

Pink: A 1024-node Single-System Image Linux Cluster*

Gregory R. Watson Matthew J. Sottile Ronald G. Minnich
Sung-Eun Choi Erik A. Hendriks

Cluster Research Lab
Advanced Computing Laboratory, CCS-1
Los Alamos National Laboratory[†]
Los Alamos, NM 87545
cluster@lanl.gov

Abstract

This paper describes our experience of designing and building Pink, a 1024-node (2048 processor) Myrinet-based single-system image Linux cluster that was installed in January 2003 at the Los Alamos National Laboratory. At the time of its installation, Pink was the largest single-system image Linux cluster in the world, and was based entirely on open-source software - from the BIOS up. Pink was the proof-of-concept prototype for Lightning, a production 1408-node (2816 processor) cluster that will begin operation at LANL later this year. Lightning is currently number 6 on the Top500 list.

In this paper we will examine the issues that were encountered and the problems that needed to be overcome in order to scale a cluster to this size. We will also present some performance numbers that demonstrate the scalability and manageability of the cluster software suite.

1. Introduction

In the past five years, Linux has gone from running on only a few small clusters to being the operating system of choice for the next-generation of high performance computing (HPC) systems at the Department of Energy (DOE) labs. The strong growth of Linux in the commercial sector,

combined with the unique requirements of the HPC environment, has convinced DOE to invest in both the development of HPC software and the development of expertise in Linux and other open source software.

As part of this effort, Los Alamos National Laboratory (LANL) has been undertaking a research project funded by the Mathematical Information and Computer Sciences (MICS) Program of the DOE Office of Science to design a fundamentally new clustering technology, known as Clustermatic. The aim of this project is to develop a complete cluster management system with the primary goal of making clusters easier to maintain and use *at virtually any scale*. The 1024-node Pink cluster is the first milestone in the realization of this research project, which is ultimately aimed at developing a 100 Teraflops Linux cluster. Pink is also the proof-of-concept for Lightning, a 1408-node production cluster that is currently number 6 on the November 2003 Top500 [6] list.

At the time of its installation, Pink was the hallmark of a number of important achievements in clustering technology for a machine of this size, including:

- The machine was the largest single-system image Linux cluster in the world.
- The vendor (Linux Networx) provided all the hardware, but almost none of the system software.
- The *entire* software suite, including the BIOS, operating system, clustering software, and other system tools were all open-source projects based at LANL and other sites.
- The system did not require a second management network, instead the Myrinet interconnect was used for both system booting and monitoring purposes, as well as a traditional HPC application interconnect.

* This research was funded by the Mathematical Information and Computer Sciences (MICS) Program of the DOE Office of Science, the Los Alamos Computer Science Institute (ASCI Institutes), and the Laboratory Directed Research and Development (LDRD) program. LANL LA-UR-04-1890.

[†] Los Alamos National Laboratory is operated by the University of California for the National Nuclear Security Administration of the United States Department of Energy under contract W-7405-ENG-36.

- The hardware installation was completed in 3 days, and the system was operational the same day. Final acceptance was less than 4 weeks later.

The remainder of the paper is organized as follows. In Section 2, we lay out the architectural design goals that were set for Pink. In Section 3, we describe the resulting hardware and software implementation that realized our goals, and present some performance numbers that demonstrate the scalability of the system. In Section 4, we detail our plans for future areas of research. Finally, in Section 5, we provide some concluding remarks.

2. Design Goals for Pink

In the last few years, cluster computing has been pushing the limits of performance that can be extracted from traditional clustering techniques. A number of research organizations have demonstrated that clusters comprising over 1000 compute nodes can be successfully built and used to achieve significant increases in application performance [3] [2] [1].

However, these performance improvements can come at a cost. Growing the number of compute nodes dramatically expands the size and complexity of the machine. From a hardware perspective, as component counts increase, so does the likelihood of a major hardware failure, as does the difficulty in locating and fixing problems. The result is that additional staff are needed to maintain and repair the machine in order keep it in an operational state. From a software standpoint, scaling issues can also have a negative impact on the operation of the cluster, both in terms of the cost of managing the system and on the performance of applications running on the machine.

Experience has shown that, unless great care is taken to mitigate these issues, the result can be a substantial increase in the cost of maintaining and operating the machine combined with a reduction in the actual application performance attained. The end result is that the price performance advantages of cluster computing can be significantly reduced.

