Life with Ed: A Case Study of a LinuxBIOS/BProc Cluster[*]

Sung-Eun Choi Erik A. Hendriks Ronald G. Minnich Matthew J. Sottile
Advanced Computing Laboratory
Los Alamos National Laboratory[†]
Los Alamos, NM 87545
{sungeun,hendriks,rminnich,matt}@lanl.gov

Aaron J. Marks
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
ajmarks@dsl.cis.upenn.edu

Abstract

In this paper, we describe experiences with our 127-node/161-processor Alpha cluster testbed, Ed. *Ed is unique for two distinct reasons. First, we have replaced the standard BIOS on the cluster nodes with the LinuxBIOS which loads Linux directly from non-volatile memory (Flash RAM). Second, the operating system provides a single-system image of the entire cluster, much like a traditional supercomputer. We will discuss the advantages of such a cluster, including time to boot (101 seconds for 100 nodes), upgrade (same as time to boot), and start processes (2.4 seconds for 15,000 processes). Additionally, we have discovered that certain predictions about the nature of terascale clusters, such as the need for hierarchical structure, are false. Finally, we argue that to achieve true scalability, terascale clusters must be built in the way of Ed.*

1. Introduction

Beowulf-style cluster computers gained enormous popularity because inherent in their design is the capability to leverage both hardware and software from the commodity computing market. Racks full of home PCs running Linux or FreeBSD (or even Windows) are a familiar sight in computer labs these days. But also inherent in their design is *excess*. The cluster nodes have all the hardware and capabilities of being independent desktop workstations, when all they need is a power supply, CPU, network interface, and the ability to spawn processes. Even specially designed cluster nodes available from major hardware vendors ship with either a floppy disk or CD-ROM drive (and sometimes even keyboard, video, and mouse capabilities), the primary purpose of which is installing and upgrading system software.

Excess limits the ultimate scalability of a cluster in terms of system administration as well as end user performance. For example, upgrading cluster software is equivalent to upgrading software on each of the individual nodes. Starting a job to run on P nodes of a cluster is equivalent to remotely logging in to each of the P nodes and starting the job. Until recently, these aspects of clustering were considered unimportant. Downtime for upgrades is to be expected; job startup is not considered part of program execution time. While this may be acceptable if downtime is on the order of an hour a week with startup on the order of seconds, the reality is that these overheads are a function of the cluster size and grow rapidly as the number of nodes in the cluster increases. Moreover, duplicating work introduces points of failure. If application software on one of the nodes is accidentally misconfigured (or configured differently from the rest), unexpected application behavior arises; if any one of

[*]This research was funded by a grant from the Department of Energy's Office of Science and the Los Alamos Computer Science Institute. LANL LA-UR-02-1231.

[†]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.

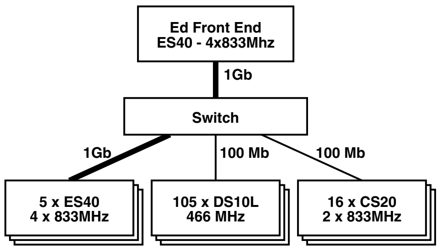


Figure 1. Ed, a Science Appliance testbed. Ed's front-end is a four processor Compaq ES40. The computer nodes are comprised of five ES40s, 105 single processor Compaq DS10s, and 16 API dual processor CS20s. The ES40s are connected to the front-end via Gigabit Ethernet, and the remainder of the nodes are connected via 100 Mb Ethernet.

the P remote log ins fails, the entire job startup fails. Current solutions to these problems all involve additional software layers that create additional points of failure [13, 2, 14].

The Cluster Research Lab at Los Alamos National Laboratory (LANL) was formed in an effort to attack these and other foreseeable problems with the next generation of cluster systems, which will be comprised of thousands of nodes and hundreds of thousands of processors. We have designed and built a prototype for a terascale-and-beyond cluster called a *Science Appliance*. The Science Appliance aims to enable a 1000-fold increase in the computational power of clusters with a 10-fold decrease in the overhead of managing and using the cluster.

In this paper, we describe Ed, the second Science Appliance testbed. Ed is a 127-node/161-processor Alpha cluster (Figure 1). Ed's front-end is a four processor Compaq ES40. The compute nodes are comprised of five ES40s, 105 (single processor) Compaq DS10s, and 16 API dual processor CS20s. The ES40s are connect to the front-end via Gigabit Ethernet, and the remainder of the nodes are connected via 100 Mb Ethernet. The compute nodes also have a single Quadrics Elan 3 interface, though none of the performance numbers reported in this paper used the Quadrics interconnect[1]. This mixed collection of machines and networks gave us a few different testbeds for our software all within the same cluster.

