox.stanford.edu/) has deployed a 300-node Rocks cluster. Each node is a dual Pentium 4 with hyperthreading enabled, that is, to the software services, each node appears to have 4 processors. There was one scaling issue discovered on this 1200-processor cluster – Maui was unable to track over 511 active jobs. After increasing Maui's global value, the single frontend can now manage more than 511 active jobs. No other scaling issues have been reported.

7 Acknowledgments

We'd like to thank Dr. Detlef Stammer [8] and Dr. Daniel Cayan [9] of SIO who graciously let us "borrow" their computing resources for a weekend.

Also, we'd like to thank Federico D. Sacerdoti for his insightful reviews of various drafts of this paper.

[8] Supported in part by the ONR DURIP Grant Number: N00014-01-1-0514

[9] Supported in part by the NSF MRI Grant Number: OCE0116643

References

[1] Thomas E. Anderson, David E. Culler, and David A. Patterson. A case for NOW (networks of workstations). *IEEE Micro*, 15(1):54–64, February 1995.

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

[3] Jack J. Dongarra. Performance of various computers using standard linear equations software, (linpack benchmark report). Technical Report CS-89-85, University of Tennessee, 2002.

[4] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.

[5] HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. *http://www.netlib.org/benchmark/hpl/*.

[6] M. J. Katz, P. M. Papadopoulos, and G. Bruno. Leveraging standard core technologies to programmatically build linux cluster appliances. In *Proceedings of 2002 IEEE International Conference on Cluster Computing*, Chicago, Il, October 2002.

[7] P. M. Papadopoulos, M. J. Katz, and G. Bruno. NPACI Rocks: Tools and techniques for easily deploying manageable linux clusters. In *Proceedings of 2001 IEEE International Conference on Cluster Computing*, New Port, CA, October 2001.

[8] 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*, volume I, pages 11–14, Oconomowoc, WI, 1995.

[9] The Ganglia Toolkit. *http://ganglia.sourceforge.net/*.

[10] TOP500 - The TOP500 Supercomputer Sites. *http://www.top500.org/*.

[11] R.C. Whaley, , A. Petitet, and J.J. Dongarra. Automated empirical optimizations of software and the atlas project. *Parallel Computing*, 27(1-2):3–35, January 2001.

Appendix

Below is the `HPL.dat` configuration file that generated the result we submitted for consideration in the TOP500 list.

A suggestion by the HPL developers to increase performance is to strive for a "square" processor grid. HPL multiplies `Ps` by `Qs` to determine the node count from the specified processor grid. In the example below, `Ps` $= 16$ and `Qs` $= 15$ which is 240. *Threading* in ATLAS ensures all 480 CPUs are utilized. We compiled ATLAS with threading enabled – a feature that recognizes all CPUs on an SMP node and spawns a thread for each CPU. This increases performance as messages sent between threads on the same node use shared memory to communicate rather than an external communication fabric (e.g., Ethernet or Myrinet). With threading enabled, the number of compute nodes needs to be specified for HPL, not the total number of CPUs.

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
2            # of problems sizes (N)
1000 150000    Ns
2            # of NBs
180 200          NBs
1            # of process grids (P x Q)
16 Ps
15 Qs
16.0         threshold
1            # of panel fact
1            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
8            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
2            RFACTs (0=left, 1=Crout, 2=Right)
2            # of broadcast
1 3          BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
160          swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```