2.1. Software goals

The primary goal is to develop a software suite that will turn commodity hardware into *purpose-built* cluster nodes. However, in order to scale to very large numbers of nodes at the same time as continuing to achieve acceptable levels of performance, a number of other factors must also be considered. These include reducing complexity, increasing reliability, and improving the maintainability of the system. The following series of software-related goals are key to achieving these outcomes.

Behave like a single system. This frees users from details of how the cluster operates, and simplifies the task of managing the system. Traditional cluster systems do this by providing scripts that attempt to hide the architecture details. However these scripts tend to be slow, cumbersome, highly sensitive to software changes, and are obviously different from commands that are familiar to users. Instead, Clustermatic employs an *asymmetric single-system image* architecture in which all processes on the system are able to be viewed and controlled from a central point.

Replace software components. Traditional approaches to clustering have predominantly involved layering software infrastructure on top of an existing operating system to create the illusion of a single coherent system. Such an approach is an very effective way of establishing a small to medium size cluster, as is reflected by the success of systems such as OSCAR [13] and NPACI Rocks [7]. However, by simply adding extra software layers to a stand-alone system, the level of complexity is increased significantly, and as the machine size increases, scaling problems in one software component can easily swamp the entire system. In contrast, Clustermatic relies on *eliminating* software that is not required, and *replacing or enhancing* existing software with that needed to achieve the level of scalability and performance required for very large clusters.

Avoid legacy software. Legacy software can be a significant factor in reducing the reliability of clusters, so ensuring this software functions correctly is an important goal. One component of commodity systems that is often overlooked, but that has a significant impact on the manageability of large clusters, is the PC BIOS. Our experience is that the standard BIOS often mis-configures hardware, and can be unreliable in operation. In addition, the BIOS has been designed to *require user intervention*, and although capable of unattended booting, there are many situations where a keyboard and monitor will need to be connected to the node. This is just not practical when there are thousands of nodes.

Use only open-source system software. The entire system software suite, including software provided by hardware vendors (*e.g.*, firmware, device drivers, etc.) should be open-source. Open-source software allows administrators to more quickly find and fix problems, and removes reliance from vendors. In our experience the quality and reliability of open-source software can also be equivalent to, or even better than, that supplied by vendors.

2.2. Hardware goals

While the physical properties of a 1024-node cluster are daunting, installing, managing, and maintaining a system of this scale is even more daunting. Therefore our primary

goal is to *reduce the parts list*, which in turn will lead to increased reliability and make it easier to diagnose and fix problems.

Minimize the number of wires per node. Most cluster systems that employ a high performance network such as Myrinet or Quadrics Elan usually include an Ethernet or serial network (or sometimes both) for system management and monitoring, and sometimes also require a network dedicated for I/O purposes. Each additional network requires more space both under the floor and in the racks, which can contribute to reduced air flow. More connectors and cables means less reliability, and makes the task of locating, replacing, and re-routing a faulty cable all the more difficult. Ideally there should only be two wires running from each node, one for power and the other for the interconnect network.

No moving parts on the node. Other than the power supply and CPU fans, no other moving parts (*e.g.*, disk, floppy, CD-ROM) should be required for a cluster node to operate. Moving parts are much more prone to failure than solid state devices, so eliminating reliance on them means the nodes are more resilient. This does not preclude the use of a disk or other device for local storage once the node is operational.

Reduce reliance on legacy components. In addition to moving parts, other components such as Ethernet devices, UARTs, keyboard and mouse controllers, and video hardware should not be required for node operation. Again, reducing reliance on these parts increases the resilience of nodes to component failure.

3. Pink

The purchase of Pink was atypical of most large system acquisitions because the vendor was required to supply only the node hardware and network interconnect portion of the system. Key features of the system include:

- 1024 power cables
- 1024 Myrinet network cards
- 2048 fiber cables (approximately 15 miles)
- 3072 Myrinet switch ports
- 4096 SDRAM memory modules
- 5120 fans
- 1 hard disk
- 1 CDROM drive

All the system software is provided by the Clustermatic software suite. The budget for the purchase of system hardware was \$6M and the successful vendor, Linux Networkx (LNXI), supplied and installed the entire machine for this amount.



