On the Appropriateness of Commodity Operating Systems for Large-Scale, Balanced Computing Systems

Ron Brightwell
Scalable Systems Integration Department
Sandia National Laboratories*
PO Box 5800
Albuquerque, NM 87185-1110
bright@cs.sandia.gov

Arthur B. Maccabe
Computer Science Department
University of New Mexico
Albuquerque, NM 8731-1386
maccabe@cs.unm.edu

Rolf Riesen
Scalable Computing Systems Department
Sandia National Laboratories*
PO Box 5800
Albuquerque, NM 87185-1110
rolf@cs.sandia.gov

Abstract

In the past five years, we have been involved in the design and development of Cplant^{tm}. An important goal was to take advantages of commodity approaches wherever possible. In particular, we selected Linux, a commonly available operating system, for the compute nodes of Cplant^{tm}. While the use of commodity solutions, including Linux, was critical to the success of Cplant^{tm}, we believe that such an approach will not be viable in the development of the next generation of very large-scale systems.

We present our definition of a balanced system and discuss several limitations of commodity operating systems in the context of balanced systems. These limitations are categorized into technical limitations (e.g., the structure of the virtual memory system) and social limitations (e.g., the kernel development process). While our direct experience is based on Linux, issues we have identified should be relevant to all commodity operating systems.

1. Introduction

With the introduction of Beowulf systems [18] in 1995, commodity solutions have become popular in high performance computing systems. The early Beowulf systems demonstrated that a large number of computations which were previously run on special purpose supercomputers could be run on systems built from commonly available components at a fraction of the cost. Since their introduction, developers have explored many variations on the Beowulf theme. The "Stone Souper" project [9] which was assembled entirely from PCs that had been scheduled for reapplication, represents one extreme. Cplant^{tm} [4], the Los Lobos Cluster at the Albuquerque High Performance Computing Center, the NSF Terascale Cluster at the Pittsburgh Supercomputer Center, machine, and the NSF Distributed Terascale Facility (DTF) [2], which combine high-end commodity compute nodes with specialized networks (e.g., Myrinet), represent another extreme.

While Beowulf systems spurred a great deal of interest in commodity solutions; by 1995, the use of commodity operating systems in high-performance computing systems was already well established. In 1993, rather than continue development of the NX [15] operating system, Intel adopted OSF/1 AD [20] as the operating system to run on all of the nodes of the Paragon. OSF/1 AD, based on the Mach microkernel [16], represents an early example of adapting a commodity operating system to the compute nodes of a massively parallel systems. In 1994, the NOW [1] project at UC Berkeley took a different approach. Rather than adapting the operating system, GLUnix [14, 6] was designed to run on top a commodity operating system without making any modifications to the underlying operating system.

As with many early systems, there were a number of

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

problems in the initial deployment of OSF/1 AD on the Intel Paragon. These problems led Sandia management to deploy SUNMOS (the Sandia/UNM Operating System) [11], a "light-weight," custom operating system, on the compute nodes of their 1842 node Intel Paragon. The success of this effort led to the development of Puma/Cougar [17], the compute node operating system for the Intel Tflops system and, in particular, on the 9000 processor ASCI/Red machine at Sandia National Laboratories.

In 1997 Sandia National Laboratories embarked on Cplant[tm], a project aimed at building a massively parallel computing system from commodity, or near-commodity components. As we will discuss, there were several issues that lead us to adopt Linux as the compute node operating system for Cplant[tm]. Overall, our use of Linux in this environment should be viewed as a success; however, as we look to future systems, our experiences with Linux have led us to conclude that the direct use of Linux will not be appropriate in these systems.

The remainder of this paper is organized as follows: The next section provides additional background for the rest of this paper. In particular, this section defines the basic system architecture for the systems we are interested in, the class of applications we are interested in, and the reasons that we selected a commodity operating system for Cplant[tm]. Section 3 presents our definition of a balanced system. Section 4 considers technical issues that we have encountered in using commodity operating systems. Section 5 considers the social issues associated with using commodity operating systems in high-performance computing environments. Finally, Section 6 summarizes our concerns with the appropriateness of commodity operating systems for high-performance computing.

2. Background

Before presenting our definition of balanced system and describing the issues we have encountered with commodity operating systems, we need to describe the context for our work.