[1]We were unable to use the Quadrics interconnect due to bugs in the cards and vendor supplied drivers. By the printing of this paper, the Quadrics cards will have been replaced with Myrinet cards which will hopefully provide a more reliable high speed interconnect.

Ed's compute nodes boot diskless from non-volatile memory (Flash RAM), and the operating system provides a single-system image of the entire cluster, much like a traditional supercomputer. Some of the advantages of such a cluster include time to boot (101 seconds for 100 nodes), upgrade (same as time to boot), and start processes (2.4 seconds for 15,000 processes). From our experiments, we have also discovered that certain predictions about the nature of terascale clusters, such as the need for hierarchical structure, are false.

The remainder of the paper is organized as follows. In Section 2, we describe our prototype Science Appliance, Ed, and the technology that distinguishes it from existing clusters. In Section 3, we give preliminary performance numbers and discussion, including surprising discoveries. In Section 4, we finish with conclusions and a discussion of future work.

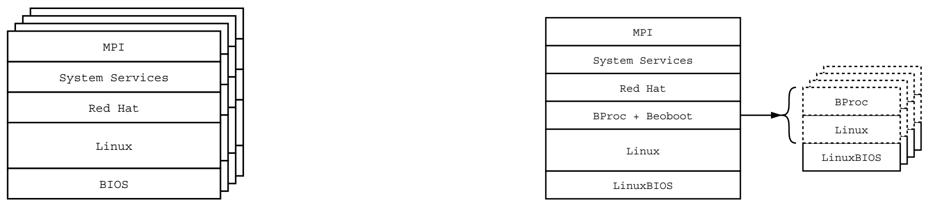
2. Building a Science Appliance

The Science Appliance is unique in many ways. It is so different, in fact, that most of the problems of managing more traditional clusters simply do not exist. The key idea is to reduce the amount of software needed to run the cluster. We achieve this by leveraging the "right" set of (open source) workstation software and replacing vast amounts of redundant software with simpler, cluster-specific software. To the maximum extent possible, we have removed scripts, configuration tools, node daemons, and the hundreds of files associated with them.

2.1. A new cluster architecture

The Science Appliance is a complete restructuring of the cluster architecture starting with the standard Flash RAM based BIOS responsible for individual node bring-up. Figure 2 contrasts the Science Appliance to the traditional approach to building clusters. The traditional cluster (*left*) is built by replicating a complete workstation installation on every node. The Science Appliance (*right*) runs an enhanced single-system image version of Linux called BProc [3]. The front-end node looks very similar to a traditional cluster node. The compute nodes run a similar kernel, but do not run any system services and in fact do not even have direct access to program binaries. All program binaries exist only on the front-end. They are started on the front-end and moved to the nodes via a technique called *directed migration* (described below).

We have replaced the normal BIOS with our own BIOS, the LinuxBIOS [8]. The LinuxBIOS boots Linux



Traditional cluster architecture   Science Appliance architecture

Figure 2. The traditional cluster architecture versus the Science Appliance. The traditional cluster (*left*) is built by replicating a complete workstation environment on every node. The Science Appliance (*right*) runs an enhanced single-system image version of Linux called BProc. The front-end node looks very similar to a traditional cluster node; the compute nodes run a similar kernel, but do not run any system services and in fact do not even have direct access to program binaries.

from the Flash memory, and starts Linux with an initial RAM disk, also loaded from Flash memory. Because we control the BIOS, we are able to control the nodes from the first instruction after reset, so the nodes function as purpose-built cluster nodes from power-on. Moreover, the nodes are more reliable as they only require a working power supply and CPU to boot. With a working network interface, they are also network manageable from power-on.

In most cases, the Flash kernel is not used as the operational kernel, though it can be. The kernel loaded from Flash is typically used to contact a front-end node and prepare the machine to run the operational kernel, which it receives from the front-end. In other words, Linux is being used as a network bootstrap to the front-end node running BProc.