Figure 1. Pink - 1024 Nodes

3.1. Node Hardware

The Pink node hardware is comprised of 1024 dual 2.4 GHz Intel Xeon processor nodes, making it theoretically a 9.6 Teraflops machine, with a nominal price/performance of \$0.625 per Megaflop. Each node has 2GB of memory, a 16MB IDE-FLASH and LinuxBIOS pre-installed in system FLASH by the vendor. Mainboards are supplied by Super Micro Computer, Inc., and use the Intel E7500 chipset.

Physically, the machine occupies three rows of nine 19-inch racks, where each rack is 50U tall (1U is approximately 1.75 inches), for a total of twenty seven racks. The racks contain five 8U chassis, and each chassis can house ten 0.8U nodes, five 1.6U nodes or two 4U nodes. All three different sizes of nodes are mounted vertically in a chassis. The special enclosure and chassis, designed by LNXI, leads to a much higher density than previously achievable using commodity mainboards (typically no smaller than 1U), and nicely combines commodity hardware with cluster-specific technology. Figure 1 shows the layout of Pink at its current location.

The 1024 nodes of Pink consist of a mix of the three sizes of nodes. The bulk of the nodes are exclusively for use as compute resources, and are the small form-factor 0.8U type. For each group of 16 compute nodes, there is one of the larger 1.6U form-factor nodes. These nodes are intended for use as an I/O testbed. The breakdown of node types is as follows:

- 959 0.8U nodes
- 64 1.6U nodes (for an I/O testbed)
- one 4U node for the front-end

The node hardware has proved remarkably reliable. Table 1 shows the node failures that occurred in the first 12 months of operation.

Number	Cause of Failure
1	hard disk
10	cooling fan
2	IDE flash
5	unknown

Table 1. Node Hardware Failures

There has been a total of only 18 hardware failures (excluding network issues) that were serious enough to prevented the nodes from operating. Note that the only disk in the system (used by the front-end) had failed on delivery. This represents a node failure rate of 0.15% per month.

3.2. Network Hardware

The Pink network is Myrinet 2000, the 3rd generation network technology from Myricom, Inc. Myrinet supports a maximum of 2Gb/s on each link. According to benchmark results [12] this provides a sustained one-way data rate for large messages of 243Mb/s, and a short message latency of 7.6 us. The network hardware comprises:

- 1024 Myrinet 2000 M3F-PCI64C network cards (one in each node)
- 16 M3-E128 switch enclosures containing 8 M3-SW16-8F line blades and 8 M3-SPINE-8F spine blades
- 8 M3-E128 switch enclosures containing 16 M3-SPINE-8F spine blades

The switch enclosures provide slots for up to 16 blades and contain an active backplane comprising 8 16-port Myrinet crossbar (XBar16) switches arranged as a Clos spreader network. Each line blade contains one XBar16, which provides 8 ports for connecting to nodes and 8 ports connecting to the enclosure backplane switches. Each spine blade provides 8 ports for connecting to spine blades in another switch. The switch enclosures also contains an M3-M maintenance blade that provides monitoring and maintenance capability.

Figure 2 shows the arrangement of switches and nodes in the Pink network (not all switches and network connections are included for clarity). Each node connects to a port on a line blade in one of the 16 Clos switch enclosures (8 ports x 8 blades x 16 switches = 1024 nodes). The remaining spine blades are used to connect the 16 Clos switch enclosures to the spine switch enclosures. This configuration provides a full bisection network with a diameter of five switches. A typical route, marked in black, shows that a packet must traverse five switches in order to reach its destination. Such a network provides excellent scaling properties as well as significant path redundancy, full bisection and low latency [11].

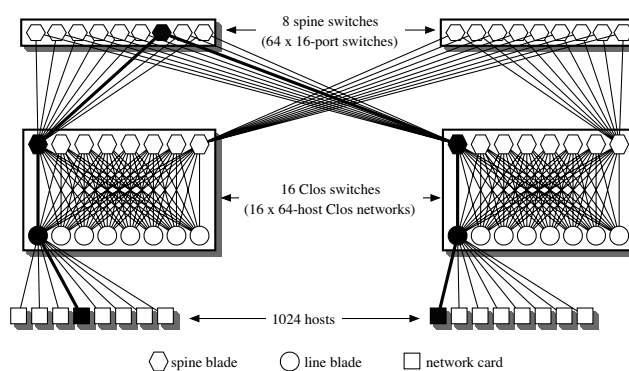


Figure 2. Pink Network Configuration

The Myrinet network topology is a full-bisection bandwidth Clos [8] network providing low latency as well as significant path redundancy for reliable point-to-point links between nodes. Myrinet 2000 is a fiber-based technology making the physical cabling issues, such as space for cables and cable weight, much more reasonable for a cluster of this size. Recall that there are 2048 fiber cables in Pink, making the weight and reliability of each cable significant. All network traffic is *source-routed*, resulting in very simple, and in our experience, much more reliable switches.