2.1. Target Architecture

Our target system architecture partitions the nodes of a massively parallel system based on functional considerations [8]. Typically, we partition the nodes into three sets: compute nodes, service nodes, and I/O nodes.

The compute partition is dedicated to delivering processor cycles and interprocessor communication for parallel applications. Nodes in the compute partition are managed in a space-shared fashion. That is, a group of nodes is dedicated to running only a single application.

Nodes in the service partition provide access to the compute partition. As a minimum, nodes in the service partition support user logins (authentication) and application launch. More typically, these nodes also perform a variety of other activities such as compiling codes, editing files, sending email, and checking the status of the nodes in the compute partition.

The I/O partition provides access to a global parallel filesystem, and may also contain nodes that provide access to secondary storage systems or high performance network interfaces to other systems.

In the context of systems software design, the primary advantage of partitioning is that it allows the system software to be tailored to specific needs. For example, compute nodes only require a small subset of the functionality provided by common operating systems.

2.2. Target Applications

In designing massively parallel computing systems, our primary goal is to provide support for *resource constrained applications*, applications that can scale to consume all of at least one type of resource (e.g., memory, processing, IPC communication, I/O communication, etc.) provided by the system.

In considering these applications, the primary concern is execution time. A single run of a resource constrained application may use the full system (in dedicated operation) for several days. Many of these applications have extensive code to manage the resources provided by the machine. Examples of resource constrained applications include: climate modeling, fire simulation, and traffic simulation.

2.3. Why Linux for Cplant[tm]

Linux was not our first choice as the compute node operating system for Cplant[tm]. Based on our experiences with the Paragon and Tflops systems we decided that the compute nodes should run a light weight kernel, like Puma, while the service and I/O nodes would run a full fledged operating system. Because Puma had not been ported to the Alpha processor, we decided that we would start with a system that ran the same OS on all partitions with the understanding that we would eventually port Puma to the Alpha architecture and would then replace Linux on the compute nodes with a light weight kernel.

Access to the source code was an important consideration in selecting the operating system that would run on the service and I/O nodes. While we did not expect to make significant modifications to the OS running on the service and I/O partitions, we would need to modify this OS to enable communications with the compute node OS. In addition to wanting source code access for our own work, we felt it

would be important that others also have access to the code to enhance collaborations. This effectively ruled out commercial operating systems such as Tru64 and OSF/1 AD.

Given this context, we started looking at open source projects. Several of us had used Linux on our personal workstations and were familiar with it from a user point of view. At the time Linux, especially the RedHat distribution, was the most advanced open source project supporting the Alpha processor. DEC (now Compaq) had given Linus Torvalds a development machine early on and dedicated two engineers in helping to port Linux to the chipsets used in its machines.

Linux was evolving rapidly and the community of kernel developers grew almost daily. Together with DEC's support of Linux, this promised continued and increasing support even though Alpha processor based systems were not as common as the Intel based systems. Choosing Linux also made it possible to move Cplant^{tm} to other architectures later on, whereas an operating system which only runs on a few select systems, would have imposed limitations in future hardware selections.

Using an operating system which already ran on our platform of choice meant that we could devote our limited resources to the message passing infrastructure and the system software which turns a group of PCs into a single supercomputer.

Linux is a monolithic kernel, but through its support for kernel modules, it is fairly easy to extend. This is especially important during development where this capability lets us change a running kernel without having to reboot for every small change.

After five years of development on Cplant^{tm}, we are finally in the process of porting our lightweight operating system, Puma, to the compute nodes of Cplant^{tm}. Interestingly, we view this more as an experiment for the development of future systems rather than something that will become part of the production Cplant^{tm} system.

In retrospect, three factors led to our selection and continued use of Linux on the compute, service and I/O nodes on Cplant^{tm}. First, Linux was available on a wide variety of systems, including the DEC Alpha systems we were using in Cplant^{tm}. Second, by using Linux we would have access to the source code and could easily modify it to suit our needs. Third, and perhaps most importantly, we would be part of a growing community of developers who were actively developing high-performance computing systems based on Linux.

3. Balanced Systems

In this section we introduce the notion of a balanced system to motivate the discussion of commodity versus custom operating systems. It is our contention that once balance in a system has been compromised at a low level, attempts to overcome the imbalance at a higher level have little or no impact. For example, a high-performance operating system may have little impact on application performance in a machine that has an imbalance at the hardware level.