This two-phased boot process, called *two-kernel monte*, is illustrated in Figure 3. Each node begins by requesting an IP address, boot image (kernel) filename, and port numbers (steps 1 and 2). The nodes then request the operational kernel, which is distributed using IP multicast (steps 3 and 4). Note that a single multicast message (dashed line) could be used by all the nodes. Since the nodes cannot be expected to simultaneously reach this step, they begin receiving multicast packets immediately, *even if the transmission of the message has already begun*. Nodes that begin receiving part-way through the boot image continue into the next multicast message until they have a complete image. Though this implies that more than one multicast message may be sent, in practice the number of copies transmitted is far less than the number of nodes. Finally, the nodes boot the operational kernel and again request an IP address (steps 5 and 6), and establish a TCP connection to the front-end (not shown).

Notice that there is an explicit handshake between each node and the front-end between steps 1 and 2 and again between steps 5 and 6. It was our anticipation that these handshakes would require us to add hierarchy to the management structure (*i.e.*, multiple front-end nodes) for clusters larger than 100 nodes. In fact, hierarchical cluster structure has been the rule for many existing production-grade cluster systems [11, 1]. In the next section, we show why we no longer believe that hierarchical structure is necessary for cluster with hundreds of nodes.

Here we see an immediate advantage of using Linux as the network bootstrap mechanism: we can exploit a protocol (IP Multicast) that reduces the network overhead for booting from $O(\#\ nodes)$ to $O(1)$. Contrast this with the widely used approach of using PXE [4] for network boot. PXE is very limited as to which interfaces (a limited number of Ethernet interfaces), protocols (one), and authentication mechanisms (none) it supports, where as the Science Appliance can support any thing Linux can support because Linux is our bootstrap mechanism. The PXE boot protocol is both naive and insecure, whereas the Science Appliance protocol can be as secure as the most secure network protocols Linux can support.

The root file system is maintained in a RAM disk. Nodes actually come up with an empty file system. The few files required for normal operation (a few entries in /etc, /dev) are created or downloaded from the front end at boot time. Dynamically linked libraries and the dynamic linker itself are downloaded on demand from the front end via multicast. Consequently, there are no issues with version skew of binaries or libraries.

Contrast this with the two most popular approaches of maintaining the root file system: local disk or NFS. The local disk is very susceptible to version skew because each local disk must be individually installed and

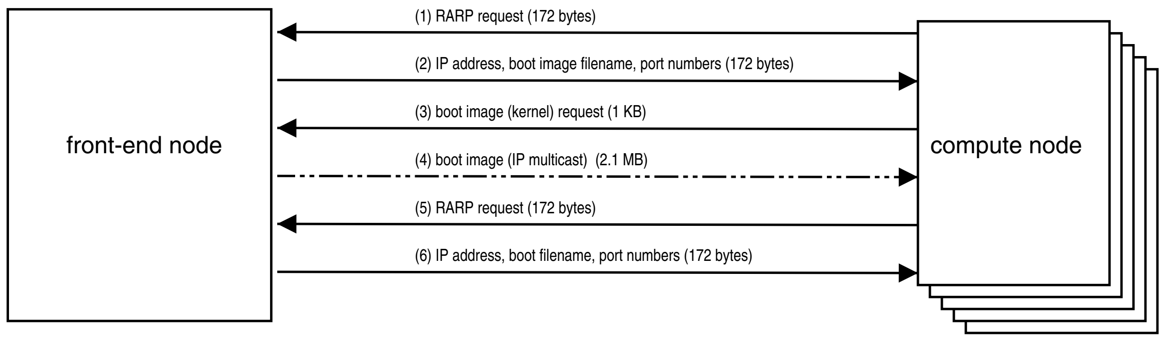

front-end node
compute node
(1) RARP request (172 bytes)
(2) IP address, boot image filename, port numbers (172 bytes)
(3) boot image (kernel) request (1 KB)
(4) boot image (IP multicast) (2.1 MB)
(5) RARP request (172 bytes)
(6) IP address, boot filename, port numbers (172 bytes)

Figure 3. Cluster node boot process. Each node begins by requesting an IP address, boot image (kernel) filename, and port numbers (steps 1 and 2). The nodes then request the operational kernel, which is distributed using IP multicast (steps 3 and 4). Note that a single multicast message (dashed line) could be used by all the nodes. Finally, the nodes boot the operational kernel and again request and IP address, boot image (kernel) filename, and port numbers (steps 5 and 6).

configured in exactly the same manner. Keeping all the local disks synchronized requires a complex tool chain and continuous *manual* management of configuration files. In a Science Appliance, if there is a working local disk, it can be used for data files, but it is not necessary for node operation. We have found that one of the most common failures on our clusters is the local disk. Since a Science Appliance node will boot even if the local disk fails, it can notify the front-end of the failure and proper action can be taken.