In contrast to the node hardware, the network hardware proved particularly unreliable. Table 2 shows the network hardware failures that occurred in the first 9 months of operation.

Number	Cause of Failure
4	network card
15	spine blade
7	line blade

Table 2. Network Hardware Failures

This high failure rate (especially of the spine and line blades) prompted an investigation by the vendor, and it was subsequently determined that the failures were likely caused by a manufacturing defect. As a result, all spine and line blades have now been replaced. Also at around the same time, the network cards were replaced with the new generation Lanai XP cards.

3.3. Booting Pink

The nodes in Pink have no disks, so they must network boot in order to obtain an operating system, and then load the complete set of software that is needed for node oper-

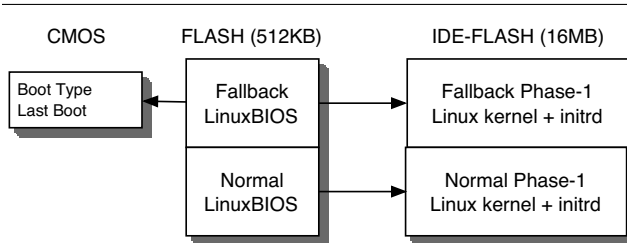


Figure 3. FLASH layout

ation. The boot process for the cluster as a whole involves the following steps:

- the front-end boots Linux normally from an attached disk
- the 1023 nodes simultaneously perform a network boot
- once booted, each node loads the appropriate application libraries and other software across the network
- the nodes wait for processes to be dispatched from the front-end control node

3.3.1. Initial Power-On LinuxBIOS is used for the initial hardware/firmware configuration that is performed when a node is first powered on. LinuxBIOS is an open-source, multi-platform, multi-architecture BIOS that replaces the standard BIOS used on most PCs [10]. It is a small hardware startup program which turns on the CPU, memory, and I/O hardware and then loads a program, called the *payload*, into memory. This payload can be anything that is required to suit the particular application, such as a boot loader or even a small Linux kernel. The main limitation being the size of the FLASH available to store the payload. On Pink, which has a relatively small 512 KB system FLASH, the payload is a boot loader called Etherboot, which loads a Linux kernel and initial RAM disk image from a much larger secondary FLASH device that is attached to the IDE bus.

As there are no disks, floppies, or CDROMs that could be used to recover if the boot sequence fails, LinuxBIOS has to be at least as, if not more, reliable than a standard BIOS for booting the machine. In addition, updating the FLASH on 1024 nodes is risky business, as having to replace the FLASH chips on all nodes would take several weeks. So Pink must be able to survive a FLASH upgrade that installs a damaged LinuxBIOS or kernel image, while still booting to full capability with no manual intervention.

For its LinuxBIOS-based Evolicity series of systems, LNXI developed a FLASH layout and recovery technique which has been extended to take the IDE-FLASH into account. The components of the boot system are shown in Figure 3.

The three components are the CMOS battery-backed memory, the system FLASH, and the IDE-FLASH. The

CMOS holds parameters that determine which image to use at startup. The system FLASH holds two copies of LinuxBIOS and Etherboot: the *fallback* and *normal* copies. The IDE-FLASH also holds a fallback and normal copy of the kernel and initial RAM disk image combination. Using a combination of checksums and the boot parameters in CMOS, it is possible to determine precisely which image is correct for booting the system. This provides such a robust fallback boot strategy that it can survive almost any kind of failed FLASH update. The strategy is so robust that the FLASH on all 1024 nodes can be updated at once, and if the reflash fails, the cluster will still boot.