For most scientific and engineering applications, the performance of a large-scale parallel computing platform is determined by the parallel efficiency of the entire system and not by the peak performance of the individual processors in the machine. In order to achieve and maintain a high degree of parallel efficiency, there must be a balance over the entire system, including: the processors, the memory subsystem, the interconnect, the I/O subsystem, and the systems software. In this section, we describe the hardware characteristics of a computing platform required to provide scalable performance to thousands of processors for a large collection important applications.

The criteria we describe are based on experience in evaluating the performance of real applications on production platforms. Evaluation in terms of real applications is extremely important, especially when considering that some advanced features of a platform cannot be reliably exploited by all applications. In the interest of brevity, we only consider the balance between the processor(s), memory bandwidth, and network bandwidth. Other factors, such as memory size, memory latency, network latency, and processor availability are also considerations. Below we consider peak memory bandwidth and peak network bandwidth as a ratio to the peak CPU floating-point operations per second (FLOPS).

3.1. Memory Bandwidth

The bandwidth between the processor(s) and the main memory must be sufficient to feed data to the processors. In our experience, our important large-scale applications do not benefit from caching. Because they are floating-point bound, there is little excess time to fill or empty caches. As such, we would like the memory system to be capable of two loads and one store per floating-point operation. This suggests that the ideal ratio would be 24 B/s per FLOPS. Recognizing that modern architectures use registers because they are unable to sustain such a memory rate, we relax this criteria to 4 B/s per FLOPS, which is a more realistic goal for modern computing systems. As an example, if the processor is capable of delivering a peak performance of 2 GFLOPS, the memory system should be able to deliver 8 GB/s.

We have observed the importance of this ratio in several real systems. As an example, the compute nodes for an early Cplant^{tm} used 500 MHz Alpha EV56. These nodes have a peak of 1 GFLOPS and 800 MB/s memory bandwidth, i.e., a ratio of .8. The compute nodes of a subsequent

Cplant™used 500 MHz Alpha EV67 nodes. These nodes also have a peak of 1 GFLOPS; however, the memory subsystem is capable of delivering 2.6 GB/s of memory bandwidth. Most applications saw at least a 50% performance improvement and several applications saw a factor of two increase in performance.

3.2. Network Bandwidth

Balance in peak network performance is also critical for sustained parallel efficiency. This is especially important in large-scale systems where compute nodes will need to be interconnected by a high-performance networking fabric. We consider three aspects of network bandwidth: bi-directional bandwidth for a single compute node, link bandwidth within the network, and bisection bandwidth.

Bi-directional bandwidth is the maximum bytes per second that can be communicated into and out of a single node. While the desired ratio is highly dependent on the structure of the application and, in some cases dependent on the application's data set, a ratio of 1.5 B/s per FLOPS is adequate to allow efficient scaling for most of our applications. For a compute node capable of delivering 2 GFLOPS, network bandwidth into the processor should be at least 3 GB/s.

Since links within the network fabric are typically shared, the link bandwidth needs to be greater than the bi-directional bandwidth of a single node to accommodate network traffic between multiple compute nodes. Our balance criteria for link bandwidth is 2 B/s per FLOPS, e.g., 4 GB/s for a compute node with a peak performance of 2 GFLOPS.

Bisection bandwidth is a succinct characterization of the network topology. In particular, the ability of the network to support multi-node communication, including collective operations. The balance metric for bisection bandwidth is 0.75 B/s per nodes $\times$ FLOPS, e.g., 1.5 TB/s for 20,000 1 GFLOPS processors.

3.3. A Comparative Example

Table 1 shows the network bi-directional link bandwidth balance ratios for several large-scale computing platforms. The ASCI/Red machine originally contained 200 MHz Intel Pentium Pro processors when it was deployed in 1997, but was subsequently upgraded to 333 MHz Intel Pentium II processors in 1999. The numbers for Cplant™ are for a system based on 466 MHz Compaq Alpha EV67 processors and 2.4 Gb/s Myrinet links. The numbers for the Platinum cluster at NCSA are based on dual 1 GHz Intel Pentium III processors and Myrinet 2000, which has 4 Gb/s links.

It is interesting to note that the first two platforms, ASCI/Red and the Cray T3E, both run lightweight compute node operation systems: Puma/Cougar on ASCI/Red and Unicos/Mk on the T3E. The remaining systems use full-featured, UNIX-based operating system: Linux on Cplant™ and Platinum, AIX on ASCI/White, and Tru64 UNIX on ASCI/Q.