Though they eliminate the version skew problem, it is fairly common knowledge that NFS-based root file systems are not a scalable technology beyond 100 nodes. For large scale clusters, this scaling limitation requires that the cluster be arranged in sub-clusters of 32 or 64 nodes, with each domain served by a local NFS server. In the case of a 1024-node Tru64 Alpha cluster maintained by another group at LANL, there are 32 separate NFS domains, each requiring a primary and backup server. Sandia National Laboratories demonstrated that scaling NFS for CPlant [11], one of the largest production clusters in existence, required the creation of whole new NFS subsystems.

2.2. Other technologies

On top of the basic architecture for the Science Appliance we have added other technologies that contribute to a final cluster environment. Here we briefly describe two: Supermon, a cluster monitoring tool, and edb, a scalable parallel debugger.

Supermon Supermon is a high speed cluster monitoring tool, able to achieve sampling rates never seen before [9, 12]. The dramatically lower overheard coupled with the higher precision of Supermon as compared to the standard Linux monitoring tools such as rstatd allows us to see behavior that was previously invisible. Supermon consists of two data collection programs, a loadable kernel module, and a library for managing the data. The first program, "mon," is a data server that runs on each node and does primary data collection. Mon can accept many connections to remote data gathering programs. Mon also supports dynamic filters so that only the data needed by remote programs need move over the network. The second program, "supermon," is a data concentrator that can collate the data from many mons so that user programs do not need to manage this task. Furthermore, since the protocol used by mon and supermon is compatible, supermon programs can be composed into a hierarchy to optimize communications based on the cluster interconnect topology (though we have not seen the need for this on Ed). The kernel module provides additional entries in `/proc` which are read by mon. Finally, the libraries support a data manipulation API so that users can easily develop programs for analysing data. We have data filters for Perl, Java, MatLab, and other programs. Some programs provide a GUI-style interface for interactive real-time monitoring, and other programs are used for off-line data analysis of supermon tracefiles.

Edb Edb is an interactive parallel debugger for *at scale* application debugging [5]. Edb is currently implemented using the UNIX `ptrace`() system call. On most clusters using `ptrace` would limit edb to running on a single node. Since BProc transparently supports ptrace operations over the network to remote nodes, edb can control processes on all the nodes in Ed. It uses the bfd library to acquire the target symbol table, the proc filesystem to query the status of running processes and the readline library for an enhanced, Emacs-like command-line interface (which includes command history and completion). Debugging commands exist for loading and running a parallel application, process control (*e.g.*, wait, continue, signals), examining and modifying state and process selection. All debugging commands are executed on the current process group (called a process focus). Users can create, name, and use as many process foci as needed. By default, the process focus is all processes, but this can be modified at any time during the debugging session.

2.3. Summary

The Science Appliance effectively reuses commodity desktop hardware and software. The design eliminates vestigial elements of PC clusters such as the BIOS and local or NFS-mounted root, while leveraging existing software such as the Linux operating system. Removing dependence on the BIOS and local or NFS-mounted root makes for enormous improvements in reliability and greatly reduces the management effort, as well as eliminating all the scripts, tools, and documentation needed for keeping nodes in sync. Leveraging Linux enables the use of the most appropriate and best available network drivers, protocols, as well as programming tools.

3. Performance Evaluation

In this section, we present a performance evaluation of our Science Appliance, Ed, with respect to installation and upgrade time, job startup time, Supermon, and edb.

3.1. Installation and upgrade (boot) time

Ultimately, it is our hope that all vendors will sell cluster nodes with LinuxBIOS pre-installed. Currently two such vendors exist (Linux Labs and Linux Networx), and only one (Linux Labs) offers BProc extensions to the installed operating system. For Ed, LinuxBIOS had to be installed manually which involved setting up a standard network boot on each node. Once the nodes came up as part of the cluster, the Flash RAM was overwritten with LinuxBIOS from the front-end node over the network. *This was the first and only time we had to write the Flash because the operational kernel is downloaded from the front-end node.*

nodes	*boot time*	*time per additional node*
1	69 seconds	N/A
10	71 seconds	0.22 seconds
100	101 seconds	0.32 seconds

Table 1. Boot time from power-on to node ready for one, ten, and 100 DS10 nodes and the time to boot each additional node.

Once a node has been booted by LinuxBIOS, it contacts the front-end node for the operational kernel and reboots itself. Upgrading the cluster nodes is achieved by building a new operational kernel on the front-end and rebooting the cluster nodes.