3.3.2. Two-Phase Boot The first phase of the boot process occurs when LinuxBIOS loads a Linux kernel from the IDE-FLASH. This kernel, known as the *phase-1 image*, has just enough hardware support in order to load and configure the Myrinet network hardware on the node. The phase-1 image then uses RARP to configure the network interface and download an operational kernel and initial RAM disk, known as the *phase-2 image*, from the front-end. This download uses an efficient multi-cast protocol in order to distribute a boot image to large numbers of nodes. This is essential if the system is going to be able to support 1023 nodes booting simultaneously.

The new kernel that is obtained from the front-end, is executed using a technique known as the *two kernel monte*. Two kernel monte loads the new Linux kernel into memory, overwriting the one that is currently running. This allows the phase-1 image to load the new kernel without writing it to some non-volatile storage or needing to reset the hardware.

Once loaded, the operational kernel repeats the RARP step that was performed by the phase-1 kernel. After the network is configured, a special control daemon that is located in the RAM disk image begins execution on the node. At this point, the node is running a kernel and the control daemon, but has no other software loaded. The control daemon notifies the front-end that it is ready to complete the rest of the node setup process. The following actions are then performed to complete the node setup:

- *Mounting file systems.* This includes `procfs` as well as other local and network filesystems.
- *Copying files and creating device nodes.* The nodes require a small number of system files to operated correctly.
- *Copy shared libraries.* A subset of the shared libraries on the front-end are copied over at boot time so that they don't have to be copied every time a process migrates to a node.
- *Copy application specific information.* Some applications require access to libraries or other shared information that is copied onto the node at this time.

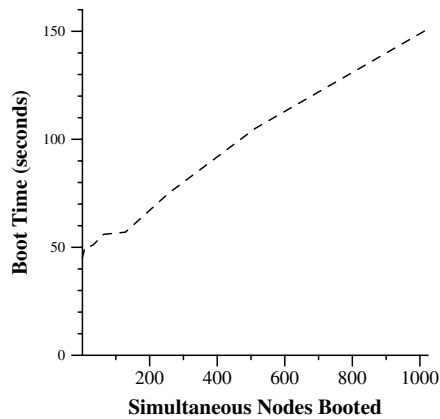


Figure 4. Pink Boot Time

3.3.3. Boot Characteristics Figure 4 shows the time required for Pink nodes to boot. This is the time from when power is first switched on to when the system is ready to accept jobs. As can be seen, the boot time of the machine scales well, ensuring the whole machine is able to boot in under three minutes.

Using Linux as a boot strap also offers a number of advantages over using traditional boot ROMs or PXE to boot the nodes. The most significant is that the boot process is able to operate with any network hardware that Linux supports. This is essential for Pink, which only has Myrinet available, and there are no boot ROMs available which support this type of network.

With no attached disk, Pink nodes are unable to retain any state across boot cycles. This leads to some useful properties for the system. If there is no software permanently installed on the nodes, there is nothing which can be misconfigured and prevent the system from operating. A normal Linux machine requires a large amount of software to be installed and configured correctly before it is able to communicate with other machines. With no local software, there is no possibility of version skew in such software causing problems. On Pink, a new node install, a reboot, and a software upgrade are all identical operations.

Finally, although it would be possible to place the operational kernel in FLASH, this isn't done for two reasons. Some systems don't have a large enough FLASH to hold both the BIOS and a full Linux kernel simultaneously. Also, if the operational kernel were stored in the system FLASH, kernel updates would become a much more complex process. Instead, using the two-phase system, only a single kernel image needs to be updated on the front-end system.

3.4. Network Configuration

Pink's network is a 1024-port full bisection Myrinet network [11]. Myrinet was chosen for Pink due to its low

latency, good bandwidth and ability to scale to 1024 nodes. Myrinet is a source routed network with very simple switches.

In a source-routed network every node has a list of routes to every other node. These routes are normally created with a program (provided by Myricom) called the *mapper*. On a network this large, the mapper requires a significant amount of time to explore the network, calculate routes and load them into every node. For Pink, the time required to generate a set of routes for all 1024 nodes is about 4 minutes.

Pink's nodes need to have their routing tables loaded twice during boot up, once for the phase-1 kernel to locate and load the phase-2 image, and again when the phase 2 kernel has booted. This is because Pink nodes do not use local storage to retain routing information across the two-phase boot sequence. The need to update the routing tables twice makes route table computation an important factor in the boot times that can be achieved on Pink.