Table 1. Comparison of Link Bandwidth / MFLOPS

Machine	Node Peak (MFLOPS)	Link BW (MB/s)	Ratio
ASCI/Red (1997)	400	800	2.0
ASCI/Red (1999)	666	800	1.2
Cray T3E	1200	1200	1.0
Cplant™ (Alaska)	1000	300	0.33
Cplant™ (Antarctica)	932	300	0.32
NCSA Platinum	2000	500	0.25
ASCI/Q	8000	1360	0.17
ASCI/White	24000	2000	0.083

4. Technical Issues

In this section, we consider the technical issues associated with using full-featured, commodity operating systems on the compute nodes of a large-scale, balanced system.

4.1. Blocking System Calls

Most Linux system calls are blocking. That is, the requesting process is blocked until the kernel provides the requested service. Historically, blocking system calls were introduced to support efficient multiprogramming. While the blocked process is waiting for an I/O operation to complete, the OS can run another process, increasing the overall system utilization and throughput.

While this approach is appropriate in a full-featured, commodity OS, it is contrary to the goals of a compute node operating system. From the space-sharing model we can assume that the resources of a compute node are dedicated to a single application and, as such, any competition for the CPU is limited to a single application. Moreover, because they are very aware of resource management issues, many resource constrained applications are designed to perform meaningful work while they are waiting for slow requests to be completed, i.e., they can overlap computation and I/O.

4.2. Daemon Processes

Daemon processes, represent another example of an inappropriate resource management strategy commonly used in full-featured, commodity operating systems. Modern operating systems delegate many of their resource management activities to daemon processes that run periodically. Examples include daemons that provide network services

(e.g., httpd), printing services (e.g., lpd), and periodic execution services (e.g., atd). This approach can introduce a great deal of variability in execution times depending on how many times the daemon processes run.

This variability introduces two problems for resource constrained applications. First, this variability makes it difficult to obtain the accurate timing information needed to tune application performance. Second, it can have a significant impact on tightly coupled applications in which activities on the compute nodes are highly synchronized. This problem is exacerbated in very-large scale systems. While it is possible to remove most of the traditional, user-level daemons from a commodity operating system, recently there has been a trend toward introducing daemons in the OS kernel, e.g., the kswap daemon in the Linux kernel which manages the pool of free page frames for the virtual memory system.

4.3. File Systems (Exec)

Modern operating systems, especially Unix derivatives, are filesystem centric. This is most easily seen in the way that applications are loaded into memory. Like most Unix derived operating systems, Linux uses the "fork-exec" model to load processes. According to this model, a new process is created using the fork system call. The new, child, process is clone of the parent. To execute a different program, the child uses the exec system call. The first argument to exec is a string which has the name of the file containing the executable image to be loaded over the current process image. While this model has worked well for general purpose environments, it is at odds with our usage model for compute nodes.

The compute nodes of Cplant^{tm} do not have disks and, as such, they do not have a "natural" local file system. The technology for "diskless" workstations is well established and involves the use of NFS (network file system) to let the compute nodes *pull* the image from a remote file server. The NFS pull strategy works well in a general network environment because there is little synchronization among the workstations that are using NFS to load applications. When an application is launched on thousands of nodes and they all try to read the image file from the same NFS server, this strategy fails spectacularly.

There are several solutions to this problem. Our approach is to *push* the executable image to the compute nodes (using a spanning tree) as the application is launched. This strategy is both scalable and efficient [3], but it does not map directly to the exec model. To make our push strategy work with the exec model, each compute node has a small RAM disk. When the compute node receives a process image (through the message passing system), it writes the image to the RAM disk and starts the fork-exec sequence. This is not a major inconvenience; however, we have had to adjust the size of the RAM disk to accommodate very large applications.

4.4. Virtual Memory

Even though memory is relatively inexpensive, it still tends to be the resource that generates the most contention among the processes running on a desktop or server system. As such, commodity operating systems tend to place a great deal of emphasis on their implementation of demand-page virtual memory. These implementations assume that processes come and go fairly often and that their access to memory pages is fairly dynamic. In contrast, the compute nodes of a massively parallel computing system have highly static memory access patterns. Upon booting these nodes launch a daemon process to manage the node. This process responds to "ping" requests and launches applications (this is the process that receives process images and initiates the fork-exec sequence). Because the compute nodes are space-shared, applications are run serially. Because the nodes do not have local disks, we do not support demand paging. In essence, the memory allocation pattern is a stack with at most two processes at any time.