In the previous section, we mentioned the current belief that hierarchical structure is necessary for clusters larger than 100 nodes. We found that using multicast capabilities changed our view on hierarchical structure.

Table 1 shows the time to boot one, ten, and 100 DS10 nodes and the time to boot each additional node. Using a faster network, we believe these boot times will improve to better than 8 minutes (based on extrpolation from the current numbers). As a reference point, prior to adding multicast functionality, we were unable to boot 100 nodes simultaneously due to time outs induced by excessive load on the network and the front end machine. Another 128-node Alpha cluster at LANL running Tru64 cannot be booted in less than two hours.

3.2. Job startup time

Job startup time is crucial to overall application performance. Consider the case of a 10-hour sequential job running on a cluster with 100 nodes. To achieve the full capability of the cluster, scalar overheads should be less than 1/100 the ideal parallel runtime, *i.e.*, perfect speed-up. Since the ideal parallel runtime is 6 minutes, the limit on scalar overhead is 3.6 seconds. Yet we have clocked MPI startup times on a cluster this size at 15 seconds. The problem only gets worse as we grow the cluster to 1024 nodes, which would in turn limit scalar overhead to less than 1/2 second.

Table 2 compares remote job fork using a flat (sequential) or tree-based startup for up to 15,000 processes on 120 nodes. For small numbers of small processes, a linear start-up is faster. This is partly due to the fact that the front end machine is connected via

job size (processes)	*process size*	*linear spawn*	*tree spawn*
10	150k	0.03	0.06
100	150k	0.36	0.18
1000	150k	5.00	0.24
10000	150k	63.00	1.54
15000	150k	81.00	2.40
10	5MB	0.81	1.5
100	5MB	7.40	4.8
1000	5MB	90.00	4.8
10000	5MB	837.00	4.9

Table 2. Job startup times in seconds. Linear spawn starts every process on the front end and migrates it to a node. Tree spawn replicates processes in a tree structure to exploit parallelism in the network.

Gigabit Ethernet and the nodes are connected via fast Ethernet. The tree spawn's time becomes basically flat once the number of processes reaches the number of nodes in the system. This is because the tree spawner uses local fork once it gets a single process on a node. Each one of those forks creates a new child which must be represented on the front end machine so there is communication for each one of those forks. Due to the apparent lack of overhead there, we believe the tree based job startup will scale very well to large (thousands) numbers of machines.

Notice that while job startup is hierarchical, the cluster structure itself does not need to be hierarchical.

In previously published work, it was shown that Supermon can collect 100 samples per second without noticeably affecting node performance [9]. More recently, improvements to the data protocol and node data source drastically increased this efficiency [12]. Supermon is composed of three levels through which data is filtered and passed back to clients. The lowest level, a kernel module providing additional entries in `/proc`, provides the raw monitoring data at a maximum sampling rate of 3500 Hz on Ed's nodes (up to 6600 Hz on Intel platforms). The single node TCP data server (mon) takes this data, filters it, and passes it to clients at a maximum sampling rate of 1400 Hz. Finally, a data concentrator (supermon) collects data from mon servers and returns it to clients over TCP at a maximum sampling rate of 750 Hz for a single node. These rates, shown in Figure 4, reflect the entire 1K per node data set being transferred, and are higher for filtered subsets of this data.

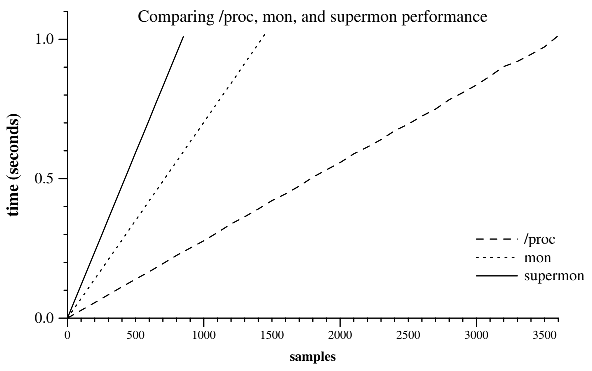


Figure 4. Comparison of /proc, mon, and supermon sampling rates for single node monitoring

3.3. Edb scalability

To demonstrate the scalability of edb, we measured its response time for startup and continuation of 15,000 processes. Startup includes distribution and process initialization of the target application and continuation includes the issue and return of a ptrace continue command for all processes. For an interactive debugging session with 15,000 processes, we measured 40 seconds for startup and 10 seconds for continuation.