There are number of possible methods to reduce the impact of this problem, including:

- maintain the routing table in flash
- improve route computation time of the Myrinet mapper
- improve route loading mechanism
- use NVRAM to store route to front-end node

While maintaining a routing table in flash memory is possible, it is not considered practical due to the difficulty in updating the flash image and the finite number of write cycles inherent in flash memory. The solution used on Pink is to store a single route to the front-end node in NVRAM. This provides enough information for a node to load the phase-2 image, and then obtain a complete set of routes from the front-end. The result is a very significant improvement in node boot times, and the time to boot the entire machine.

3.5. System Operation

Once booted, Pink employs BProc to provide single-system image and process migration facilities for the cluster. BProc, which has been described in detail elsewhere [9], is a set of kernel modifications and utility programs that enable the cluster to be operated, and used, as if it is a single machine. All interaction with the cluster is controlled through the front-end node. Processes to be executed on the cluster nodes are first created on the front-end, then migrated to the nodes on which they will run. At the same time, the process space of the entire machine is visible from the front-end. This allows the normal UNIX commands, such as `ps` and `kill`, to be used to manage and control processes that are running anywhere on the cluster.

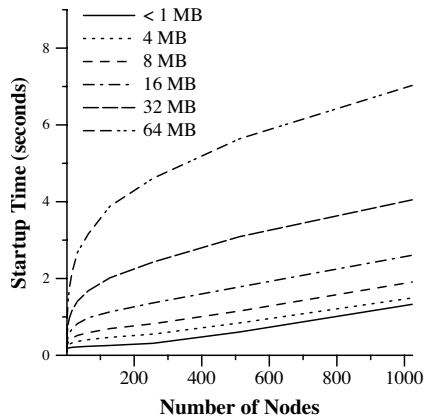


Figure 5. MPI Startup Time For Different Executable Sizes

Apart from the obvious advantage of making the cluster easier to use and manage, the process migration feature provides a fast and scalable mechanism for starting jobs on the nodes. It does this by employing an ad-hoc tree spawn technique for efficiently migrating processes to a large number of nodes. The result is a reduction in the job startup overhead, which ultimately reduces overall job run times. Figure 5 shows startup times for various sizes of MPI job. The graph clearly shows that the startup time scales well for the range of executable sizes and number of nodes likely to be encountered on Pink.

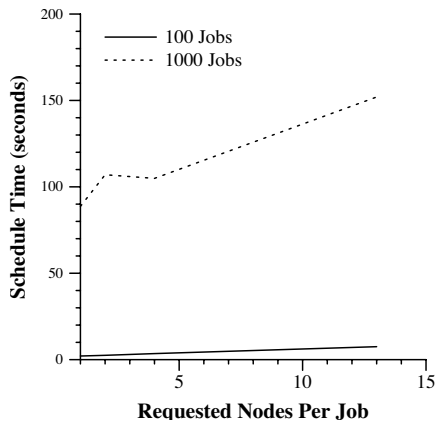


Figure 6. BJS Scheduling Overhead

Clustermatic also provides a light-weight batch scheduler known as the BProc Job Scheduler (BJS). BJS utilizes the capabilities provided by BProc to control access to cluster nodes, and extends this to support typical scheduling activities, such as job queueing, resource allocation, back filling, etc. By default, BJS only provides basic scheduling ser-

vices, but it is designed to be extensible through the creation of policy modules which can define arbitrary rules for allocating jobs to nodes. Since BJS exploits a range of existing BProc services, it is extremely light-weight, fast, and scalable. Unlike traditional job schedulers which require substantial infrastructure on each node (daemons, control files, etc.), BJS runs entirely on the control node, resulting in very simple deployment and minimal interference with executing jobs.

Figure 6 shows the overhead imposed by the job scheduler for different allocation requests. The overhead is measured from when the job is submitted until it is completed, less the actual execution time of the job. The lower line shows the overhead when 100 jobs are submitted simultaneously, while the upper line is for 1000 jobs submitted simultaneously. As can be seen, the overhead required when scheduling 100 jobs is virtually independent of the number of nodes requested per job. This becomes more of an issue when scheduling 1000 jobs, but still requires less than two minutes to complete the scheduling activity.

4. Future Work

Pink is a systems research platform for a 100 Teraflops machine. The purpose of reaching the current milestone is to identify those areas of the Clustermatic architecture that are likely to cause problems when scaling to machines of this size. As hoped, the installation of Pink has highlighted a number of areas that require additional research if we are to ultimately achieve our goal.