While having a complex virtual memory system does not, in and of itself, have a direct impact on application scalability, it does limit our ability to take advantage of the relatively simple memory access patterns on compute nodes. For example, on the Intel Paragon switching the page sizes for applications from 4KB to 4MB decreased run times by approximately 25%. Although we do not have direct performance data, our communication layers would be greatly simplified if we could assume that logically contiguous addresses mapped to physically contiguous addresses. Given the complexity of the Linux virtual memory system it would be very difficult to make the modification needed to make this assumption.

4.5. Communication

High-performance, zero-copy communication is also critical to the performance of a large-scale massively parallel computing platform. While there have been several efforts to develop true zero-copy networking [10, 5, 7], none of these has had a significant impact on Linux kernel development. Within the Linux community, zero-copy is focused on the ability to move disk blocks directly to the network interface without the need to copy data to and from application space [13]. Importantly, Linux kernel developers seem focused on the needs of servers to send efficiently and have largely ignored to difficulties with zero-copy receiving. This asymmetry is inappropriate for HPC communication requirements.

4.6. Time to Solution

In many instances, time to solution is an important reason for selecting an commodity approach. In the case of Cplant[tm], Linux provided most of the solution that we needed and we were able to adapt it to serve our purposes in a timely fashion. However, there are other cases where the use of commodity components actually increase the time to solution. In 1993, the developers of OSF/1 AD on the Intel Paragon faced the exec problem that we described earlier. Rather than use the push approach that we have used, the developers of OSF/1 AD chose to implement paging trees [12]. While this solution is quite elegant; by the time it was implemented, we had ported SUNMOS to the Paragon and were running applications on all 1842 nodes.

As a second counter example, we consider the addition of "virtual node" mode to Puma/Cougar (the lightweight OS on ASCI/Red). Like the Intel Paragon, the Intel ASCI/Red architecture has two processors per node. In concept, one processor is supposed to be used for running applications while the other is used to manage the network. In Puma, we call this mode "message co-processor" mode. Puma also provides "compute co-processor" mode, in which one processor is used to manage the network and run application code while the other processor is used as a co-processor. Based on demand from application developers, a new mode, "virtual node" mode, was added to Puma. In this mode, each processor is treated as a virtual node. Two programmers working for less than six months were able to extend Puma to add this mode. This is even more impressive when we consider that neither of these programmers had had prior experience in Puma development.

5. Social Issues

In this section, we consider social issues surrounding the use of commodity operating systems on the compute nodes of a large-scale, balanced system. In some cases, these issues can be more challenging to overcome than the technical issues above.

5.1. Staying Current

As stated in Section 2.3, being part of a growing community actively developing high-performance computing systems based on Linux was an important factor in our choice of Linux for Cplant[tm].

In reality, the development surrounding the Linux kernel has moved too quickly for us to be able to keep pace. Being part of the Linux development community means that, to some extent, you have to move at the pace of Linux. The earliest Cplant[tm] clusters continued to run a Linux 2.0 kernel long after 2.2 was available. We finally moved to using a 2.2 kernel, not because of any enhancements in the later version, but because we purchased nodes whose PCI chipset was only supported by the 2.2 kernel.

We are currently experiencing the same situation in moving from 2.2 to 2.4. However, this time there are several features in 2.4 that we would like to use, but there are also features that we would like to avoid. Since Linux is monolithic, we are forced to take the bad with the good.

Consider the example of the the changes to the virtual memory subsystem that occurred in the middle of the Linux 2.4 kernels. We had been working closely with a third party software vendor to add support for using larger memory pages in the VM system of the 2.2 kernel. By the time we were able to evaluate the performance implications of the enhancement in the 2.2 kernel, the 2.4 kernel had been released. We received an estimate for adding this functionality into the 2.4 kernel, but we did not immediately go forward with the enhancements. Luckily, the change to the VM system in 2.4 happened before we contracted to have the proposed work done. Had the work already begun, it would have been made obsolete by the new 2.4 VM system.

In addition to the kernel proper, we also needed to keep pace with a particular Linux distribution, which in our case was RedHat. We continually find ourselves being forced into using a later distribution sooner than we would like.