4. Conclusions and Future Work

The current state-of-the-art technology for building cluster computers will not be usable for terascale-and-beyond systems due to the direct relationship between cluster size and cluster management and programming overhead. In this paper, we have presented our experience with a prototype Science Appliance, Ed, a testbed for future generation cluster computers. We have shown that LinuxBIOS, a replacement for the standard PC BIOS, can be used as a network boostrap to bring up the cluster nodes. The front-end node provides a single-system image of the cluster, much like a traditional supercomputer, and thus the nodes can be conveniently and efficiently managed from the front-end. Our Science Appliance can be booted in under two minutes, which means that it can be upgraded in under two minutes. The front-end can support 15,000 processes which can be started on the computed nodes in 2.4 seconds, using a simple tree-based startup. Finally, we have built two clusterware tools, Supermon and edb, which also scale well to large clusters.

Our initial experiences with Ed have shown us that hierarchical structure, already being used in production clusters today, may not be necessary for clusters up to 1000 nodes if they are built in the way of Ed. Notice that hierarc hy is neededfor job startup, but this does not require hierarchical structure.

Compared to LANL's other experiences with v endor supported clusters of similar size and make, Ed is several orders of magnitude easier to manage and maintain. Consequently, we can only advocate building terascale-and-beyond clusters as Science Appliances.

4.1. Future Work

By the prin tingof this paper, Ed's high speed interconnect will hav e been replaced with Myrinet. We expect to see even more impressive numbers for boot time and start up using Myrinet (versus 100 MB Ethernet). Our next steps with this cluster will include additional scaling w ork, a scheduler, and file system research.

Lighter weight boot process The current boot process inv olv esrunning scripts on the fron t-end to setup the nodes. These scripts are currently complex and involve a fair amount of processing per node on the fron t-end. Replacing these scripts with equivalen t programs which can do the same work on the slave node instead of the front-end would considerably lighten the fron t-end's load at boot time.

BProc Job Scheduler BProc's single-system image requires a new job scheduler. The interesting aspect of the sc heduler is that the nodes hav e attributes similar to files and th us access to the nodes can be trivially managed from the fron t end. Unlike existing cluster schedulers, there is no need to contact the individual nodes. In addition, fast boot times open up other interesting possibilities such as rebooting the nodes after each job for securit y reasons or booting different kernels for different jobs.

Private Name Spaces Current research in cluster file systems attempt to provide a global file system for the entire cluster. Our belief is that this is inherently unscalable. We are currently implementing Plan 9 [10] style Priv ateName Spaces for Lin ux[6, 7]. We will incorporate this into the next Science Appliance environment.

References

[1] Argonne National Laboratory. City - The MCS Large Cluster System Software Toolkit.

[2] G. Bruno and P . M. P apadopoulos. NPA CI Rocks: T ools and Techniques for Easily Deploying Manageable Linux Clusters. In *Proceedings of Cluster 2001*, Anaheim, CA, October 2001.

[3] E. A. Hendriks. The Beowulf distributed process space. *Submitted for public ation*, 2002.

[4] Intel Corporation. Preboot execution en vironment (PXE) specification.

[5] A. J. Marks. T ow ard the design and implementation of a scalable debugger. (In preparation), 2001.

[6] R. Minnich. 9.2.u: A user-mode private name space system for unix. Technical report, Defense Advanced Research Projects Agency, September 1998.

[7] R. Minnich. Private namespaces for linux. *Dr. Dobb's Journal*, December 2001.

[8] R. Minnich, J. Hendricks, and D. Webster. The Linux BIOS. In *Proceedings of the F ourth Annual Linux Showcase and Conference*, Atlan ta, GA, October 2000.

[9] R. Minnich and K. Reid. Supermon: High performance monitoring for linux clusters. In *Pr oceedings of the Fifh A nnual Linux Showcase and Conference*, Oakland, CA, November 2001.

[10] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom. Plan 9 from Bell Labs. *Computing Systems*, 8(3):221–254, 1995.

[11] R. Riesen, R. Brightwell, L. A. Fisk, T. Hudson, J. Otto, and A. B. Maccabe. Cplant. In *Pr oceedings of the Second Extreme Linux Workshop*, June 1999.

[12] M. J. Sottile and R. G. Minnich. Supermon: Cluster monitoring as if performance mattered. *Submitted for public ation*, 2002.

[13] The Open Cluster Group. OSCAR: A pack aged cluster softw are stac k for high performance computing. January 2001.

[14] T urboLinux. Turbolinux P owerCockpit T echnology Overview (white paper). September 2001.