Really only two wires. One goal that we did not meet with Pink was that of having only two wires per node. In addition to power and network, the nodes in each chassis are connected to a Linux Network ICE Box that provides cluster management functions, such as power control, temperature monitoring and serial console access. The additional features proved valuable during the initial bring-up and troubleshooting of the machine, but we would like to eventually remove the ICE Boxes because the added complexity is already causing problems. In particular, the ICE Boxes have proved unreliable after a power failure, and often fail to restart. This in turn causes some nodes to fail to boot automatically, requiring user intervention for the machine to become fully operational.

Multiple front-end nodes. Though a single front-end node performs very well for 1024 nodes, it is still a single point of failure. If the front-end node fails for any reason, then the compute nodes will eventually reboot even if they are currently running jobs. While the Clustermatic system already permits nodes to accept processes from more than one front-end, we plan to make multiple front-ends share the entire process space. This will ensure that when one front-end

fails, another will be able take over jobs that were submitted from the failed node.

Improvements in route loading. The current route loading scheme requires that the mapper be re-run when a node is replaced, or when the network topology changes. On Pink, the failure rate of nodes is very low (less than two per month), but on larger systems the mapper run times will eventually become unacceptably long. We also hope to alter the route loading scheme to be location specific, rather than physical device (*i.e.*, MAC address) specific, in order to simplify the replacement of nodes.

Parallel file systems. The provision of a high-performance distributed file system is essential for machines the size of Pink. There are a number of projects currently underway that aim to address the needs of distributed file system support for cluster environments. The two most promising technologies are the Panasas ActiveScale File System [5] and the Lustre Cluster Filesystem [4], however it is too early in the development process to determine which, if any, will meet the needs of very large clusters. We are currently installing an 80-node, 40 Terabyte Panasas system for testing with Pink. We will also be equipping the 64 I/O nodes with multi-terabyte RAID arrays to test as Lustre servers, with the other 959 compute nodes running as clients. In both cases the Myrinet network will be used as the file server network.

5. Conclusion

The motivation for designing and building Pink, a 1024-node cluster, is to demonstrate that the Clustermatic architecture is a suitable platform for research into developing Terascale clusters. The basic premise of Clustermatic is that by removing the dependency on unneeded hardware, replacing rather than adding software, and avoiding legacy software, it is possible to achieve the level of reliability required for systems of this scale.

The result of this strategy is that Pink nodes are more reliable than expected. In twelve months, we have seen only a small number of complete node failures. In contrast, we have seen a significant number of network component failures, including Myrinet adapter cards and switch blades. The majority of failures in the network infrastructure has highlighted the need for better mechanisms for dealing with network related problems.

The software provided by the Clustermatic suite has made Pink significantly easier to manage and use than smaller clusters employing traditional clustering systems. Both firmware and operating systems updates are performed reliably and quickly over the network. We believe that such an increase in reliability and decrease in management effort will be the only reasonable way to build and operate Terascale clusters in the future.

References

- [1] <http://computing.vt.edu/>.
- [2] <http://icnn1.lanl.gov/ldswg/icnn/content/qadev/>.
- [3] <http://www.llnl.gov/linux/mcr/>.
- [4] <http://www.lustre.org/>.
- [5] <http://www.panasas.com/>.
- [6] <http://www.top500.org/list/2003/11/>.
- [7] G. Bruno and P. M. Papadopoulos. NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters. In *Proceedings of Cluster 2001*, Anaheim, CA, October 2001.
- [8] C. Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, March 1953.
- [9] E. A. Hendriks. Bproc: The Beowulf distributed process space. *16th Annual ACM International Conference on Supercomputing*, June 2002.
- [10] R. Minnich, J. Hendricks, and D. Webster. The Linux BIOS. In *Proceedings of the Fourth Annual Linux Showcase and Conference*, Atlanta, GA, October 2000.
- [11] Myricom, Inc. Guide to Myrinet-2000 switches and switch networks. August 2001.
- [12] Myricom, Inc. GM 1.6.3. API performance with PCI64B and PCI64C Myrinet/PCI interfaces. October 2002.
- [13] The Open Cluster Group. OSCAR: A Packaged Cluster Software Stack for High Performance Computing. January 2001.