In short, a compute node operating system has a fairly fixed set of requirements that need to be met. We find the rapidly advancing functionality and enhancements in Linux and the Linux development environment does not allow us the time needed to concentrate on optimizing a particular snapshot of the system software. Slowing down to focus on a particular version or release has the potential to make us unique within the community, which directly counters one of the main advantages we saw in using Linux.

5.2. Staying Focused

Another drawback to being part of the Linux community is that they are not focused on solving problems related to high-performance parallel computing systems. The trend for Linux, and to some extent other commodity operating systems as well, has been toward the desktop and server markets. While there is some commonality, to a large extent the problems of HPC are unique, especially for our target architecture.

We discovered that the sheer number of development projects surrounding Linux became a problem. Rather than first implementing something ourselves, we had to spend resources evaluating what others were working on. In a few cases we were able to save some resources by leveraging previous or ongoing work. However, in nearly all cases, our specific needs and requirements were not being addressed, so the evaluation effort was largely unproductive.

There are several examples of functionality that is continually being added to Linux which directly contradicts the environment that we would like to support on our system. For example, consider the Out-Of-Memory (OOM) killer [19]. This feature is used to protect the Linux kernel from becoming unresponsive due to exhaustion of virtual memory resources. When virtual memory becomes threateningly scarce, the OOM killer selects a process to eliminate to free up memory resources. The main criteria that the OOM killer uses in determining which process to kill is memory use – the user process consuming the most memory will be killed. This particular example demonstrates a fundamental difference in philosophy between a desktop or server operating system and the operating system on the compute node of a parallel machine. For the compute node, we expect processes to consume all of the resources on a node and design the operating system to allow and encourage that.

Efforts to steer Linux development toward addressing HPC problems have largely been unsuccessful. The now defunct Extreme Linux Forum, which was started by researchers at Los Alamos National Laboratory, was an effort founded to do just that. While Extreme Linux did provide a forum for the HPC community to exchange ideas and approaches, it had little real impact on the direction of Linux.

6. Summary and Future Directions

In this paper we have discussed some of the characteristics of a balanced, large-scale parallel computing system. We believe that the balance of the entire system with respect to the major architectural components is the key to achieving overall system performance for a variety of scientific and engineering applications. We view the system software as one of these important components.

As we noted in the introduction, we consider our use of Linux in Cplant^{tm}to be a success. To a large extent this is because the base hardware for Cplant^{tm} is not balanced (see Table 1). In essence, Linux did not increase the damage done by the base hardware. For platforms whose hardware architecture already imposes a restriction on the balance of the system, it is less critical that the system software be optimized to deliver to the maximum capability of the hardware. In contrast, for platforms whose hardware architecture exhibits good balance characteristics, system software becomes a critical factor in determining the overall efficiency of the system.

What are the costs if we move away from commodity operating systems and back to specialized operating systems for the compute nodes of a massively parallel system? The most significant costs include development time, cross platform porting, limitations to functionality, and potential loss of third party tool support. While these costs are not insignificant, we must compare them to the costs associated with adapting commodity solutions. Moreover, we must take into consideration that commodity operating systems are drifting further and further from the needs of HPC systems.

It is important to keep in mind that, in contrast to commodity operating systems, our goal is to develop a solution for a limited set of platforms and applications. As such, our development and ongoing support costs are much less. While a commercial operating system may involve the effort of hundreds of person years and millions of dollars for supporting a vast array of hardware and applications, our costs for developing and maintaining specialized operating systems have been much more modest.

References

[1] T. E. Anderson, D. E. Culler, and D. A. Patterson. A case for NOW (network of workstations). *IEEE Micro*, 15(1):54–64, Feb. 1995.

[2] F. Berman. Viewpoint: From TeraGrid to knowledge grid. *Communications of the ACM*, 44(11):27–28, Nov. 2001.

[3] R. Brightwell and L. A. Fisk. Scalable parallel application launch on Cplant(tm). In ACM, editor, *SC2001: High Performance Networking and Computing. Denver, CO, November 10–16, 2001*, New York, NY 10036, USA and 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 2001. ACM Press and IEEE Computer Society Press.

[4] R. B. Brightwell, , L. A. Fisk, D. S. Greenberg, T. B. Hudson, M. J. Levenhagen, , A. B. Maccabe, and R. Riesen. Massively parallel computing using commodity components. *Parallel Computing*, 26:243–266, February 2000.

[5] A. Gallatin, J. Chase, and K. Yocum. Trapeze/IP: TCP/IP at Near-Gigabit Speeds. In *Proceedings of the FREENIX Track (FREENIX-99)*, pages 109–120, Berkeley, CA, June 1999. USENIX Association.

[6] D. P. Ghormley, D. Petrou, S. H. Rodrigues, A. M. Vahdat, and T. E. Anderson. GLUnix: A Global Layer Unix for a network of workstations. *Software–Practice and Experience*, 28(9):929–961, July 1998.

[7] P. Gilfeather and T. Underwood. Fragmentation and High Performance IP. In *Proceedings of the 2001 Workshop on Communication Architecture for Clusters*, April 2001.

[8] D. S. Greenberg, R. Brightwell, L. A. Fisk, A. B. Maccabe, and R. Riesen. A system software architecture for high-end computing. In ACM, editor, *SC'97: High Performance Networking and Computing: Proceedings of the 1997 ACM/IEEE SC97 Conference: November 15–21, 1997, San Jose, California, USA.*, New York, NY 10036, USA and 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1997. ACM Press and IEEE Computer Society Press.

[9] W. W. Hargrove, F. M. Hoffman, and T. Sterling. The do-it-yourself supercomputer: Scientists have found a cheaper way to solve tremendously difficult computational problems: connect ordinary PCs so that they can work together. *Scientific American*, 285(2):72–79, Aug. 2001.

[10] C. Kurmann, M. Müller, F. Rauch, and T. M. Stricker. Speculative Defragmentation–A Technique to Improve the Communication in Software Efficiency for Gigabit Ethernet. In *Proceedings of the 9th International Symposium on High Performance Distributed Computing (HPDC)*, August 2000.

[11] A. B. Maccabe, K. S. McCurley, R. Riesen, and S. R. Wheat. SUNMOS for the Intel Paragon: A brief user's guide. In *Proceedings of the Intel Supercomputer Users' Group. 1994 Annual North America Users' Conference.*, pages 245–251, June 1994.

[12] D. S. Milojicic, D. L. Black, and S. Sears. Operating system support for concurrent remote task creation. In IEEE, editor, *Proceedings 9th International Parallel Processing Symposium, Santa Barbara, CA, USA, April 25–28, 1995*, pages 486–492, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1995. IEEE Computer Society Press.

[13] V. S. Pai, P. Druschel, and W. Zwaenepoel. IO-Lite: A Unified I/O Buffering and Caching System. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 15–29. USENIX Association, February 1999.

[14] D. A. Patterson and T. P. Anderson. GLUnix: A New Approach to Operating Systems for Networks of Workstations. In *Proceedings of the First Workshop on Networks of Workstations*, San Jose, Oct. 1994.

[15] P. Pierce. The NX/2 operating system. In *3rd Conference on Hypercube Concurrent Computers and Applications*, volume I, Architecture, Software, Computer Systems and General Issues, pages 384–390, Pasadena, CA, Jan. 1988. ACM. Intel.

[16] R. Rashid, R. Baron, A. Forin, D. Golub, M. Jones, D. Orr, and R. Sanzi. Mach: a foundation for open systems (operating systems). In IEEE, editor, *Workstation operating systems: proceedings of the Second Workshop on Workstation Operating Systems (WWOS-II), September 27–29, 1989, Pacific Grove, CA*, pages 109–113, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1989. IEEE Computer Society Press.

[17] L. Shuler, C. Jong, R. Riesen, D. van Dresser, A. B. Maccabe, L. A. Fisk, and T. M. Stallcup. The Puma operating system for massively parallel computers. In *Proceeding of the 1995 Intel Supercomputer User's Group Conference*. Intel Supercomputer User's Group, 1995.

[18] 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, Architecture, pages I:11–14, Boca Raton, FL, Aug. 1995. CRC Press.

[19] R. van Riel. Linux-MM docs: the OOM killer. `http://linux-mm.org/docs/oom-killer.shtml`.

[20] R. Zajcew, P. Roy, D. Black, C. Peak, P. Guedes, B. Kemp, J. LoVerso, M. Leibensperger, M. Barnett, F. Rabii, and D. Netterwala. An OSF/1 UNIX for massively parallel multicomputers. In *Proceedings of the 1993 Winter USENIX Technical Conference*, pages 449–468, Jan. 1993.